Extreme®
Connect Beyond the Network

ExtremeWireless™ Integration Guide

V10.01.02

9034918

Published January 2016

Legal Notice

Extreme Networks, Inc. reserves the right to make changes in specifications and other information contained in this document and its website without prior notice. The reader should in all cases consult representatives of Extreme Networks to determine whether any such changes have been made.

The hardware, firmware, software or any specifications described or referred to in this document are subject to change without notice.

Trademarks

Extreme Networks and the Extreme Networks logo are trademarks or registered trademarks of Extreme Networks, Inc. in the United States and/or other countries.

All other names (including any product names) mentioned in this document are the property of their respective owners and may be trademarks or registered trademarks of their respective companies/owners.

For additional information on Extreme Networks trademarks, please see: www.extremenetworks.com/company/legal/trademarks/

Support

For product support, including documentation, visit: www.extremenetworks.com/documentation/

For information, contact:
Extreme Networks, Inc.
145 Rio Robles
San Jose, California 95134
USA

Table of Contents

Preface

Text Conventions

The following tables list text conventions that are used throughout this guide.

Table 1: Notice Icons

Icon	Notice Type	Alerts you to...
	General Notice	Helpful tips and notices for using the product.
	Note	Important features or instructions.
	Caution	Risk of personal injury, system damage, or loss of data.
	Warning	Risk of severe personal injury.
	New	This command or section is new for this release.

Table 2: Text Conventions

Convention	Description
`Screen displays`	This typeface indicates command syntax, or represents information as it appears on the screen.
The words **enter** and **type**	When you see the word "enter" in this guide, you must type something, and then press the Return or Enter key. Do not press the Return or Enter key when an instruction simply says "type."
[Key] names	Key names are written with brackets, such as **[Return]** or **[Esc]**. If you must press two or more keys simultaneously, the key names are linked with a plus sign (+). Example: Press **[Ctrl]**+**[Alt]**+**[Del]**
Words in italicized type	Italics emphasize a point or denote new terms at the place where they are defined in the text. Italics are also used when referring to publication titles.

Providing Feedback to Us

We are always striving to improve our documentation and help you work better, so we want to hear from you! We welcome all feedback but especially want to know about:

- Content errors or confusing or conflicting information.
- Ideas for improvements to our documentation so you can find the information you need faster.
- Broken links or usability issues.

If you would like to provide feedback to theExtreme Networks Information Development team about this document, please contact us using our short online feedback form. You can also email us directly at internalinfodev@extremenetworks.com.

Getting Help

If you require assistance, contact Extreme Networks Global Technical Assistance Center using one of the following methods:

Web	www.extremenetworks.com/support
Phone	1-800-872-8440 (toll-free in U.S. and Canada) or 1-603-952-5000 For the Extreme Networks support phone number in your country: www.extremenetworks.com/support/contact
Email	support@extremenetworks.com To expedite your message, enter the product name or model number in the subject line.

Before contacting Extreme Networks for technical support, have the following information ready:

- Your Extreme Networks service contract number
- A description of the failure
- A description of any action(s) already taken to resolve the problem (for example, changing mode switches or rebooting the unit)
- The serial and revision numbers of all involved Extreme Networks products in the network
- A description of your network environment (such as layout, cable type, other relevant environmental information)
- Network load and frame size at the time of trouble (if known)
- The device history (for example, if you have returned the device before, or if this is a recurring problem)
- Any previous Return Material Authorization (RMA) numbers

Related Publications

ExtremeWireless and ExtremeWireless AP documentation can be found on Extreme Documentation page at: http://documentation.extremenetworks.com

Extreme recommends the following guides for users of ExtremeWireless products:

- *ExtremeWireless AP3935 Installation Guide*
- *ExtremeWireless AP3965 Installation Guide*
- *ExtremeWireless Appliance C5210 Quick Reference*
- *ExtremeWireless Appliance C5110 Quick Reference*
- *ExtremeWireless Appliance C4110 Quick Reference*
- *ExtremeWireless Appliance C25 Quick Reference*
- *ExtremeWireless Appliance C35 Quick Reference*
- *ExtremeWireless CLI Reference Guide*
- *ExtremeWireless End User License Agreements*
- *ExtremeWireless External Antenna Site Preparation and Installation Guide*

- *ExtremeWireless Getting Started Guide*
- *ExtremeWireless Integration Guide*
- *ExtremeWireless Maintenance Guide*
- *ExtremeWireless Open Source Declaration*
- *ExtremeWireless User Guide*

1 Overview of Facilities and Usage

Session Control
Captive Portal

This chapter describes the main integration features offered by the ExtremeWireless product line and how they can be used. This document is based on release 10.01 of the ExtremeWireless software.

Session Control

The most important attributes of a user's wireless session are:

- Whether the session is active or has terminated.
- Whether the user has authenticated successfully.
- Which access control role is assigned to the user.

Session control is the process of manipulating these attributes. These are the main times at which session control is exerted:

- When a user's device first associates to an AP.
- When a user attempts to authenticate.
- Asynchronously (unsolicited) during the user's session.

When a user's device first associates to an AP, the user's identity is unknown. Only a generic access control role can be assigned to the user until the user is identified. The role assigned to a newly associated station is determined by the configuration of the Virtual Network Service (VNS) the user is accessing. Typically the role assigned to the station is the VNS' "Default Non-Authenticated Role." The only way to change which role is assigned by default to a newly associated station is to change the static VNS configuration on the controller.

ExtremeWireless lets external servers exert session control during station authentication and asynchronously (unsolicited) during the user's session. Session control at authentication time can be thought of as solicited or synchronous in that the controller (or its ECP) will ask the authorization server what to do with the current session. Asynchronous (unsolicited) session control allows an authorization server to interrupt the controller to change a session's authentication state and role. The interruption can happen virtually any time during a user's session, at the convenience of the authorization server.

The wireless controller exposes two interfaces for exerting session control:

- RADIUS Protocol
- Session Control Web Interface

The RADIUS protocol is an established standard for session control. Most of the RADIUS standards deal with session control that is exerted at the moment the station authenticates. RFC 5176 specifies RADIUS extensions that allow an external authorization server to:

[1] Numerous IETF Request For Comment (RFC) documents deal with aspects of the RADIUS protocol and function. The most important ones for ExtremeWireless are RFCs 2865, 2866, 2869, 3579, 3580 and 5176.

- Terminate a station's session at any time using "Disconnect Messages" (DM).
- Change a station's access control role at any time using "Change-of-Authorization" (CoA) messages.

The ExtremeWireless controller supports both of DMand CoA extensions. Their use is discussed in Facility Reference—RADIUS Authentication and Authorization on page 105 The controller also supports RADIUS Accounting, which is discussed in Facility Reference—RADIUS Accounting on page 134.

Because many developers are unfamiliar with the use of the RADIUS protocol, the wireless controller exposes an HTTP/HTTPS web interface with similar functionality. The web interface includes request messages that exert session control at the moment the station authenticates. These messages are intended for use by External Captive Portals. The messages and their functions are described in Facility Reference—External Captive Portal: External Authorization on page 22 and Facility Reference—External Captive Portal: Internal Authorization on page 40.

Like the RADIUS protocol the web service includes messages that can be sent unsolicited at any time to terminate a session or changing its access control role. These messages are described in Facility Reference—Unsolicited Session Control Web Interface on page 49.

This web interface is available only if at least one external captive portal has been configured on the controller. As explained later in this document, the web interface for a single external captive portal can be used to manage any user sessions on the controller.

Captive Portal

A Captive Portal (CP) provides a simple way for users to authenticate to a network. Captive Portals work by intercepting a station's HTTP requests and redirecting them to a web server's login web page. Users do not need to remember or even bookmark the login page location because they will be redirected to it whenever they need to authenticate. Once the user has been authenticated, they typically aren't redirected to the CP again until the next time the user logs in to the network.

Figure 1: Simplified Captive Portal Authentication Sequence on page 10 contains a simplified example of the steps in a typical captive portal authentication. The user has a wireless connection to the network. The user's traffic is contained to a VLAN that is bridged at the ExtremeWireless controller. All the traffic to and from the user flows through the controller, so the controller is in a position to intercept and redirect it.

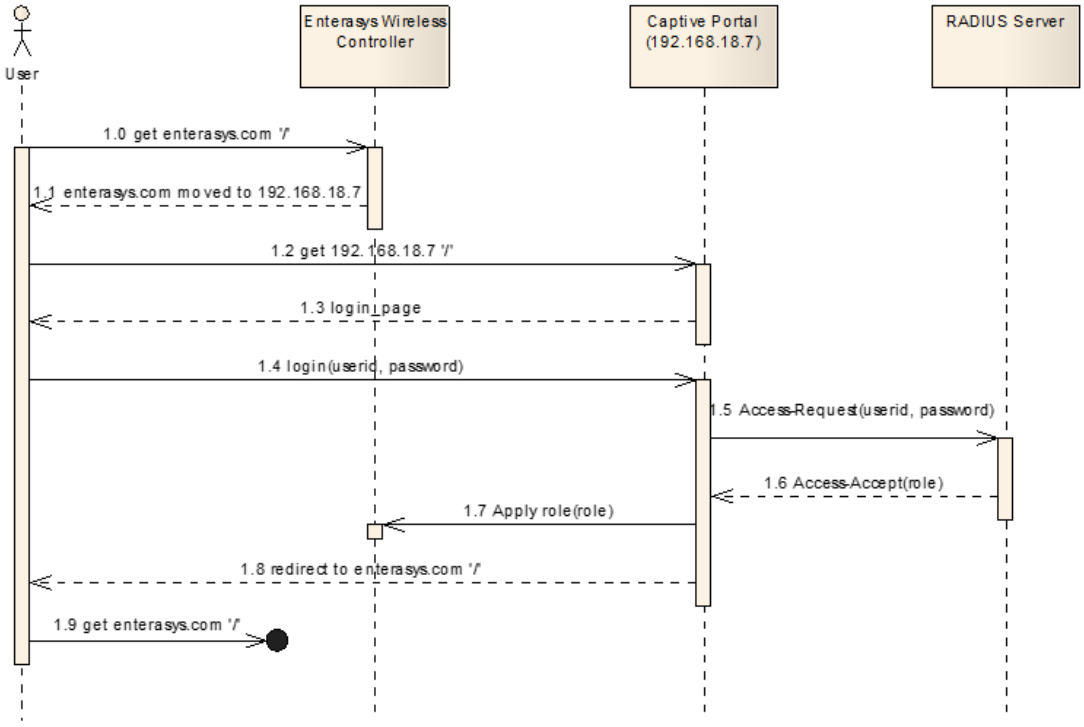


Figure 1: Simplified Captive Portal Authentication Sequence

In this example the user points the web browser to www.extremenetworks.com (the company home page). The controller intercepts the request and notices that the user has not yet authenticated. The controller impersonates the extremenetworks.com web server and tells the user's browser that the extremenetworks.com home page temporarily moved to 192.168.18.7. However, 192.168.18.7 is in fact the captive portal web server.

The user's browser follows the redirection and issues a get request for the captive portal home page (step 1.2 above). The captive portal server sends the station a login page as if the login page was the requested extremenetworks.com page.

The user fills in a form on the login page and submits it to the captive portal web server, which extracts the credentials and forwards them to an authentication server (a RADIUS server in this example).

The RADIUS server verifies the entered credentials, potentially using other authentication servers in the process. After deeming that the credentials are valid, the RADIUS server responds with an Access-Accept message containing the name of an access control role that it wants applied to the user (step 1.6).

The captive portal takes two actions in response to the Access-Accept message:

- It requests the controller apply the selected access control policy to the user (step 1.7).
- It either sends a new page to the user's browser or redirects the browser to another website (step 1.8). In this example, the browser is redirected to the extremenetworks.com home page that the user originally requested.

After a successful authentication, the user's HTTP requests are no longer redirected to the captive portal login page.

Captive Portal Variants

Captive portals are configured as part of an Extreme Networks WLAN Service. Captive portal is just one of the authentication mechanisms supported by ExtremeWireless WLAN Services. Captive portal authentication can be used on its own or combined with other supported authentication types like MAC-based authorization and 802.1x authentication.

ExtremeWireless supports the following main varieties of captive portal, each with its own set of configurable options:

- Internal Captive Portal
- Guest Portal
- Guest Splash Screen Portal
- External Captive Portal

Each WLAN Service defined on the controller can have its own captive portal. Different WLAN Services can use different types of captive portal, although a single WLAN Service can only use one captive portal.

The following sections provide an overview of each type of captive portal.

Internal Captive Portal

An Extreme Networks internal captive portal has the following key characteristics:

- The wireless controller that intercepts and redirects the wireless user's traffic hosts the web server that serves the login page.
- All wireless users must have a valid user ID and password to access the network.
- The captive portal web server authenticates user credentials against a back-end RADIUS server. The RADIUS server is separate from the wireless controller. The RADIUS server can use its own database or that of another authentication server to validate the credentials and select the access control policy assigned to the user's session.

All software needed to implement an internal captive portal is installed on the wireless controller. The captive portal can be configured on the Authentication and Accounting tab of the WLAN Service page. The captive portal login page and auxiliary web pages can be designed directly on the controller using its graphical CP page editor.

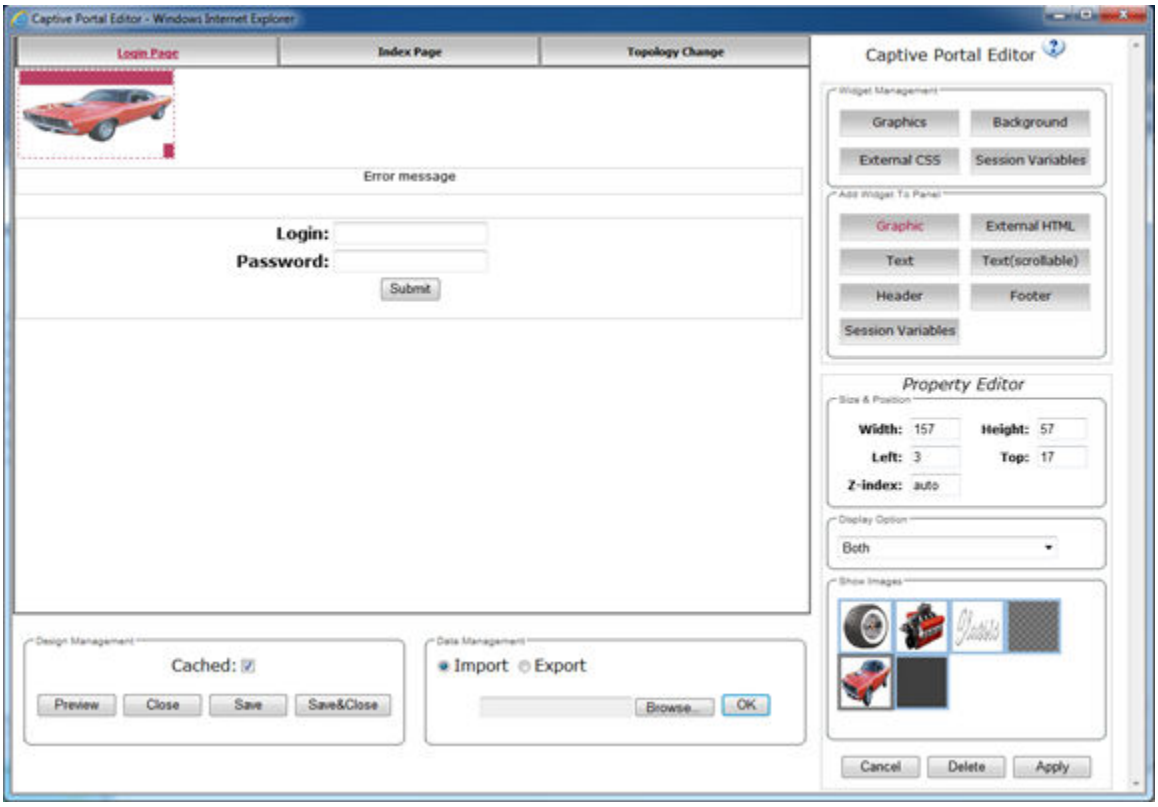


Figure 2: Graphical Captive Portal Page Editor

Guest Portal

An ExtremeWireless Guest Portal has the following key characteristics:

- The wireless controller that intercepts and redirects the wireless user's traffic hosts the web server that serves the login page.
- All wireless users must have a valid user ID and password to access the network.
- The captive portal web server authenticates user credentials against a database that is maintained on the controller itself.

All software needed to implement an internal captive portal is installed on the wireless controller. The captive portal can be configured on the Authentication and Accounting tab of the WLAN Service page. The actual page layout of the login page and auxiliary pages can be created using the graphical CP page editor built into the controller.

The controller provides a special graphical user interface (GUI) for administering the database of authorized user credentials. The GUI allows an administrator to create, edit and delete any account. The GUI can set account attributes such as the user ID and password, the total amount of network time granted to the account and the days and times during which the account can be used.

The GuestPortal Guest Administration GUI is intended for use by guest account administrators, who can be non-IT, non-technical personnel. A fully privileged administrator must create a guest account administration account on the controller. A guest administrator must login to the controller to use the guest account administration GUI and cannot access any other part of the controller GUI.

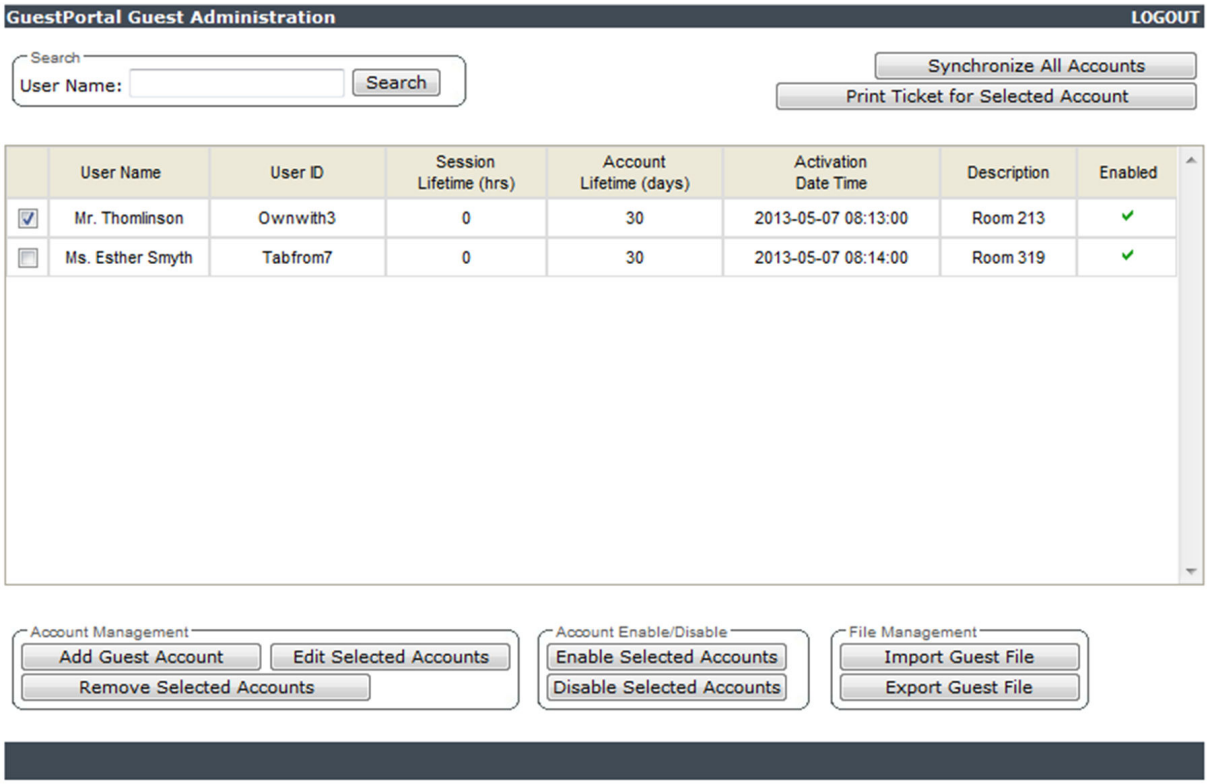


Figure 3: Guest Account Administration GUI

Each controller can have at most one configured and active guest portal. This makes it simple for account administrators because they do not need to know which guest portal each account should be assigned to.

Guest Splash-Screen

Guest Splash-Screen "authentication" refers to an authentication process in which the authenticating user does not provide credentials. Instead, any user who performs a specific action using the Splash-Screen page is treated as authenticated. Typically a user of a guest splash-screen portal must read and agree to abide by terms and conditions described on the guest splash-screen "login" page. Splash-screen "authentication" is often used by vendors that offer free wireless to a large but transient clientele for which the overhead of managing user credentials is unnecessary and unacceptable. Splash-screen "authentication" also can be used for promotional purposes.

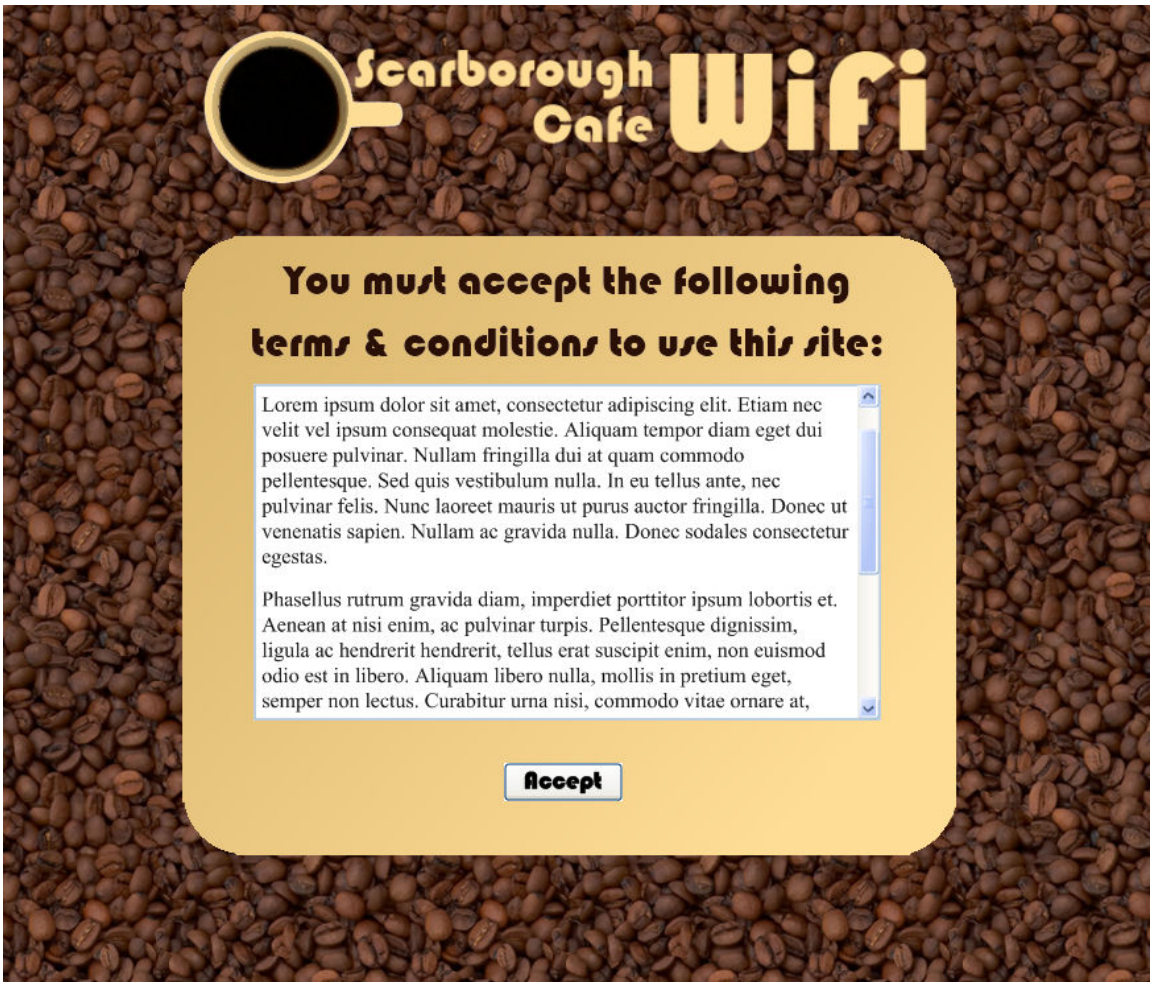


Figure 4: Example Splash-Screen Login Page

A Guest Splash-Screen captive portal has the following key characteristics:

- The wireless controller that intercepts and redirects the wireless user's traffic hosts the web server that serves the guest splash-screen page.
- Users accessing a network through guest splash screen authentication do not require credentials.
- Although guest splash-screen users do not have network credentials, they must use the guest splash screen page to "authenticate" to the network. Prior to using the guest splash-screen page the user is considered unauthenticated. After using controls on the splash-screen page they are considered authenticated. Typically, a guest splash-screen user is assigned different access control roles depending on whether he has "authenticated".

All software needed to implement a guest splash-screen portal is installed on the wireless controller. The guest splash-screen portal can be configured on the Authentication and Accounting tab of a WLAN Service configuration page. The actual page layout of the login page and auxiliary pages can be created using the graphical CP page editor built into the controller.

External Captive Portal

The defining attribute of an External Captive Portal is that the web server that hosts the login page is not running on the wireless controller; it is actually running on another web server or on a network access control device such as Extreme Networks NAC.

Because the web server is not on the controller, the external captive portal implementer has complete freedom to determine the portal server's capacity, how a station is to be authenticated, how exactly the web site should appear, and the languages and tools it is built with. Although this flexibility is useful, someone has to develop and maintain the external captive portal site.

Because the external captive portal does not sit on the controller, it must use the session control web interface to manage user sessions hosted on the controller. Several chapters of this guide will explain how to do that.

The ExtremeWireless controller supports two main variants of External Captive Portal

- External Captive Portal with External Authentication
- External Captive Portal with Internal Authentication

External Captive Portal with External Authentication places the burden of authenticating users squarely on the external captive portal. It can use whatever method it wants to authenticate the station including splash-screen authentication, RADIUS authentication, or authentication against an Active Directory server or a proprietary database. The external captive portal simply tells the controller when a station has authenticated and can specify the policy to assign to the station if desired.

External Captive Portal with Internal Authentication relies on the controller to authenticate the user. The external captive portal collects the user's credentials and forwards them to the controller. The controller in turn forwards them to a RADIUS server that performs the actual authentication. The controller receives and applies the RADIUS server response and then forwards the authentication response details back to the external captive portal. The external captive portal is then responsible for sending the user to an appropriate web page.

At first External Captive Portal with Internal Authentication might seem convoluted. However, it has the advantage of separating web interactions from RADIUS interactions. Web developers are unlikely to be familiar with RADIUS protocols and client libraries. Firewall Friendly External Captive Portal with Internal Authentication lets the controller mediate between the world of the web and the world of RADIUS.

Firewall Friendly External Captive Portal (FFECP)

Administrators increasingly are using third-party cloud providers for authentication and authorization services. These providers typically offer value-add services, such as customized web portals and station analytics.

These services often exist on the unsecured side of a firewall, while the stations and devices being authenticated exist on the protected side. This makes it difficult for the third party provider to use either unsolicited RADIUS messages or the controller's unsolicited session management web interface to authorize authenticated stations. These protocols require the third party provider to initiate a message exchange, which is typically blocked by their customer's firewall. Customers are not keen to open more ports in their firewall.

The wireless controller implements Firewall Friendly External Captive Portal (FFECP) to address this audience. When configured for FFECP, all session management communication is initiated from the

controller itself. All session management instructions from the third-party to the controller are piggybacked on HTTP responses to the station being authenticated. Because the station is behind the firewall, they are typically allowed to send through it to servers on the unsecured side, and firewall / NAT servers are able to tell which session to forward third-party authentication responses to.

FFECP has other advantages as well: the FFECP developer does not need to code a state machine that interacts with the controller over a “back door” channel in order to authorize the station. All external captive portal actions are triggered by requests from a client being authenticated and the authentication is just a response to the station. This flow is much more natural for traditional web developers to implement.

FFECP has the ability to transmit data about the current station’s session as part of the authentication request. The other forms of external captive portal require the external web server to explicitly request this information if it needs it.

FFECP handles stale session identifiers better than other solutions. All FFECP communication is timestamped. If a user uses an old URL to get to the external login page (for example, by using a browser bookmark) the FFECP and controller immediately can detect that the session identifier is stale and take appropriate action. The timestamps also protect against replay attacks on the controller and on the authentication service web servers.

For enhanced security, FFECP requests can be signed with a shared secret to mutually identify the authentication web server and the controller that redirected a user to it. The signature also provides protection against message tampering.

For these reasons customers looking at their first External Captive Portal solution should strongly consider using FFECP in place of the legacy ECP mechanisms.

Selecting the Type of Captive Portal to Use

Many factors can influence the selection of the type of captive portal to implement. The following flowchart shows a recommended decision tree for selecting the type of captive portal to implement.

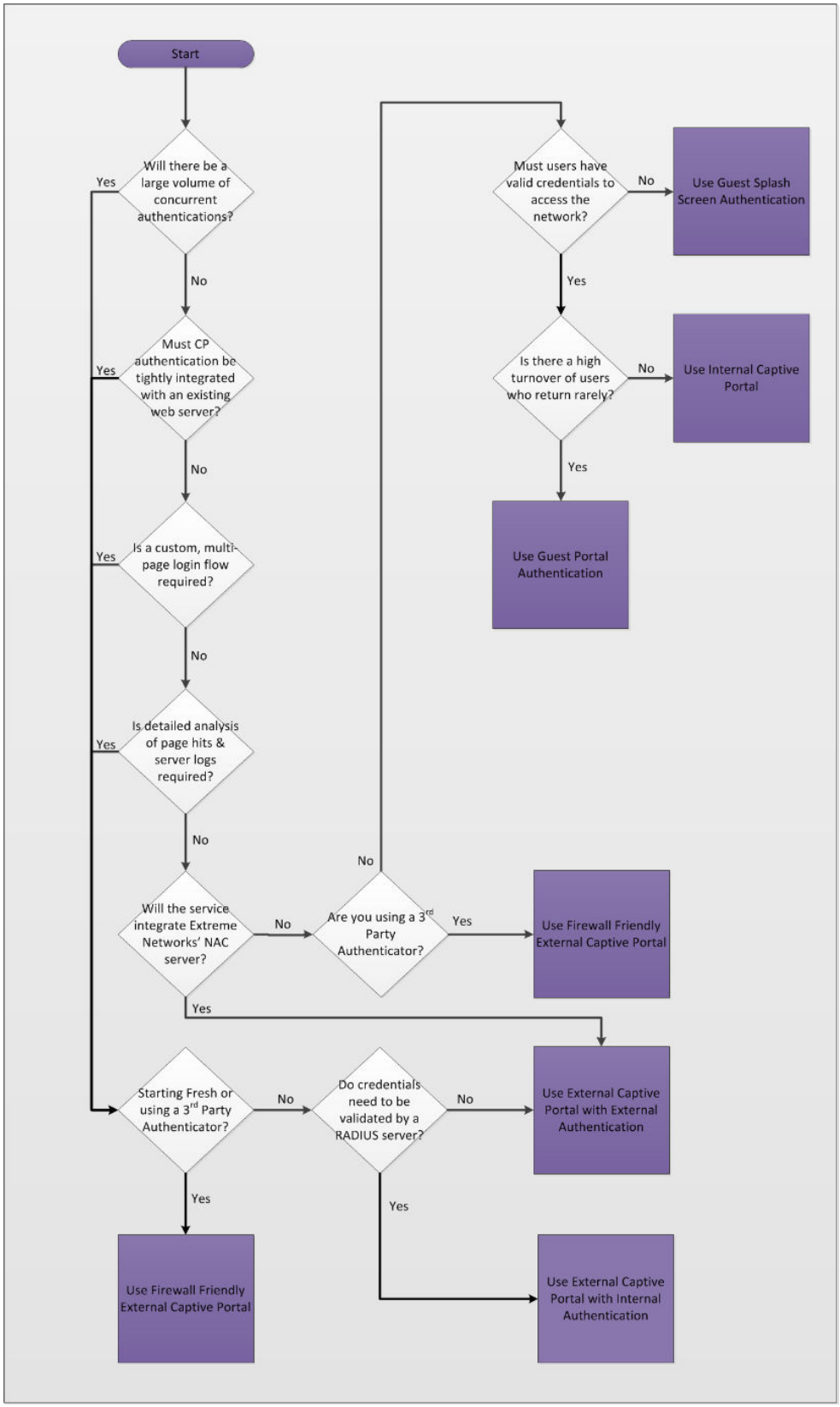


Figure 5: Flowchart for Selecting the Type of Captive Portal to Implement

The first major decision is whether to implement an external captive portal or an internal one. The main questions that need to be answered are:

1 Will the web server have a high volume of concurrent authentication attempts?

   The controller operates as a multi-function server, and thus has a limit on the resources it can dedicate to captive portal authentication. If there is a very high volume of users authenticating more or less concurrently, a dedicated web server will provide a better user experience.

2 Must CP authentication be tightly integrated with an existing web server?

   It is quite possible to make an internal captive portal have the same look and feel as an external web server. However, if the network login needs to be integrated into web pages on an existing web server, then it will be easier to implement an external captive portal. For example, if network authentication is just one of the options available on the web server home page, then the external captive portal approach is preferable.

3 Is a custom, multi-page flow required for authentication?

   All forms of internal captive portal provide for a single login page (per captive portal). While it is possible to upload custom content to an internal captive portal site so that a multi-page login sequence is possible, it will be easier to implement a custom multi-page login flow on an external web server. However, putting all the login-related data and forms on a single web page is common and may be preferred by the users.

4 Is detailed analysis of page hits & server logs required?

   The controller does not maintain page hit counters and does not expose web page access logs. For security and stability reasons, users cannot install non-Extreme Networks software on a controller. Consequently, it is difficult to collect page hit counts from an internal captive portal. External web servers are typically more open, facilitating the collection and analysis of page hits and other web usage metrics.

5 Will the wireless service be integrated with Extreme Networks Network Access Controller (NAC)?

   The Extreme Networks NAC provides many of its services, such as registration and authentication, through an integrated web server. To take advantage of these features, the wireless controller should be configured to treat the NAC like an external captive portal server.

6 Are you using a third-party authenticator? A third-party authenticator is a service provider who maintains authentication web sites on the Internet. Customers direct their users to this site to authenticate. Typically the authenticator provides other services such as a website or user analytics.

   Selecting a third-party authenticator forces the External Captive Portal choice and leans the decision strongly towards the Firewall Friendly External Captive Portal solution.

7 Is the Enterprise's organization more suited to an External Captive Portal implementation?

   While this is not strictly a technical question, it has bearing on whether the captive portal implementation will be seen as a success.

   Answering this question requires considering things like the skill sets available to implement the captive portal and how they are distributed within the enterprise. For example, it may be the case that an enterprise's marketing department is responsible for creating or maintaining the enterprise's web presence, while the IT department maintains the servers and manages the various types of user accounts.

In this type of scenario, employees within the marketing department are unlikely to be comfortable implementing on a controller and the security policy may prohibit granting non-IT personnel administrative access to network equipment. In this scenario, implementing an External Captive Portal is preferable, as it allows each department to work in their preferred environments using the tools with which they are most familiar. On the other hand, if a small IT department is responsible for the end-to-end implementation it will likely be cheaper and easier to create an internal captive portal using the Captive Portal Editor on the controller.

If the answer to any of the above questions is "yes," then it is appropriate to invest in an External Captive Portal solution. If an ECP solution is chosen, it may be necessary to decide whether the controller will be used as an interface between the ECP web server and the RADIUS server. If a NAC or third-party authentication service will be used, the controller should not be used this way; instead, an ECP with external authentication should be implemented.

If the solution does not involve a NAC or third-party service, the questions to consider are:

- Does a RADIUS server already exist in the network and is its use required? And, if so:
- How comfortable is the implementer with using a RADIUS client library and the RADIUS protocol?

If the solution must include a RADIUS server and the team developing the eECP is unfamiliar with RADIUS, then implement an external captive portal with internal authentication.

If an internal captive portal is indicated, then it is fairly easy to decide the type of portal to implement. If there will be a large number of casual, occasional users all of whom can be covered by a generic access control role, then it is not worth the effort to maintain an authentication database of credentials. A captive portal splash-screen implementation will suffice.

If different users need to receive different access rights, then the users must be identifiable. This means a guest splash-screen portal cannot be used. If an enterprise authentication infrastructure already exists and the wireless users are part of that infrastructure, then it makes sense to implement the standard internal captive portal. If the wireless service primarily serves transient visitors, who may need to be given accounts on short notice, then it is worth considering implementing a guest portal.

Web Jail or Walled Garden?

The phrases "Web Jail" and "Walled Garden" are frequently used in conjunction with "Captive Portal." Web jails and walled gardens are not specific types of captive portals. In fact web jails and walled gardens can be created with any of the types of captive portals discussed previously. What is special about a web jail or walled garden is the access control policy assigned to each station before, and sometimes after, an authentication.

A simple approach to captive portal authentication is to allow unauthenticated users just enough network access to be able to reach the captive portal. In this case an unauthenticated user might be assigned to a role that only permits the use of DHCP, DNS, ARP, and HTTP that is directed to the captive portal web server. All other traffic is dropped at the Access Point (AP) or controller. It is easy to see how a user in this scenario could feel he or she was trapped in a web jail. If the user authenticates successfully he or she is assigned to a less restrictive access control role.

It is not necessary to be this restrictive. It is quite possible to allow an unauthenticated user to have some access to the Internet while prohibiting access to the enterprise servers. This may even be necessary if the user is expected to download the latest patches and virus definitions before being granted full network access. This scenario is sometimes referred to as a "Walled Garden" because the

user is given some network access without authenticating, but is blocked from many other network locations and services until he or she authenticates.

SNMP

The ExtremeWireless Controller hosts an SNMP agent that supports either SNMPv2c or SNMPv3, depending on how it is configured. The SNMP agent is intended primarily to provide read access to the configuration and statistics managed by the controller. The standard get, get-next and get-bulk operations are supported.

The controller also supports sets on a limited number of OIDs. Set support has been implemented primarily to integrate with Extreme Networks NAC and Extreme Networks Policy Manager. Most configuration changes should be made through the controller's GUI or CLI or using NetSight Wireless Manager.

The currently supported MIBs are listed in the release notes that accompany each release of ExtremeWireless software. The release notes are authoritative in this respect. The list shown below is provided as a convenience and reflects the MIBs that are supported as of release 9.01.01.

Supported IETF & IEEE Standards MIBs

RFC No.	Title	Groups Supported
Draft version of 802.11	IEEE802dot11-MIB	Standard MIB for Wireless Devices
1213	RFC1213-MIB	Most of the objects supported
1573	IF-MIB	ifTable and interface scalar supported
1907	SNMPv2-MIB	System scalars supported
1493	BRIDGE-MIB	HWC supports relevant subset of the MIB
2674	P-BRIDGE-MIB	HWC supports relevant subset of the MIB
2674	Q-BRIDGE-MIB	HWC supports relevant subset of the MIB
	IEEE-8023-LAG-MIB	LAG configuration information. Set is permitted for LAG L2 port configuration only.

Supported Extreme Networks Private Enterprise MIBs

Extreme Networks Private Enterprise MIBs are available in ASN.1 format from the Extreme Networks website at: www.extremenetworks.com/support/mibs/. Indexed MIB documentation is also available.

Title	Description
ENTERASYS-CONFIGURATION-MANAGEMENT-MIB	Used to perform configuration backup and restore
ENTERASYS-CLASS-OF-SERVICE-MIB	Used for configuration/monitoring CoS and rate control
ENTERASYS-POLICY-PROFILE-MIB	Used for configuration/monitoring policy and rules assignments
ENTERASYS-RADIUS-AUTH-CLIENT-MIB	Used for configuration of RADIUS Authentication servers
ENTERASYS-RADIUS-ACCT-CLIENT-EXT-MIB	Used for configuration of RADIUS Accounting servers

Title	Description
ENTERASYS-IEEE8023-LAG-MIB-EXT-MIB	Used for configuration/monitoring LAG port
CTRON-ALIAS-MIB	Used by Extreme Networks NAC to determine the MAC address that goes with a particular IP address.
ENTERASYS-RF-LOCATION-MIB	Used to configure aspects of the ExtremeWireless device location service.

Supported Siemens HiPath Wireless MIBs

Title	Description
HIPATH-WIRELESS-HWC-MIB	Configuration and statistics related to EWC and associated objects
HIPATH-WIRELESS-PRODUCTS-MIB	Defines product classes
HIPATH-WIRELESS-DOT11-EXTNS-MIB	Extension to IEEE802dot11-MIB that complements standard MIB
HIPATH-WIRELESS-SMI	Root for Chantry/Siemens MIB

2 Facility Reference—External Captive Portal: External Authorization

Overview

As discussed in Overview of Facilities and Usage on page 8, an External Captive Portal (ECP) is a web server that hosts a site that allows users to authenticate to the network. The web server is not hosted on the wireless controller, but instead intercepts some of the user's HTTP messages and redirects them to the ECP web server.

This chapter describes the implementation of External Captive Portal with External Authorization (ECPEA), which is a variant of ECP where the web server is responsible for contacting any back-end authentication server in order to verify a user's credentials and to determine the access control role to assign to a station.

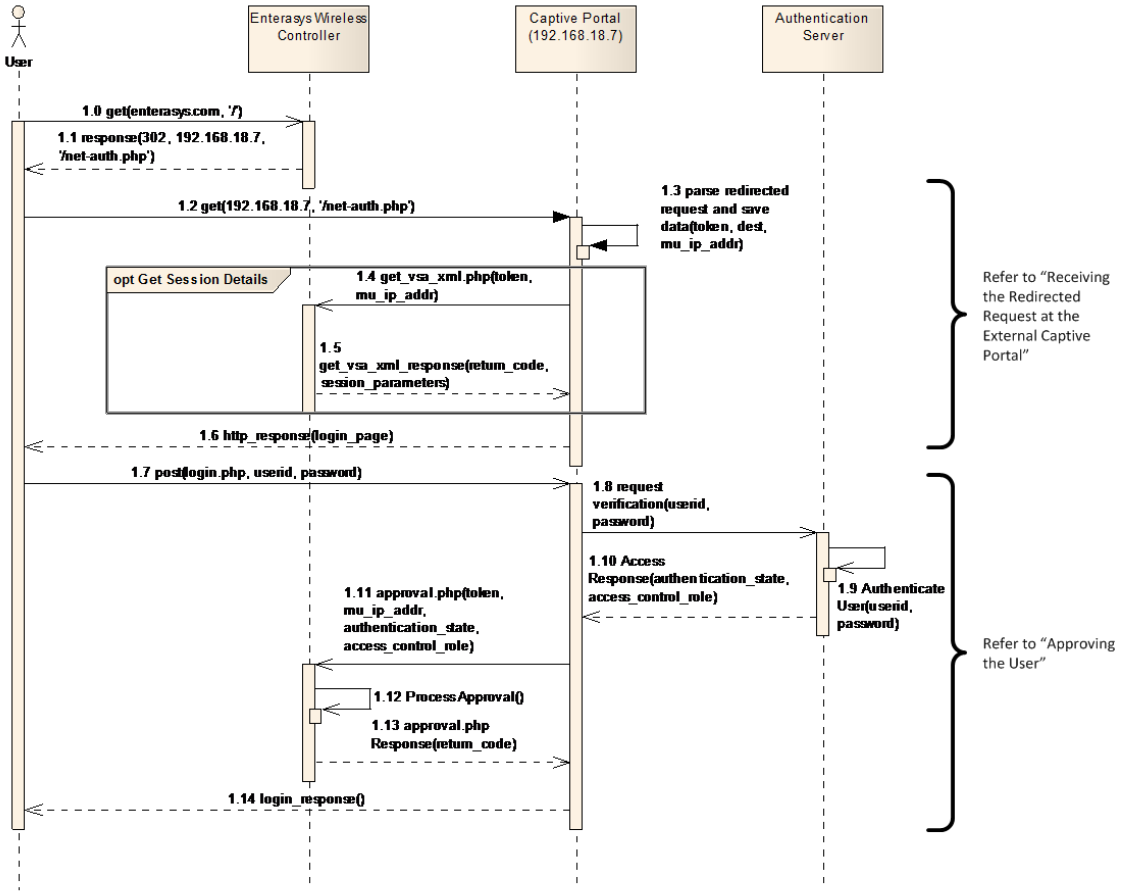


Figure 6: Flow of External Captive Portal with External Authorization

Figure 6: Flow of External Captive Portal with External Authorization on page 23 illustrates the primary event flow of a user authenticating with an external captive authentication using external authorization. The illustration expands on Captive Portal on page 9.

The sequence begins with the user making an HTTP request. The controller intercepts the request and redirects it to the ECP web server. The controller does not do this automatically; the administrator must configure the controller to perform the redirection. The steps required to configure the controller to redirect traffic are covered in Configuring the Controller to Redirect to an External Captive Portal on page 25.

As shown at step 1.1, the redirection takes the form of the controller sending an HTTP response containing the 302 response code. This code indicates that the requested URL has been relocated to the external captive portal server temporarily. Most modern browsers automatically follow the redirection to the server indicated in the response.

The ECP web server receives the redirected HTTP request and serves the login page. The web server invokes a script or program to prepare and serve the login page. Although the script can be very simple, it must capture a few identifiers for the user's device. These identifiers are sent to the controller in HTTP session control messages for the user's session. More details are provided in section "(missing section link in original content)".

Optionally, the script can ask the controller for details about the station's session, which may then be displayed on the login page. As Figure 6: Flow of External Captive Portal with External Authorization on page 23 illustrates, details about a user's wireless session can be obtained by sending a request for "get_vsa_xml.php" (see step 1.4). This message and the response to it are covered in more detail in Composing the Login Page on page 33.

The script that receives the redirected HTTP request and serves the login page is provided by the customer. The script can be written in any programming language, including Java, PHP, and C#. The script can have any valid file name as long as the controller is configured to redirect to it. The example provided here assumes that the script is called "net-auth.php" and is written in PHP. A sample of the net-auth.php script is contained in PHP External Captive Portal: External Authentication on page 167.

Typically the "net-auth" script emits a web page containing a form that allows the user to submit his or her user ID and password. The form is submitted to the ECP, not to the controller. It is up to the ECP to authenticate the credentials and decide which access control role to apply to the user. The ECP can use any mechanism it likes to authenticate the user, including "splash-screen" style "authentication." If an authentication server is used, it could be a standards-based RADIUS server, a proprietary authentication database server, an Active Directory service, or something else. The main requirement on the authentication server is to be able to determine whether submitted credentials are valid and, if so, which role to assign to the user.

Regardless of how the web server authenticates the user, it must tell the controller whether the user is authenticated and, if so, which role to apply to the user's session. As Figure 6: Flow of External Captive Portal with External Authorization on page 23 illustrates, a script/program on the web server does this by sending an HTTP request for "approval.php" to a specific interface on the controller. This request and the controller's response to it are covered in Approving the User on page 36.

The last step the external web server performs is sending feedback to the user based on the outcome of the authentication attempt (See step 1.13 in Figure 6: Flow of External Captive Portal with External Authorization on page 23.) The feedback can take the form of a simple success or error message, a redirection to the page the user was originally trying to get to, or a redirection to another page.

As Figure 6: Flow of External Captive Portal with External Authorization on page 23 illustrates, most of a user's interactions during external captive portal authentication are with the external captive portal web server. Even though the user's traffic likely is flowing through the controller, the controller simply forwards it on. The controller directly interacts with the station only to perform the initial redirection. The external captive portal interacts with the user and then uses a separate session control web API to control the user's wireless session through the controller.

A single controller can use many different external captive portals at once. The only restriction is that each WLAN Service can use at most one external captive portal for authentication.

[2] For example, an enterprise might want to allow users to by a block of network time upfront to use whenever they choose. In this case, the enterprise might have a proprietary application and database set up to accept purchases of time blocks and to track their usage. In this example, the external captive portal would validate credentials against the proprietary application database to make sure the user had credentials and time left on his or her account. The interface to the application could be completely proprietary.

Configuring the Controller to Redirect to an External Captive Portal

The first step of configuring the controller is to redirect to the ECP. HTTP traffic is redirected when:

- It is blocked by a rule or default action in an access control role, and
- The user that sent the traffic is in an unauthenticated state, and
- The traffic is carried on a tunneled (Bridged at Controller, Routed) topology, and
- Some type of captive portal authentication has been configured for the user's WLAN Service, including a destination for redirected traffic.

The controller needs to be configured to redirect user HTTP traffic and it needs to be told where to redirect that traffic as well as how to listen for instructions sent to it by the External Captive Portal web server. Policy is used to cause the controller to redirect unauthenticated HTTP traffic. WLAN Service configuration determines where the redirected traffic goes and how communication between the controller and the External Captive Portal takes place.

Causing HTTP Traffic to be Redirected using Roles

To use HTTP redirection, an access control role must be defined on the controller with the following characteristics:

- The role allows the station to use DHCP, DNS, and ARP.
- The role allows the station to communicate with the external captive portal server using HTTP or HTTPS.
- The role blocks at least some HTTP traffic, although not HTTP traffic destined for the External Captive Portal.

The role must permit the station to send and receive DHCP, DNS, and ARP traffic so that the station can function on an IPv4 network. The reasons the role must block some HTTP traffic to enable redirection are:

- It allows the controller to redirect only a specific subset of HTTP traffic. This is useful when unauthenticated users should be granted some degree of Internet HTTP access.
- It permits an efficient implementation in the controller and AP data planes.

HTTP traffic to the external captive portal must not be blocked by the role. Otherwise, every HTTP request from the station for the ECP server will trigger the controller to send a redirect back to the station. This can cause an infinite loop and obviously must be avoided.

A minimal access control role that meets these requirements is shown in Figure 7: A Minimal Access Control Role for External Captive Portal on page 26. The role's default action (applied to all traffic not matching a policy rule) is to contain to VLAN 16 (External CP). The first policy rule allows access to port 80 on 11.11.11.254, which is the external captive portal server in this example, and resides on VLAN 16's subnet. If this was not the case, then a rule allowing access to VLAN 16's gateway server might be required. The second and third rules allow DHCP and DNS traffic to and from the user.

The role does not contain an explicit rule for handling ARP messages. In this case, an ARP request or response for an address is filtered via the role exactly like an IPv4 message sent to or received from the address.

The last policy rule denies all traffic not matching a preceding rule. This will catch all HTTP traffic not sent to 11.11.11.254. When the role is applied to an unauthenticated user on a WLAN Service using captive portal authentication, the user's HTTP traffic will trigger a redirection to the captive portal page.

This role is extremely restrictive. As already indicated, a rule allowing access to the gateway is required if the captive portal does not have an interface on the same subnet as the authenticating user. Additional resources can be made available to an unauthenticated user by adding rules that allow access to them (rules with an action of "Allow" or "Contain to VLAN").

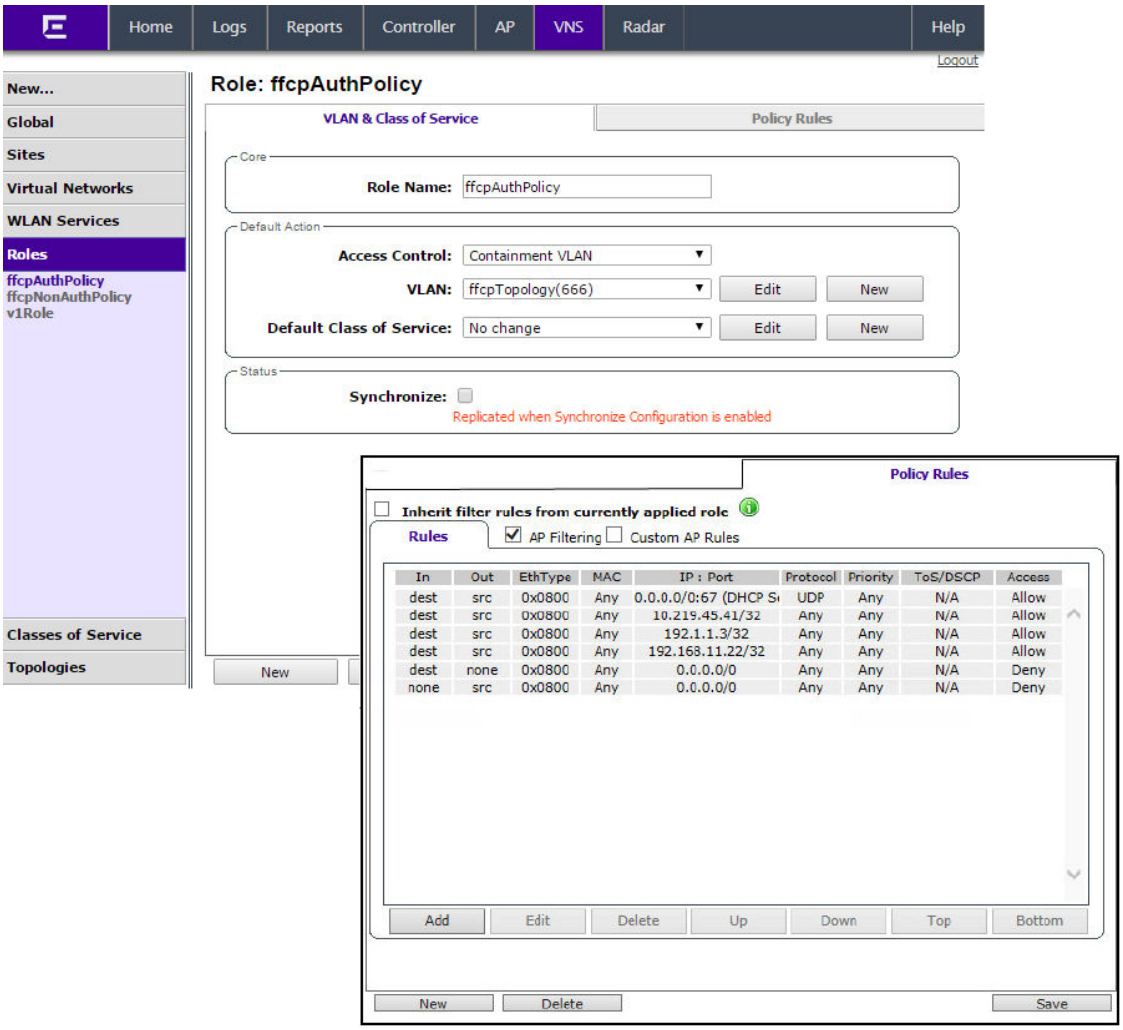


Figure 7: A Minimal Access Control Role for External Captive Portal

To have an effect, the role must be assigned to a user even before that user authenticates. The role should be assigned as the default non-authenticated role of the VNS / WLAN Service that is using external captive portal authentication. Figure 8: A VNS that Assigns the Restrictive Role to Unauthenticated Stations on page 27 shows the definition of a VNS that uses the restrictive policy

shown in Figure 7: A Minimal Access Control Role for External Captive Portal on page 26 to enable external captive portal authentication for unauthenticated users.

Note

Why a Default Action and a "Deny All IPv4" Rule?

Since 9.01 roles have a default action, why not just make the default action "Deny All"?

As of release 9.01, ExtremeWireless introduced a feature permitting an AP to redirect denied HTTP traffic to the controller under exactly the same circumstances that the controller would have redirected the traffic to a captive portal. The AP forwards the blocked traffic to the controller, which triggers the controller to redirect the user's browser to a captive portal. This allows administrators to combine filtering at the AP with captive portal authentication, a combination that was not available in previous releases.

When the AP sends a blocked HTTP request to the controller it must tell the controller on which VLAN to place the redirection. The VLAN can determine the specific IP address to which the browser is redirected. A rule that denies traffic does not specify a VLAN on which to send the traffic. So the AP assigns the blocked HTTP traffic to whichever VLAN the role's default action contains. If the default action was also deny all, the AP would not know how to mark the HTTP request being redirected to the controller.

The upshot is that any role that is intended to be used as the default non-authenticated role of a VNS should have a "Contain to VLAN" default action.

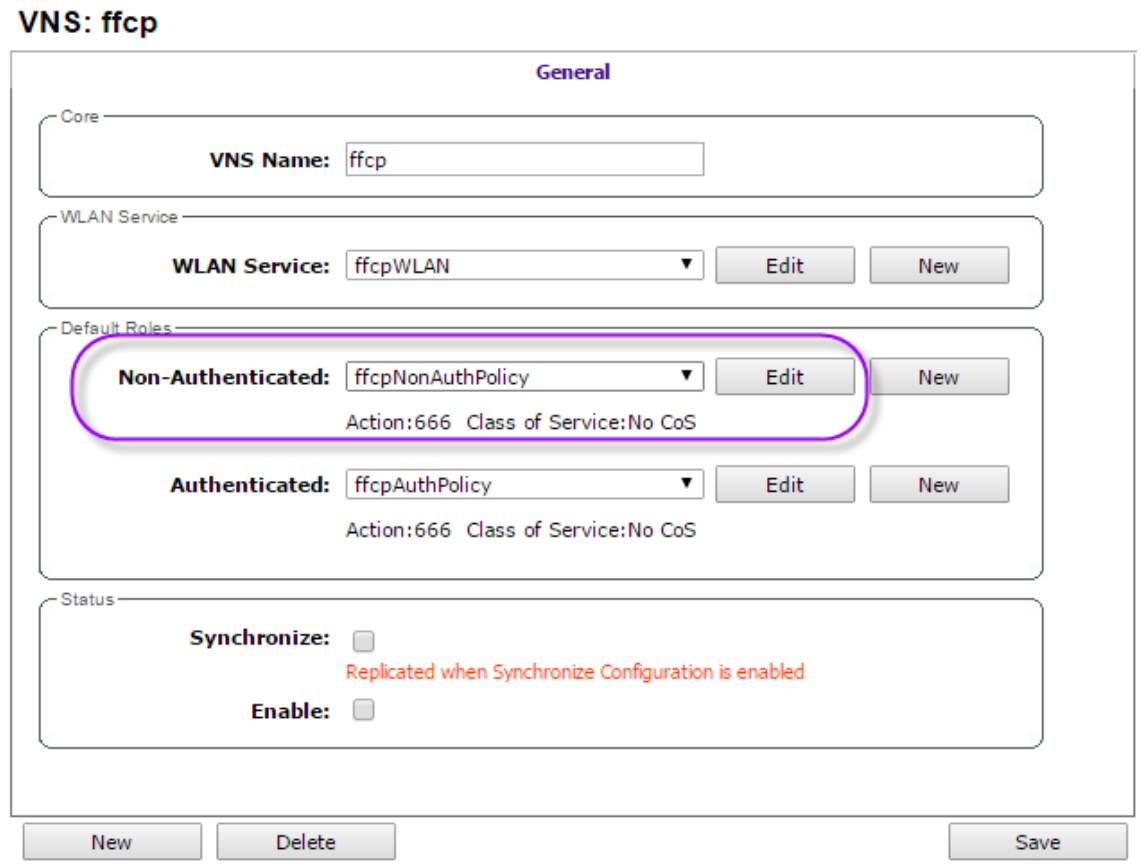


Figure 8: A VNS that Assigns the Restrictive Role to Unauthenticated Stations

General configuration and use of access control roles are beyond the scope of this document. More information on access control roles and how to configure them can be found in the *ExtremeWireless Convergence Software User Guide* and in the documentation for Extreme Networks Policy Manager.

Configuring the WLAN Service for ECPEA

The remaining required controller configuration is performed on the **Authentication and Accounting** tab of the WLAN Service that will be using ECPEA. If external captive portal authentication is not enabled already, select **External** from the **Mode** drop-down list and save the WLAN Service. This will enable the **Configure** button.

Selecting **Configure** opens a dialog box containing configurable external captive portal options. The options only affect the current WLAN Service's external captive portal.

The **HWC Connection** option contains the IPv4 address and port to which the External Captive Portal sends all HTTP session control requests. Session control messages include HTTP requests for:

- get_vsa_xml.php
- approval.php
- event.php

The selected IP address is for one of the controller's physical topologies. If the controller has more than one WLAN Service using external captive portal, there is no restriction on which of the controller's physical interface IP addresses and ports are used for each WLAN's session control interface.

The **Enable https support** option causes the controller to use TLS to encrypt HTTP traffic to and from the "HWC Connection" address. This has no impact on the traffic exchanged between users' browsers and the External Captive Portal. When enabled, this option protects the session control traffic between the external captive portal and the controller from being read by a third party. This is particularly useful when a dedicated network management VLAN is unavailable to carry the session control traffic.

The controller acts as the server end of the HTTPS connection. Communication with the ECP is secured by a certificate on the controller. The ECP does not need a client certificate to access the controller's web session control interface using HTTPS.

When this option is enabled, the scripts/programs on the ECP that send session control messages to the controller must use an SSL/TLS library for communication. External captive portal scripts written in PHP can use the file_get_contents() procedure to make HTTP or HTTPS requests. Portal scripts implemented in. Net can use System.Net.Http.HttpClient. Portals implemented in Java can use the Apache HttpComponents' HttpClient library, among others.

The library may need to be configured to trust the controller's certificate. Each library has its own method of setting up trust relationships and overriding certificate validation procedures. Please consult the chosen library's API documentation for details.

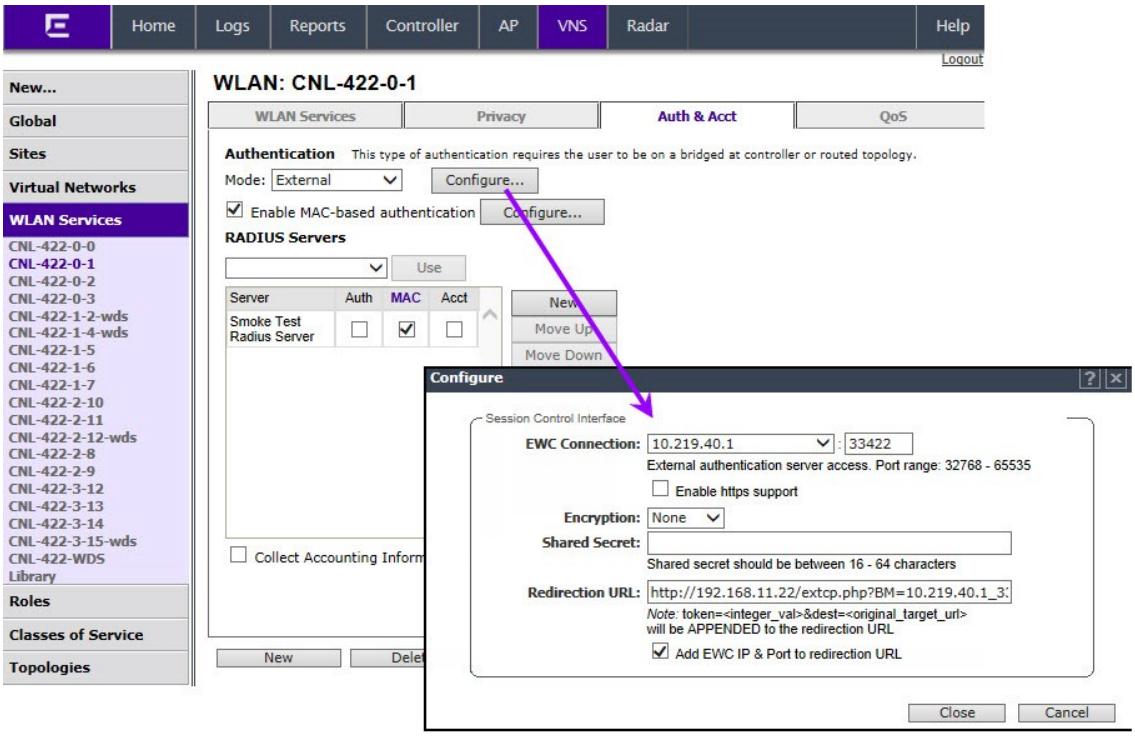


Figure 9: Configuring the Example WLAN Service for External Captive Portal (Release 9.01)

The Encryption drop-down list provides an alternative or additional privacy for session control messages. The supported options are:

- None—no encryption is performed. If the HTTPS option is not enabled, session control messages are sent in plain text over the network.
- Legacy—both the ECP and the controller are expected to use simple message encryption based on MD5. Frames are encrypted by Xoring session control message payload with a keystream generated from an MD5 hash of a shared key. This is a weak encryption algorithm and is only supported for backward compatibility. If encryption is needed, consider using the option below.
- AES—session control messages sent by the controller and ECP are encrypted with the "Advanced Encryption Standard" based on the Rijndael cipher. AES encryption is considerably more secure than legacy encryption.

Note

Using the encryption option has one advantage over using the HTTPS option alone. When HTTPS is enabled, the ECP can authenticate the controller's certificate, but the controller does not ask the client to provide one. Consequently, HTTPS does not prevent unauthorized users from sending messages to the session control interface.

Because the encryption option is based on a shared key, the encryption provides a form of authentication. If the controller can decrypt the payload of a session control message, then it is has reason to believe the message came from the external captive portal.

If encryption is enabled then a shared key must be entered. A shared key is a string that both the controller and the ECP use to encrypt and decrypt session control messages. The shared key must be

between 16 and 64 characters long. For better security, use a long key composed of randomly selected characters.

The Redirection URL field contains the URL to which the controller will redirect all blocked, unauthenticated HTTP traffic on this WLAN Service. This should be the URL of the page that will prompt the user to authenticate. The redirected browser will issue a “get” to the ECP for this URL.

The “Redirection URL”:

- Can begin with “http://” or “https://”.
- Must end with a “?” or “&”. Use “&” if the base URL contains some query strings.

The Add HWC IP & Port to redirection URL option is useful if the external captive portal serves more than one controller. An ECP must send its session control messages to the controller hosting the controlled session. If an ECP serves more than one controller, then the Add HWC IP & Port to redirection URL option must be used to identify the source of the redirection. The ECP should store the controller address and port with the token and other session details so that it is available throughout the authentication process.

Close the Configure dialog box and save any changes. The controller will now redirect all blocked HTTP traffic on the WLAN Service to the external captive portal.

Using AES Encryption

When a WLAN Service is configured to use AES encryption on the session control interface, it expects a very specific criteria to be met precisely. External Captive Portals implemented in PHP can simply copy the example crypt_AES script. However, it is important to understand the controller’s expectations if using the example script is not an option.

AES uses Rijndael encryption. Rijndael is a block cipher meaning that it operates on fixed size blocks of bits. AES uses Rijndael with a block size of 128 bits.

A clear text message requiring encryption typically is longer than 128 bits. There are a number of different ways to apply block ciphers to messages that are longer than one block. The session control web interface uses AES in Cipher Block Chaining (CBC) mode, where the cipher text output from encrypting the previous block is fed into the encryption of the next plain text block. This makes it more difficult to replace a cipher text block with an arbitrary block of text.

When encrypting the first block, there is no previous block to feed into the encryption process with the clear text message. CBC mode requires the use of an Initialization Vector (IV) to act as the cipher text input for encrypting the first clear text block. The IV is the same size as one cipher block. In the case of AES-CBC the IV needs to be 128 bits (16 bytes).

Normally the IV is generated using a random number generator. Usually a new IV is generated for each new message requiring encryption and included in clear text form with the encrypted message. The controller uses a simplification to save space in the HTTP get request. The controller always uses an IV with 16 binary zeros. Because it is constant, it does not need to be included in the get request. The controller expects that all AES encrypted messages sent to it are encrypted using a 16 byte sequence of binary zeros.

Because AES uses a 128 bit block size, the clear text to be encrypted must have a length that is a multiple of 128 bits. To ensure the clear text is the correct length, it may need to be padded before it is

passed to the encryption engine. If the message needs padding, it must take a special form; the padding consists of one byte repeated a sufficient number of times to bring the clear text to the correct length. The padding byte itself must be a byte in the range 1 to 15. The value of the byte is the number of padding characters added to the message. In other words, the value of the padding byte is computed as:

```
padding_byte = AES_Block_Size - (length(clear text) mod AES_Block_Size)
```

where

AES_Block_Size = 16 (bytes)

If padding_byte = AES_Block_Size, then no padding is needed.

Finally, it is very important to know that the controller uses the password entered on the ECP configuration dialog box of the WLAN Service configuration page as the encryption key. The controller may pad or truncate the password to the length necessary for the AES encryption key. Obviously the ECP must truncate or pad the password exactly the same way as the controller or the cipher text will not decrypt correctly.

If the clear text password is less than 32 bytes (32 ASCII characters) long, use AES-128 bit encryption (128-bit encryption key length). If the clear text password is 32 ASCII characters or longer (up to 64 ASCII characters), use AES-256 bit encryption (a 256-bit encryption key length). The controller does not allow the password to be fewer than 16 characters. So if the password is between 16 and 31 characters inclusive in length, it will be truncated to its first 16 characters. If the password is 32 characters or longer, then the first 32 characters of the password are used as the key.

Using Legacy Encryption

The legacy encryption algorithm consists of exclusive ORing, the plain text message with a 16 byte string generated using an MD5 hash algorithm. The clear text is broken into 16-byte blocks and each block is encrypted with a different MD5 generated hash string. The first 16-byte plain text block is Xored with a hash string generated by MD5 hashing the password string entered into the ECP configuration dialog box of the WLAN Service configuration GUI. Subsequent 16-byte blocks are encrypted using an MD5 hash of the password entered in the ECP dialog box concatenated to the hash string Xored with the previous 16-byte block of clear text, concatenated to the previous 16-byte block of clear text.

Legacy encryption and decryption are essentially the same algorithm. This can be seen by comparing decrypt_md5 to crypt_md5 in PHP External Captive Portal: Internal Authentication on page 183.

Receiving the Redirected Request at the External Captive Portal

The script on the ECP that receives redirected requests has two responsibilities:

- Parse the redirection URL and preserve critical parameters for future use.
- Compose the web page that the user fills in to log into the network.

Parsing the Redirection URL

The request for the login page will take the form of an HTTP/HTTPS get request. All the arguments to the request are passed as query strings appended to the URL. Typically the web server or the back-end

runtime system will have parsed the query strings and made them available in a more convenient form for the back-end scripts.

The format of the redirected URL depends on the options configured on the External Captive Portal configuration page. If encryption is not enabled on the controller then the web server will make the individual login parameters available to the script. The parameters are described in the following table.

Table 3: Parameters Available on the Redirection URL

Parameter Name	Parameter Value	Mandatory	Notes
token	Alphanumeric String	Yes	An identifier for the user's wireless session hosted on the controller that performed the redirection.
dest	URL	Yes	The URL that the user's browser was requesting when it was redirected to the ECP.
hwc_ip	IPv4 Address	No	IP address of the controller that redirected the user. This is the IP address configured in the **Connection** field on the **External Captive Portal configuration** dialog on the controller. This is the address to which all session control messages should be sent. This optional parameter is included only if the controller is explicitly configured to send it, as discussed in Configuring the WLAN Service for ECPEA on page 28.
hwc_port	TCP Port Number	No	This is the TCP port number configured in the **Connection** field on the **External Captive Portal configuration** dialog on the controller that redirected the user. This is the port to which all session control messages should be sent by the ECP receiving the redirect. This optional parameter is included only if the controller is explicitly configured to send it, as discussed in Configuring the WLAN Service for ECPEA on page 28.

The token identifies the user's wireless session on the controller that redirected the user. It is required by all session control messages except requests for event.php. Consequently the ECP needs to associate each authenticating user with the user's token for the duration of authentication. The token can be written into hidden variables on a page, stored in a cookie or stored in session state variables on the ECP. How the ECP remembers the user's token does not matter to the controller.

The originally requested URL is useful if the web server wants to redirect the user to his original destination after a successful authentication. If the web server will do this redirection then it must store the originally requested URL for later use. Otherwise it can ignore this parameter.

Whether the IP address and port of the controller that redirected the user must be tracked like the token depends on whether the ECP serves more than one controller. Obviously the ECP must know the IP address and port to send session control messages to since it must at least send a request for approval.php after a successful authentication. If the ECP serves a single controller then the controller's IP address and port can be read from a configuration file or hard-coded into the scripts. If the ECP serves more than one controller then it must associate the controller's IP address and port with the user

for the duration of the authentication. Only the controller that redirected the user can process successfully session control messages for the user.

Note

More about the Session Token

The session token is a mandatory parameter for get_vsa_xml.php, approval.php, and auth_user_xml.php requests. The token is created by the controller when it is about to redirect an HTTP request. The ECP should include the token in all session control messages sent to the controller during the authentication process (e.g., in requests for get_vsa_xml.php and approval.php).

The controller destroys its record of the token after authentication completes. Consequently, the token is not used in requests for event.php.

The session token is included in the redirection URL. Therefore, if a user bookmarks the ECP login page, the bookmark includes the token. If at some other time the bookmark is used to go directly to the ECP login page, the old parameters, including the old token, will be provided. So long as it is the same user and same computer submitting the URL, the controller will handle the stale token correctly and the user will be able to login.

If the user shares the bookmark with someone else, then the bookmark will also contain the original user's original token. This can cause problems for the person receiving the bookmark.

To work around this possibility, the controller allows the ECP to send the current user's IP address as well as his token in the get_vsa_xml.php and approval.php requests. The host web server usually provides access to the user's IP address through an environment variable or API call.

It is generally a good idea to send both the user's token and IP address. When both identifiers are provided, the controller will try to use the token and fall back to the IP address if there is a problem with the token.

Composing the Login Page

How the login page is composed depends on the programming language and tool set used. This is largely outside the scope of this document. Any programming language that can be used for web development can be used to create an External Captive Portal.

The content on the login page will depend on the overall environment the ECP serves. It may contain little more than the fields necessary to submit a user ID and password, or it could include news, notifications, and links to other relevant websites.

Sometimes it is useful to display some information about the use, such as the SSID the user is accessing on the login page. The get_vsa_xml.php request in the session control API provides a convenient way to collect the controller's information about the user prior to authentication. The request message must include the user's token and can include the IP address. The request should be sent as an HTTP/HTTPS get. The response will take the form of an XML document containing the following details:

- The token that was sent in the request.
- The name of the AP that the station is associated with currently.
- The serial number of the AP that the station is associated with currently.
- The name of the Virtual Network Service (VNS) that the station is accessing the network from.
- The SSID that the station associated to.

- The MAC address of the station.
- The name of the policy assigned to the station currently.
- The name of the topology assigned to the station currently.

The document also contains elements for ingress rate control and egress rate control (“ingress_rc” and “egress_rc”, respectively). These are deprecated attributes. They are always set to “N/A”. The various rate limits that can be applied to a station are a function of the role assigned to the station as identified by the policy element in the get_vsa_xml.php response document.

The code snippet below shows an example of the XML document returned by a get_vsa_xml.php request.

```
<?xml version="1.0"?>
<response>
  <token>T7vb1LdUZmsuY0q9V60Iww!!</token>
  <ap_name>Frasier Building 3rd Floor Room 110</ap_name>
  <ap_serial>0002010809520954</ap_serial>
  <vns_name>Frasier Building</vns_name>
  <ssid>Library</ssid>
  <mac>00:50:56:C0:00:08</mac>
  <status>1</status>
  <policy>Unauthenticated</policy>
  <topology>VLAN 16</topology>
  <ingress_rc>n/a</ingress_rc>
  <egress_rc>n/a</egress_rc>
</response>
```

Sample get_vsa_xml.php Response

The exact format of the request message depends on whether the controller is configured to use encryption on its session control interface. If not the request will look something like:

```
http://192.168.18.7:54321/get_vsa_xml.php?token=T7vb1LdUZmsuY0q9V60Iww!!
&mu_ip_addr=192.168.22.105
```

or

```
https://192.168.18.7:54321/get_vsa_xml.php?token=T7vb1LdUZmsuY0q9V60Iww!!
&mu_ip_addr=192.168.22.105
```

where:

- T7vb1LdUZmsuY0q9V60Iww!! is replaced by the token appended to the redirection URL by the controller.
- 192.168.22.105 is replaced by the IP address of the station that is authenticating as learned from the ECP web server.
- 192.168.18.7:54321 should be replaced by the IP address and port configured in the controller Connection fields of the controller’s ECP configuration dialog box.

If the controller is configured for encryption on the session control interface, then the request URL contains a single parameter called “param”. In this case the request will look something like:

```
http://192.168.18.7:54321/get_vsa_xml.php?param=<parameters>
```

or

```
https://192.168.18.7:54321/get_vsa_xml.php?param=<parameters>
```

where

- 192.168.18.7:54321 should be replaced by the IP address and port configured in the controller "Connection" fields of the controller's ECP configuration dialog box.
- <parameters> is replaced by the encrypted string of request parameters.

MD5 Encryption: <parameters> argument

If MD5 encryption is used, the <parameters> argument is created by:

1. Create a single string of the form "token=<token>" or "token=<token>,mu_ip_addr=<ip_address>" where "<token>" is replaced by the token appended to the redirection URL and "<ip_address>" is replaced by the authenticating user's IP address.
2. Encrypt the output of step 1 using the shared key specified in the controller's ECP configuration page. If the ECP is implemented in PHP it can use the crypt_MD5 procedure, defined in "Appendix: Example PHP External Captive Portal: External Authentication".
3. Convert the binary output from step 2 to ASCII-encoded HEX. In PHP this can be done by calling the bin2hex procedure.

AES Encryption: <parameters> argument

If AES encryption is used, the <parameters> argument is created by:

1. Create a single string of the form token=<token> or token=<token>,mu_ip_addr=<ip_address> where <token> is replaced by the token appended to the redirection URL and <ip_address> is replaced by the authenticating user's IP address.
2. Encrypt the output of step1 using the shared key specified in the controller's ECP configuration page and AES.
3. Base64 encode the output of step 2.
4. Replace '+' with '-', '/' with '_' and '=' with '!' in the output of step 3.

   The following PHP snippet illustrates performing steps 3 and 4 in one line of PHP code:

   ```
   strtr(base64_encode($input), '+/=', '-_!');
   ```

Encoding Encrypted Parameters for HTTP/HTTPS Transmission

If encryption is enabled, the response sent by the controller is encrypted similarly. To decrypt the XML response document when MD5 encryption is used:

1. Convert the entire response document from ASCII-encoded hex to binary. In PHP this can be done by passing the response body through hex2bin.
2. Decrypt the output of step 1 using MD5 decryption. If the external captive portal is coded in PHP then it can use the decrypt_md5 procedure defined in PHP External Captive Portal: External Authentication on page 167.

Decrypting the XML Response Document

To decrypt the XML response document when AES encryption is used:

1 Replace '-' with '+', '_' with '/' and '!' with '='.

2 Base64 decode the output of step 1. The result will be a binary string.

3 Use AES and the shared key to decrypt the output of step 2. If the ECP is coded in PHP it can use the decrypt_aes() procedure defined in PHP External Captive Portal: External Authentication on page 167.

Code Snippet 3 "Getting the 'param' String Ready for Decryption" illustrates how to perform steps 1 & 2 in a single line of executable PHP code.

```
//decode: '-'=>'+', '_' => '/' , '!' => '='
base64_decode(strtr($param, '-_!', '+/='));
```

At this point the ECP script can parse the XML document and use its contents on the login page.

Note

The script get_vsa_xml.php runs on the controller and can require interactions with several components on the controller. As a result, when a large volume of users are authenticated, this script can place a heavy load on the controller and can have longer response times.

Approving the User

As discussed in Overview of Facilities and Usage on page 8, the user must submit his or her credentials to the ECP web server. Typically, credentials are submitted in an HTTP "post". The post invokes a script on the ECP web server passing the user's credentials to the script as arguments. The script is written by the customer and is adapted to the customer's specific requirements.

The script file can have any name. For example purposes, we will name the script "login.php". The script can be written in any programming language that supports web development, but for example purposes, the login script is written in PHP.

The main job login.php is to coordinate the user's browser, the back-end authentication server, and the controller. Login.php takes the submitted credentials, sends them to an authentication server, and waits for the server's reply. The exact steps taken here depend on the selected programming language, operating system, and the type of authentication server selected.

Once the authentication server has verified the user and potentially returned an access control role to assign to the user, the script needs to tell the controller that the user is authenticated and, optionally, the policy to apply. The script does this by calling the approval.php script exposed in the controller's session control API.

The ECP script should perform an HTTP get request for approval.php. The parameters to the request are listed in the table below.

[3] The table leaves out two legacy parameters "AuthStatus" and "auth_method" neither of which are used now. Older ECP implementations may include these parameters in their calls to approval.php. They can continue to do so; the parameters are simply ignored.

Table 4: Parameters to the Request for approval.php

Parameter Name	Parameter Value	Mandatory	Notes
token	Alphanumeric string	Yes	An identifier for the user's wireless session hosted on the controller that performed the redirection.
mu_ip_addr	IP Address	No	IP address of the authenticating user's device. It is an optional argument but it should be included if possible.
username	Alphanumeric string	No	The name that the user logged in with, if known. This is optional but is used in logs and accounting records so should be provided if at all possible.
opt27[4]	Base 10 Number	No	The maximum amount of time, in seconds, that the current session can last before being terminated. If not specified the default for the WLAN Service will be applied to this user.
filter	Alphanumeric string	No	The name of a role defined on the controller. If a filter parameter is not provided the controller will use the default authenticated role of the VNS that the authenticated user is accessing.
vns	Alphanumeric string	No	The name of the VNS that the authenticated user is accessing. There is no need to send this attribute.

Again, exactly how these parameters are sent to the controller's session control interface depends on whether encryption is enabled. If encryption is not enabled, then the query strings are simply appended to the approval.php URL in the standard way. For example:

```
http://192.168.18.7:54321/approval.php?token=T7vb1LdUZmsuY0q9V60Iww!!
&mu_ip_addr=10.10.10.16&opt27=36000&username=jsmith&filter=Guest_Access
```

If encryption is enabled on the interface for this ECP then all the parameters are put into a single string and encrypted. This single string is the value of the query parameter 'param'. An example of a call to approval.php with encryption enabled might look like:

```
http://192.168.18.7:54321/approval.php?
param=x9j26agXoYeJ5n5dPbajowbaih4PYnB5NE-
XMgqtCXUBSyCGuUlwkNoEQjjWNaDEYdHc8HYgKJfFjR2QjVn-Vw!!
```

If MD5 encryption is enabled create the 'param' string by

1 Taking each "name=value" query string and concatenate them together with comma separators. For example:

   token=T7vb1LdUZmsuY0q9V60Iww!!,mu_ip_addr=10.10.10.16,opt27=36000,username=jsmith

2 Encrypting the output of step 1 using MD5 and the shared key. If the ECP is implemented in PHP then it can call "crypt_md5" which is defined in PHP External Captive Portal: External Authentication on page 167.

3 Converting the binary output from step 2 to ASCII-encoded hexadecimal. In PHP this can be done by calling the bin2hex procedure.

   If AES encryption is enabled, create the param string by:

[4] In the RADIUS protocol option number 27 is the Session-Timeout attribute.

4 Taking each “name=value” query string and concatenate them together with comma separators. For example:

```
token=T7vb1LdUZmsuY0q9V60Iww!!,mu_ip_addr=10.10.10.16,opt27=36000,username=j
smith
```

5 Encrypting the output of step 4 using AES and the shared key. If the ECP is implemented in PHP then it can call “crypt_aes” which is defined in PHP External Captive Portal: External Authentication on page 167.

6 Base64 encode the output of step 5. In PHP this can be done by calling the base64_encode procedure.

7 Replace ‘+’ with ‘-‘, ‘/’ with ‘_’ and ‘=’ with ‘!’ in the output of step 6.

Composing the Login Page on page 33 illustrates how to do the last two steps in a single line of standard PHP code.

The controller will send an HTTP response to this request. The response takes the form of an XML document. The document contains two elements described in the table below.

Parameter Name	Parameter Value	Notes
token	Alphanumeric string	An identifier for the user’s wireless session hosted on the controller that performed the redirection. In some circumstances this could be used to match the response document to the request that produced it.
status	[0..9|14..16|18..21|99]	A return code. This is most likely to return 1 (Success) or 9 (General Failure) but can return other errors. See Facility Reference—Unsolicited Session Control Web Interface on page 49 for more details.

A full response document might look like the one in the code snippet below.

```
<?xml version="1.0"?>
<response>
  <token>T7vb1LdUZmsuY0q9V60Iww!!</token>
  <status>1</status>
</response>
```

Converting the Response to Binary from MD5 Encryption

Sample Document Returned in Response to an Approval.php Request

If the session control API is configured to use encryption for this interface, the response from the controller will be encrypted.

If MD5 decryption is used:

1 Convert the entire response document from ASCII-encoded hex to binary. In PHP this can be done by passing the response body through hex2bin.

2 Decrypt the output of step 1 using MD5 decryption. If the external captive portal is coded in PHP then it can use the decrypt_md5 procedure defined in PHP External Captive Portal: External Authentication on page 167.

At this point the login.php script running on the ECP can parse the XML document. The contents of the XML document might be written to the server logs. The arrival of the response document can be used to trigger delivery of the final page or server redirection to the user.

Converting the Response to Binary from AES Encryption

Sample Document Returned in Response to an Approval.php Request

If the session control API is configured to use encryption for this interface, the response from the controller will be encrypted.

To decrypt the XML response document when AES encryption is used:

1 Replace '-' with '+', '_' with '/' and '!' with '='.
2 Base64 decode the output of step 1. The result will be a binary string.
3 Use AES and the shared key to decrypt the output of step 2. If the ECP is coded in PHP it can use the decrypt_aes() procedure defined in PHP External Captive Portal: External Authentication on page 167.

At this point the login.php script running on the ECP can parse the XML document. The contents of the XML document might be written to the server logs. The arrival of the response document can be used to trigger delivery of the final page or server redirection to the user.

Conclusion

This chapter explained how to implement External Captive Portal with External Authorization (i.e., the ECP is responsible for making sure the user gets authenticated). Implementation involves the following main steps:

- Configuring the controller to redirect the blocked HTTP traffic of unauthenticated users to the ECP.
- Writing a script that receives redirected HTTP requests, saves critical information in the request, and displays a login page as a response to the redirection. The script can be called anything and can be written in any programming language.
- Writing a script that receives credentials from a user, authenticates them however it likes, and then tells the controller to change the authentication state, and possibly the role, of the authenticated user. The script can be called anything and can be written in any programming language.

Optionally, the "net-auth" script may use the controller's web session control API to request information about the station so that it can be displayed on a login page ("get_vsa_xml.php"). The "login" script must use the controller's web session control API to tell it when a user successfully authenticated and optionally, which role to apply to the user ("approval.php").

The appendices in this guide contain sample "net-auth" and "login" scripts implemented in PHP. They also include PHP source code for the utilities required by these two scripts but are not built into PHP.

The next chapter discusses a variation on External Captive Portal in which the controller is responsible for "proxying" between the ECP and the authentication server. This type of ECP is referred to as External Captive Portal with Internal Authentication, since the authentication step is internal to the controller.

3 Facility Reference—External Captive Portal: Internal Authorization

Overview

This chapter describes the implementation of External Captive Portal with Internal Authorization (ECPIA). ECPIA is a variant of captive portal where the captive portal web server is hosted on the controller and relies on the controller to handle authenticating and authorizing the station. The controller uses one or more external RADIUS servers to perform authentication and authorization of credentials.

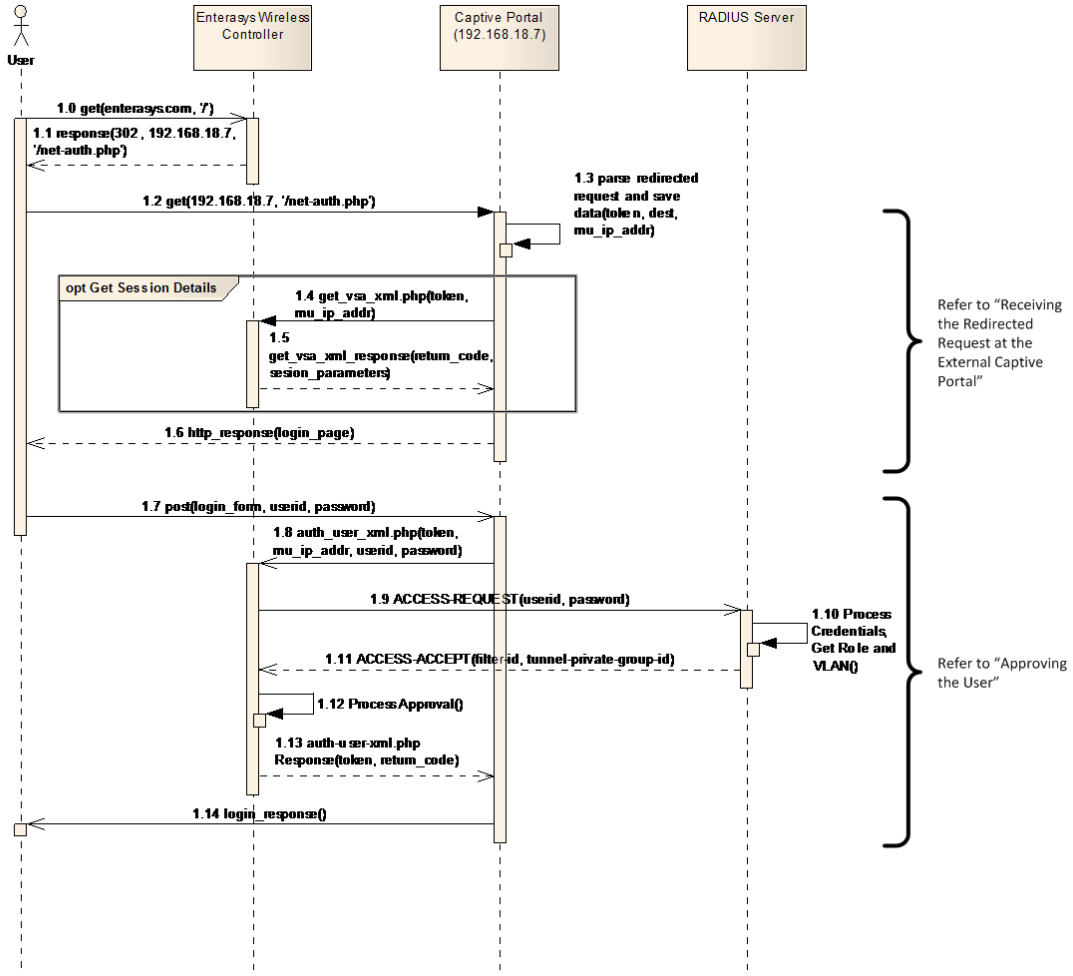


Figure 10: Flow of External Captive Portal with Internal Authorization

The above flowchart illustrates the flow of a user authentication using an external captive portal with internal authorization. The illustration expands on Figure 1: Simplified Captive Portal Authentication Sequence on page 10 and is very similar to Figure 6: Flow of External Captive Portal with External Authorization on page 23. In fact, the first eight steps in the sequence (1.0 to 1.7) are identical to the first eight steps of External Captive Portal with External Authorization. Those steps begin with the user requesting a URL served by a website that is blocked by his current policy, through receiving a login page to submitting his credentials using the login page. Those steps are covered in more detail in the Overview on page 40 and Receiving the Redirected Request at the External Captive Portal on page 44 sections of Facility Reference—External Captive Portal: External Authorization on page 22.

As Figure 10: Flow of External Captive Portal with Internal Authorization on page 41 shows, the difference between ECPEA and ECPIA begin at step 1.8. At step 1.8 of ECPEA the external captive portal sends the user's submitted credentials to the authentication server of its choice. At step 1.8 of ECPIA the external captive portal sends the user's credentials to the controller and lets the controller authenticate and authorize them. The ECP sends an HTTP/HTTPS get request for "auth_user_xml.php" on the configured controller's session control web interface. The "auth_user_xml.php" request and the controller's response to it are discussed in Approving the User on page 45.

As Figure 10: Flow of External Captive Portal with Internal Authorization on page 41 shows, the controller relies on an external RADIUS server to perform the authentication and authorization. The controller submits the user's credentials to the RADIUS server in a standard RADIUS ACCESS-REQUEST message. The RADIUS server responds with a standard RADIUS ACCESS-ACCEPT message if it wants the controller to allow the user on the network, or with a standard RADIUS ACCESS-REJECT message if it wants the controller to drop the user's session.

The controller must be configured to use a RADIUS server for authentication. Each WLAN Service that uses ECPIA authorization must be configured with at least one RADIUS server. This is discussed in Configuring the WLAN Service for ECPIA on page 43.

The RADIUS server may need to be configured to use additional authentication and authorization services. Configuring the RADIUS server to use such services is outside the scope of this document.

The controller applies the ACCESS-ACCEPT or ACCESS-REJECT to the target user's session as soon as the response is received. The ACCESS-ACCEPT optionally can include:

- A Filter-Id attribute, which contains the name of a role to apply to the authenticated station.
- A Tunnel-Private-Group-Id attribute, which contains an identifier for a VLAN. Depending on how the controller is configured, this VLAN could be used to select a role already defined on the controller, or it could be used in conjunction with the role specified in the Filter-ID attribute.
- A Login-Lat-Port attribute, which can be used to indicate that the user should be allowed on the network but still be treated as unauthenticated.
- A Session-Time attribute, which determines the maximum amount of time the user can remain on the network without re-authenticating.

These options are discussed further in Facility Reference—RADIUS Authentication and Authorization on page 105.

After the controller receives the response from the RADIUS server, it sends the results back to the External Captive Portal in an HTTP response. The response contains an XML document that describes the outcome of the authentication attempt. This information can be used to select the response to the user's login form submission.

The "auth_user_xml.php" request and the associated response are described in Approving the User on page 45.

Configuring the Controller to Redirect to an External Captive Portal

The conditions under which HTTP traffic is redirected to the External Captive Portal are identical for ECPEA and ECPIA. They are:

- It is blocked by a rule or default action in an access control role, and
- The user that sent the traffic is in an unauthenticated state, and
- The traffic is carried on a tunneled (Bridged at Controller, Routed) topology, and
- Some type of captive portal authentication has been configured for the user's WLAN Service, including a destination for redirected traffic.

The controller has to be configured to meet these requirements:

- A role that blocks some HTTP traffic and which allows traffic to the ECP must be defined.

- The role must be set as the default non-authenticated role of the VNS/WLAN Service that uses ECP authentication.
- The WLAN Service mentioned above must be configured to use ECP authentication and to communicate with the ECP through the HTTP session control interface.

The process of defining and using roles to redirect the HTTP traffic of unauthenticated users is covered in section “Causing HTTP Traffic to be Redirected using Roles” in Chapter “Facility Reference – External Captive Portal with External Authentication”.

The process of configuring a WLAN Service to use ECPIA for authentication is covered in the following section.

Configuring the WLAN Service for ECPIA

Configuring the WLAN Service ECPIA is almost identical to configuring it to use ECPEA. showFigure 11: Configuring a WLAN Service for ECPIA (Release 9.01) on page 44s the two configuration pages used to enable External Captive Portal support. Not surprisingly, it looks very much like in Facility Reference —External Captive Portal: Internal Authorization on page 40. The attributes that are configured for both types of ECP are described in that chapter. Also refer to that chapter for details regarding the common configuration items.

The big difference between configuring the controller for ECPIA and ECPEA is that ECPIA requires the WLAN Service to have a RADIUS server configured for authentication. To configure a RADIUS server:

1 From the WLAN > Auth & Acct tab, select a RADIUS server from the drop-down list of servers and click Use.
2 Click the Auth check box in the row containing the selected RADIUS server name.
3 From the WLAN Service configuration page, click Save...

[5] If the drop down list is empty a new RADIUS server can be configured for use on the controller by clicking the “New” button beside the list of RADIUS servers configured for use. When clicked, the “New” button opens a dialog box that can be used to define a new RADIUS server for use by the controller. The server will be added to the list of RADIUS servers configured on the controller. The WLAN Service can then be configured to use the new server.

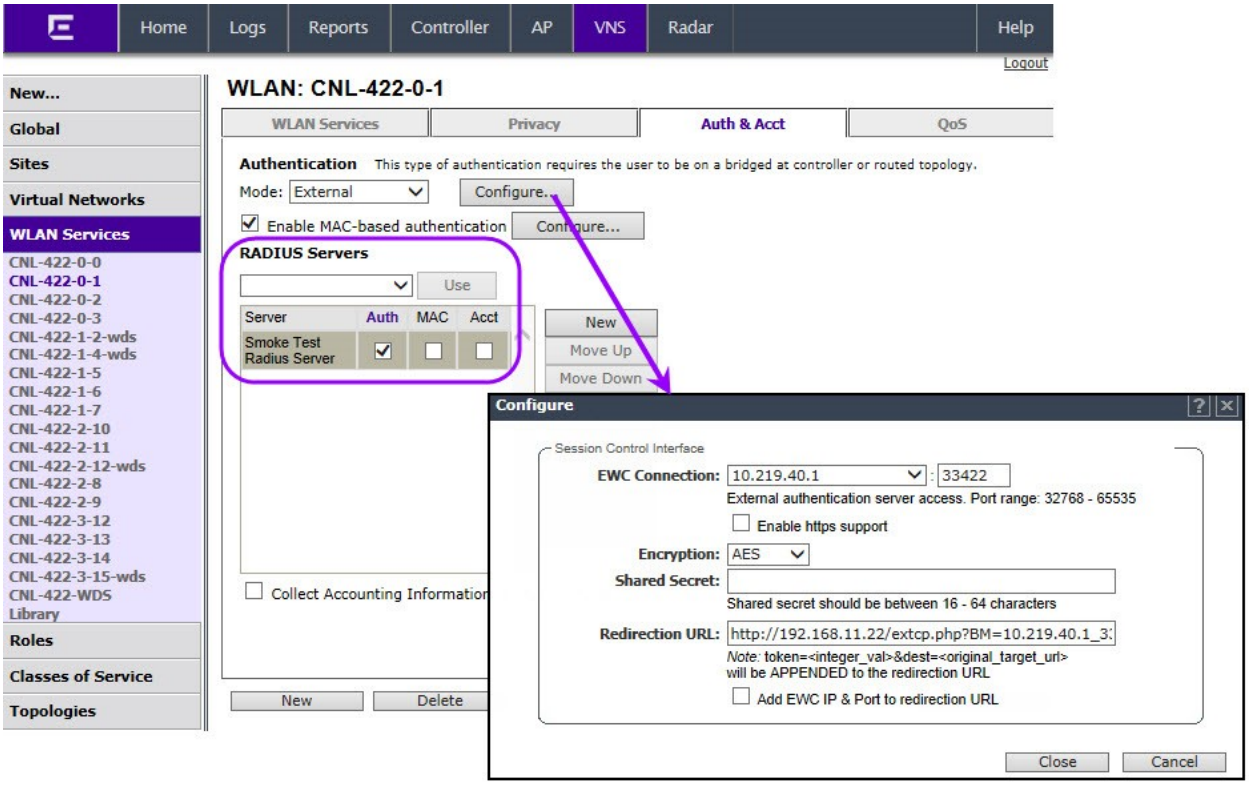


Figure 11: Configuring a WLAN Service for ECPIA (Release 9.01)

The controller can have up to three RADIUS servers configured for use with a single WLAN Service. For details and more RADIUS configuration options, please refer to the current *ExtremeWireless User Guide*.

Note

The controller will allow a WLAN Service using ECP authentication to be saved without a RADIUS server being specified. This is because the controller can't tell at configuration time whether ECPEA or ECPIA is being configured. So the administrator is responsible for configuring at least one RADIUS server when ECPIA will be used. There is no harm in configuring a RADIUS server for ECPEA; the configured server just isn't used by the WLAN Service.

Receiving the Redirected Request at the External Captive Portal

Customer-written scripts running on the external captive portal are responsible for:

- Receiving the redirected HTTP requests.
- Parsing the redirected HTTP requests and saving critical information contained in them for use later in the authentication process.
- Displaying a login form to the user.

These responsibilities are identical whether ECPIA or ECPEA is used. More details are available in section Receiving the Redirected Request at the External Captive Portal on page 31.

[6] The controller only uses one RADIUS server per WLAN Service for authentication and authorization. The additional RADIUS servers are used when the controller decides that the server it is using has failed.

Approving the User

The customer writes the ECP script that receives the user's credentials and forwards them to the controller for authentication. The script can be written in any programming language and can be called anything. This section assumes that the script is called "login.php" and is implemented in PHP.

The main responsibility of login.php is to send user credentials to the controller and wait for its response. Once login.php hears from the controller, it can send an HTTP response to the user.

The login.php script must send a get request to the controller for "auth_user_xml.php". The request can be transported as HTTP or HTTPS, depending on how the ECP session control interface is configured on the controller. The parameters to the auth_user_xml.php request are shown in the following table.

Parameter Name	Parameter Value	Mandatory	Notes
token	Alphanumeric String	Yes	An identifier for the user's wireless session hosted on the controller that performed the redirection.
username	Alphanumeric String	Yes	The name that the user logged in with, if known. This is optional but is used in logs and accounting records so should be provided if at all possible.
password	Alphanumeric String	Yes	The password submitted by the user. It is in clear text (i.e., it is not hashed or encrypted).
mu_ip_addr	Alphanumeric String	No	IP address of the authenticating user's device. It is an optional argument but it should be included if possible.

The format of the auth_user_xml.php request depends on the type of encryption that has been selected. If no additional encryption is configured then the request URL will look like one of the following:

```
http://192.168.18.7:54321/auth_user_xml.php?token=T7vb1LdUZmsuY0q9V60Iww!!
&username=originaldude15&password=ChangeMe&mu_ip_addr=192.168.12.105
```

or

```
https://192.168.18.7:54321/auth_user_xml.php?token=T7vb1LdUZmsuY0q9V60Iww!!
&username=originaldude15&password=ChangeMe&mu_ip_addr=192.168.12.105
```

where

- 192.168.18.7:54321 is replaced by the IP address and TCP port number of the controller's session control interface. Each WLAN Service can have a different IP address-TCP port pair for use by the ECP.
- T7vb1LdUZmsuY0q9V60Iww!! is replaced by the token created by the controller for this user's session.
- originaldude15 is replaced by the user ID submitted by the authenticating user.
- ChangeMe is replaced by the password entered by the user.
- 192.168.12.105 is replaced by the IP address of the user as reported by the ECP's web server.

If encryption is enabled then the URL contains a single parameter called “param”. The request URL would look something like:

```
http://192.168.18.7:54321/auth_user_xml.php?
param=x9j26agXoYeJ5n5dPbajowbaih4PYnB5NE-
XMgqtCXUBSyCGuUlwkNoEQjjWNaDEYdHc8HYgKJfFjR2QjVn-Vw!!
```

or

```
https://192.168.18.7:54321/auth_user_xml.php?
param=x9j26agXoYeJ5n5dPbajowbaih4PYnB5NE-
XMgqtCXUBSyCGuUlwkNoEQjjWNaDEYdHc8HYgKJfFjR2QjVn-Vw!!
```

If the controller is configured to use MD5-based encryption then the value of the param string is created by:

1 Concatenating the individual query strings into a single string, using commas as separators. For example:

```
token=T7vb1LdUZmsuY0q9V60Iww!!,username=originaldude15,password=ChangeMe,mu_
ip_addr=192.168.12.105
```

2 Encrypting the output of the previous step using MD5 and the shared password configured in the WLAN Service’s ECP settings. If the ECP is implemented in PHP, then the login script can use the crypt_md5() procedure defined in PHP External Captive Portal: Internal Authentication on page 183.

3 Converting the binary output from 2 on page 46 to ASCII-encoded HEX. In PHP this can be done by calling the bin2hex procedure.

If the controller is configured to use AES encryption, then the value of the param string is created by:

4 Concatenating the individual query strings into a single string, using commas as separators. For example:

```
token=T7vb1LdUZmsuY0q9V60Iww!!,username=originaldude15,password=ChangeMe,mu_
ip_addr=192.168.12.105
```

5 Encrypting the output of 4 on page 46 using the shared key specified in the controller’s ECP configuration page and AES.

6 Base64 encode the output of 5 on page 46.

7 Replacing ‘+’ with ‘-‘, ‘/’ with ‘_’ and ‘=’ with ‘!’ in the output of 6 on page 46

Composing the Login Page on page 33 shows how to do step 6 and step 7 in a single line of PHP code.

The controller sends an HTTP response once it has finished authenticating the user. The reply consists of a short XML document containing the two attributes described in the table below.

Element Name	Element Value	Notes
token	Alphanumeric String	An identifier for the user's wireless session hosted on the controller that performed the redirection. In some circumstances this could be used to match the response document to the request that produced it.
Status	[0..9|14..16|18..21|99]	A return code. '1' indicates a successful authentication. Any other value indicates that either an error occurred or the user failed authentication. See Facility Reference—Unsolicited Session Control Web Interface on page 49 for more details.

A full response document will look like the one in Approving the User on page 36.

Again the exact format of the reply document depends on whether encryption was enabled. If no encryption was selected when the ECP interface was configured, then the document will be sent in the clear, ready for XML parsing.

Converting the Response to Binary from MD5 Encryption

Related to Approving the User - For Internal Authorization

If MD5 encryption was selected, then the reply document must be converted to plain text before it can be parsed. To convert the reply to plain text:

1 Convert the entire response document from ASCII-encoded hex to binary. In PHP this can be done by passing the response body through pack('H*',...).
2 Decrypt the output of the previous step using MD5 decryption. If the external captive portal is coded in PHP then it can use the decrypt_md5 procedure defined in the appendix PHP External Captive Portal: Internal Authentication on page 183.

At this point the document is ready for XML parsing. The document can be logged if desired. The status field can be used to decide the type of feedback to give the user.

Converting the Response to Binary from AES

Related to Approving the User - For Internal Authorization.

If the controller is configured to use AES on this ECP interface, then the reply must be converted to plain text using a different procedure:

1 Replace '-' with '+', '_' with '/' and '!' with '='.
2 Base64 decode the output of step 1. The result will be a binary string
3 Use AES and the shared key to decrypt the output of step 2. If the ECP is coded in PHP it can use the decrypt_aes() procedure defined in PHP External Captive Portal: Internal Authentication on page 183.

At this point the document is ready for XML parsing. The document can be logged if desired. The status field can be used to decide the type of feedback to give the user.

Conclusion

This chapter explained how to implement External Captive Portal with Internal Authorization (i.e., the ECP hands over responsibility for authenticating the user to the controller). Implementation involves these steps:

- Configuring the controller to redirect the blocked HTTP traffic of unauthenticated users to the ECP. The steps are similar regardless of whether external authorization or internal authorization will be used. The key difference is that when internal authorization is used the WLAN Service using ECPIA must be configured with at least one RADIUS server for authentication.
- Writing a script that receives redirected HTTP requests, saves critical information in the request, and displays a login page as a response to the redirection. The script can be called anything and can be written in any programming language. The script does not depend on whether external authorization or internal authorization is selected.
- Writing a script that receives the user's credentials, forwards them to the controller for processing, and waits for the controller to reply. The reply indicates whether the authentication was successful. If it was not, the return code provides hints as to why the authentication failed. The script can use this feedback to decide how to compose a response page for the user's login attempt.

The script that composes the login page may use the controller's web session control API to request information about the station ("get_vsa_xml.php"). The "login" script must use the controller's session control web interface to ask the controller to authenticate the user ("auth_user_xml.php").

The appendices contain sample "net-auth" and "login" scripts implemented in PHP. The appendices also include PHP source code for the utilities required by these two scripts but which aren't built into PHP.

This chapter and the previous one dealt with using the controller's HTTP/HTTPS session control interface to configure a user's session at login time. There are also many scenarios in which it is necessary to alter a station's access, possibly terminating a session access after they have authenticated. The next section discusses the portion of the controller's session control web interface devoted to management of a user's session at times when the user is not authenticating.

[7] And in some cases it may be necessary to change their status before they have fully authenticated.

4 Facility Reference—Unsolicited Session Control Web Interface

Overview

The Unsolicited/Asynchronous Session Control Web Interface allows an External Captive Portal (ECP) or authorization server to manage a user's session. This interface allows the caller to:

- Query the state of a user's session.
- Add a user's device to the controller's blacklist.
- Change a user's role or authentication state.
- Change the URL to which an unauthenticated user is redirected for captive portal authentication.

This part of the API is referred to as "unsolicited" because it can be used any time during a user's session regardless of whether the subject of the message is authenticating. This allows an administrator to do things like:

- Terminate the session of an authenticated user before the user's session expires.
- Change the role assigned to a session based on information obtained after the user authenticated.

This API can be used to manage sessions that were not authenticated using External Captive Portal. All sessions on a controller can be managed using the unsolicited HTTP session control interface. It does not matter whether the session has authenticated or what type of authentication was applied to the session.

The API supports either HTTP or HTTPS messaging as configured on the controller's WLAN Service Authentication and Accounting tab.

The controller implements its unsolicited HTTP session control interface in PHP. The API is contained in a single file called "event.php". Not surprisingly, the unsolicited session control web interface is referred to as "event.php" for short.

Any programming language that can send web get requests and receive web responses can use the unsolicited session control web interface.

Requirements

At least one WLAN Service using ECP authentication must be defined on the controller in order to use event.php. The WLAN Service does not need to be part of an enabled VNS. The controller connection IP address and port configured for the WLAN Service can receive event.php requests. Any WLAN Service configured for ECP can receive event.php requests for any user with a session on the controller. This includes users who:

- May be using different WLAN Services on the controller.
- May not have authenticated yet.
- May have authenticated using some form of authentication other than ECP.

Note

Only the controller can receive and process web session control messages. If the controller can communicate with the AP hosting the user identified in the control message then the controller will apply the command to the user. It does not matter whether the user is associated to a site-based AP.

Invoking Event.php

All calls to event.php take the form of an HTTP get request. The parameters to the request are appended as HTTP query strings. An HTTP query string has the form "parameter-name=parameter-value". If the session control interface is not using encryption then a call to event.php would look like:

```
Sample Request for Event.php without Encryption
http://192.168.18.7:33333/event.php?type=1&value=10.10.10.15
```

where

- `192.168.18.7:33333` is the connection address and port specified in the configuration of a WLAN Service that uses ECP authentication. The address and port are configured on the ECP configuration dialog of a WLAN Service's Authentication and Accounting tab.
- `type=1` indicates the type of service being requested. Exactly one type value is allowed per request. The value of this parameter is a number in the range of 1 to 12 (although 11 is not used currently). The service that each number requests is identified in Table 5: Services Offered by Event.php on page 50.
- `value=...` is a query string containing all the arguments required to invoke the requested service. The list of arguments depends on the type of service being requested. If more than one argument is present they must be separated by commas (',').

Table 5: Services Offered by Event.php

Code	Service Type
1	Disassociate a user, identified by the device's IP address.
2	Disassociate a user, identified by the user's device's MAC address.
3	Change the role, and optionally the authentication state of a user who is identified by the device's IP address.

Table 5: Services Offered by Event.php (continued)

Code	Service Type
4	Change the role, and optionally the authentication state (authenticated, unauthenticated) of a user who is identified by the device's MAC address.
5	Place a user's device on a blacklist. The device is identified by its MAC address.
6	Retrieve information about an active user's session identified by the device's MAC address.
7	Change the URL to which a specific user will be redirected for captive portal authentication. Optionally the user's authentication state (authenticated, unauthenticated) can be changed. The user is identified by the device's IP address.
8	Change the URL to which a specific user will be redirected for captive portal authentication. Optionally the user's authentication state (authenticated, unauthenticated) can be changed. The user is identified by the device's MAC address.
9	Change the role currently assigned to a user and the URL to which a specific user will be redirected after a successful authentication. Optionally the user's authentication state (authenticated, unauthenticated) can be changed. The user is identified by the device's IP address.
10	Change the role currently assigned to a user and the URL to which a specific user will be redirected for captive portal authentication. Optionally, change the user's authentication state. The user is identified by the device's MAC address.
12	Retrieve information about an active user's session based on the device's MAC address.

Note

The order in which multiple items are listed in the value= query parameter is critical. The controller identifies each item by its position in the comma-separated sequence. The different parts of the sequence must be present in the order they are listed in the "Request Parameters" section of the service description. Failure to use the specified ordering will cause the arguments to be misinterpreted.

The exact format of the parameters for an 'event.php' request depends on the type of encryption selected for the interface. If encryption is not selected, then the URL's query strings should be sent in clear text shown above.

If the controller's ECP interface is using legacy encryption then the event.php request contains a single query string named 'param'. The 'param' string is constructed by:

1 Concatenating the type and value query strings into a single string, using commas as separators. For example: type=1,value=10.10.10.15
2 Encrypting the output of the previous step using MD5 and the shared password configured in the WLAN Service's ECP settings. If the ECP is implemented in PHP then it can use the crypt_md5 procedure defined in the script "crypt_md5.php" included in PHP External Captive Portal: Internal Authentication on page 183.
3 Converting the binary output from the step above to ASCII-encoded HEX. In PHP this can be done by calling the bin2hex procedure.

If the controller is configured to use AES encryption then the value of the param string is created by:

1 Concatenating the individual query strings into a single string, using commas as separators. For example: type=1,value=10.10.10.15
2 Encrypting the output of the step above using the shared key specified in the controller's ECP configuration page and AES.
3 Base64 encode the output of step 2.
4 Replacing '+' with '-', '/' with '_' and '=' with '!' in the output of step 3.

The code snippet below shows what a typical AES-encrypted event.php request looks like.

Sample Request for Event.php with Encryption

```
http://192.168.18.7:33333/event.php?param=x9j26agXoYeJ5n5dPbajowbaih4PYnB5NE-
XMgqtCXUBSyCGuUlwkNoEQjjWNaDEYdHc8HYgKJfFjR2QjVn-Vw!!
```

Response from Event.php

The event.php response takes the form of an XML document. In most cases the response document contains a single element called 'status'. The 'status' element contains the return code from event.php. In general a status code other than 1 means that the request was not successfully processed. The status code is a hint as to why request processing failed. The possible values of the return code depend on which service was requested. The complete set of return codes is listed in Siemens VSA Dictionary in FreeRadius Format on page 156.

The code snippet below contains a sample response from event.php. The exact format of the response depends on whether encryption is enabled on the ECP interface that the request was received on.

The following example shows the format of the response when no encryption is used:

Sample Response from event.php – No Encryption

```
<?xml version="1.0"?>
<response>
    <status>1</status>
</response>
```

There are two principle variations on the response document format:

1 When the request is a query for details about a user's session (request types 6 and 12). The response is an XML document containing a list of details about the user's session. Response to a Request with Valid Query Parameters - Found Subject Device. User does not have a known identifier yet on page 66 contains an example of this type of response.
2 When the target IP or MAC address does not have a session on the receiving controller the document will look like the one in Response to a Request with Valid Query Parameters - Found Subject Device. User does not have a known identifier yet on page 66

If legacy encryption is enabled then the returned document must be decrypted before it is parsed. To convert a legacy-encrypted document to plain text:

1 Convert the entire response document from ASCII-encoded hex to binary. In PHP this can be done by passing the response body through a call to pack('H*',...).
2 Decrypt the output of step 1 using MD5 decryption. If the external captive portal is coded in PHP, it can use the decrypt_md5 procedure defined in crypt_md5.php on page 181.

If the controller is configured to use AES on this ECP interface then the reply must be converted to plain text using a different procedure:

1 Replace '-' with '+', '_' with '/' and '!' with '='
2 Base64 decode the output of step 1. The result will be a binary string.
3 Use AES and the shared key to decrypt the output of step 2. If the ECP is coded in PHP it can use the crypt_aes() procedure defined in crypt_aes.php on page 180, included in PHP External Captive Portal: External Authentication on page 167.

At this point the document is ready for XML parsing. The document can be logged if desired. The status field can be used to decide the type of feedback to give the user.

Event.php Service Descriptions

The following sections provide more detail on each of the services offered by event.php.

Disassociate a User Identified by his Device's IP Address

This service terminates the identified user's session and disassociates the user from the AP. Sessions for the user are terminated on both the controller and AP, though the user can associate to the AP again after being disconnected. If authentication is enabled on the WLAN Service, the user will need to authenticate again. If accounting is enabled, session information is written to an accounting stop record.

Service Number 1

Request Parameters:

1 IP address of the device belonging to the user who is to be disassociated.

Table 6: Primary Return Codes

Return Code	Meaning
1	Success.
3	Failed to disassociate the owner of the IP address. More information may be available in the controller events. The most likely cause of the failure is that the device with the given IP address does not have a session on the controller to which the message was sent. This can happen if an incorrect IP address was sent, the user terminated the session, the session already timed out, or fast failover is enabled and the station roamed to an AP managed by the controller's availability partner.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=1&value=10.10.10.15
```

or

```
https://192.168.18.7:33333/event.php?type=1&value=10.10.10.15
```

The sample attempts to terminate the session of a user whose device has IP address 10.10.10.15.

Sample Response (No encryption)

```
Sample Response to a Disassociate Request
<?xml version="1.0"?>
<response>
    <status>1</status>
</response>
```

Disassociate a User Identified by his Device's MAC Address

This service terminates the identified user's session and disassociates the user from the AP. The user is identified by his or her device's MAC address. Sessions for the user are terminated on both the controller and AP. If authentication is enabled on the WLAN Service the user is using, he or she will need to authenticate again. If accounting is enabled, session information is written to an accounting stop record. The user can associate to the AP again.

Service Number 2

Request Parameters:

1 MAC address of the device to be disassociated. The MAC address string uses colons to separate the octets composing the MAC. The following is an example of an acceptable MAC address string: 00:26:B9:DE:27:CB

Table 7: Primary Return Codes

Return Code	Meaning
1	Success.
3	Failed to disassociate the owner of the IP address. More information may be available in the controller events. The most likely cause of the failure is that the device with the given IP address does not have a session on the controller to which the message was sent. This can happen if an incorrect IP address was sent, the user terminated the session, the session already timed out, or fast failover is enabled and the station roamed to an AP managed by the controller's availability partner.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=2&value=00:26:B9:DE:27:CB
```

or

```
https://192.168.18.7:33333/event.php?type=2&value=00:26:B9:DE:27:CB
```

The sample attempts to terminate the session of a user whose device has MAC address 00:26:B9:DE:27:CB.

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 2 “event.php” request.

Change the Role and Authentication State of a User Identified by Device IP Address

This service assigns the indicated role to the station. The new role completely replaces the one the station was assigned prior to the controller receiving this request.

The authentication state of a user is either authenticated or not-authenticated. This service will set the user’s authentication state to the value specified in the request. The newly assigned state completely replaces the old state. The user’s session is not terminated. If the user’s state was changed to unauthenticated, he or she may have to authenticate again. If the authentication state argument is missing, the station’s role is changed but it retains its authentication state.

Note

One use for this service is to force a user to interact with an assessment server even though the station may have authenticated already. By setting the user’s authentication state to unauthenticated, the service causes the station to be redirected to the assessment server when some of the user’s HTTP traffic is blocked. Obviously, the role assigned to the station must block some HTTP traffic while making the assessment server accessible. The assessment server can display a web page to the user explaining what the user has to do to get his device assessed and his access restored.

Service Number 3

Request Parameters:

1 IP address of the device belonging to the user whose role is to be changed.

2 The name of the role defined on the controller which is to be assigned to the station. The role name is case sensitive. It must exactly match the name of a role defined on the controller or request processing will fail and return a status code of 21.

3 The authentication state to assign to the session. This is an optional parameter and can be absent. The table below shows how the authentication state argument is interpreted.

Code for Authentication State	Interpretation
1	Treat the user’s session as authenticated.
2	Treat the user’s session as unauthenticated.

Table 8: Primary Return Codes

Code for Authentication State	Interpretation
1	Success.
3	General failure.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.
15	Authentication in progress. The user has been redirected to a captive portal during the last minute or so. Try again in a few minutes or disassociate the target user.
21	The role name argument does not match the name of a role defined on the controller.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=3&value=10.10.10.15,default,2
```

or

```
https://192.168.18.7:33333/event.php?type=3&value=10.10.10.15,default,2
```

The sample attempts to change the role assigned to the user with IP address 10.10.10.15 to “default” and reset the user’s authentication state to unauthenticated. For this request to succeed, a session for a user with IP address 10.10.10.15 must exist on the controller receiving the request and a role named “default” must be defined on that controller.

The sample below attempts to apply the role named ‘admin’ to the session of the user whose device has IP address 10.10.10.15. The session’s authentication state is not changed.

```
http://192.168.18.7:33333/event.php?type=3&value=10.10.10.15,admin
```

or

```
https://192.168.18.7:33333/event.php?type=3&value=10.10.10.15,admin
```

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 3 “event.php” request.

Change the Role and Authentication State of a User Identified by Device MAC Address

This service assigns the indicated role to the station. The new role completely replaces the one the station was assigned prior to the controller receiving this request. The target user is identified by the MAC address of his or her device.

The authentication state of a user is either authenticated or unauthenticated. This service will set the user’s authentication state to the value specified in the request. The newly assigned state completely

replaces the old state. The user's session is not terminated. If the user's state was changed to unauthenticated, he or she may have to re-authenticate. If the authentication state argument is missing, the station's role is changed but it retains its authentication state.

Service Number 4

Request Parameters:

1 MAC address of the device belonging to the user whose role is to be changed. The bytes of the MAC address are colon-separated. The following is an example of a correctly formatted MAC address: `00:26:B9:DE:27:CB`.

2 The name of the role defined on the controller that is to be assigned to the station.

3 The authentication state to assign to the session. This is an optional parameter and can be absent. The table below shows how the authentication state argument is interpreted.

Code for Authentication State	Interpretation
Not present	The user's authentication state will not be changed.
1	Set the user's session to authenticated state.
2	Set the user's session to unauthenticated state.

Table 9: Primary Return Codes

Return Code	Meaning
1	Success.
3	General failure.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.
15	Authentication in progress. The user has been redirected to a captive portal during the last minute or so. Try again in a few minutes or disassociate the target user.
21	The role name argument does not match the name of a role defined on the controller.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=4&value=00:26:B9:DE:27:CB,default,2
```

or

```
https://192.168.18.7:33333/event.php?type=4&value=00:26:B9:DE:27:CB,default,2
```

The sample attempts to change the role assigned to the user with MAC address 00:26:B9:DE:27:CB to "unauthenticated" and reset the user's authentication state to unauthenticated. For this request to succeed a session for a user with MAC address 00:26:B9:DE:27:CB must exist on the controller receiving the request and a role named "default" must be defined on that controller.

The sample below attempt to apply the role named 'admin' to the session of the user whose device has MAC address 00:26:B9:DE:27:CB. The session's authentication state is not changed.

```
http://192.168.18.7:33333/event.php?type=4&value=00:26:B9:DE:27:CB,admin
```

or

```
https://192.168.18.7:33333/event.php?type=4&value=00:26:B9:DE:27:CB,admin
```

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 4 "event.php" request.

Change the Redirection Destination for a User Identified by his Device's IP Address

This service changes the URL to which the device with the specified IP address will be redirected for captive portal authentication.

Optionally, the request can assign an authentication state to the target user. If an authentication state is specified, it will replace the authentication state assigned to the user by the controller.

Service Number 7

Request Parameters:

1 IP address of the device belonging to the user whose redirection URL is to be changed.
2 The URL to which the user's browser will be redirected after a successful authentication.
3 The authentication state to assign to the user's session. This is an optional argument and can be absent. The table below shows how the authentication state argument is interpreted.

Code for Authentication State	Interpretation
Not present	The user's authentication state will not be changed.
1	Set the user's session to authenticated state.
2	Set the user's session to unauthenticated state.

Table 10: Primary Return Codes

Return Code	Meaning
1	Success.
3	General failure.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.
15	Authentication in progress. The user has been redirected to a captive portal during the last minute or so. Try again in a few minutes or disassociate the target user.

\

Sample Request (No encryption)

The sample below attempts to change the redirection URL for the user of 10.10.10.15 to “http://www.xyz.com/notices.jsp” and set the user to unauthenticated state.

```
http://192.168.18.7:33333/event.php?type=7&value=10.10.10.15,http://
www.xyz.com/notices.jsp,2
```

or

```
https://192.168.18.7:33333/event.php?type=7&value=10.10.10.15,http://
www.xyz.com/notices.jsp,2
```

The sample below attempts to change the redirection URL for the user of 10.10.10.15 to “http://xyz.com/notices.jsp”. The session’s authentication state is not changed.

```
http://192.168.18.7:33333/event.php?type=7&value=10.10.10.15,http://
www.xyz.com/notices.jsp
```

or

```
https://192.168.18.7:33333/event.php?type=7&value=10.10.10.15,http://
www.xyz.com/notices.jsp
```

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 7 “event.php” request.

Note

Changes to the redirection URL made via event.php have no effect unless the target user is in an unauthenticated state. Any event.php request that can change a user’s redirection URL also can change the user’s authentication state to unauthenticated.

Change the Redirection Destination for a User Identified by Device MAC Address

This service changes the URL to which the device with the specified MAC address will be redirected for captive portal authentication. Optionally the request can change the authentication state of the target user. If an authentication state is specified it will replace the authentication state assigned to the user by the controller prior to receiving this request.

Service Number 8

Request Parameters:

1. MAC address of the device belonging to the user whose redirection URL is to be changed. The MAC address is a colon-separated sequence like 00:26:B9:DE:27:CB.
2. The URL to which the user’s browser will be redirected after a successful authentication.
3. The authentication state to assign to the user’s session. This is an optional argument and can be absent. The table below shows how the authentication state argument is interpreted.

Code for Authentication State	Interpretation
Not present	The user's authentication state will not be changed.
1	Set the user's session to authenticated state.
2	Set the user's session to unauthenticated state.

Table 11: Primary Return Codes

Return Code	Meaning
1	Success.
3	General failure.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.
15	Authentication in progress. The user has been redirected to a captive portal during the last minute or so. Try again in a few minutes or disassociate the target user.

Sample Request (No encryption)

The sample below attempts to change the redirection URL for the user of 00:26:B9:DE:27:CB to "http://www.xyz.com/notices.jsp" and set the user to unauthenticated state.

```
http://192.168.18.7:33333/event.php?type=8&value=00:26:B9:DE:27:CB,http://
www.xyz.com/notices.jsp,2
```

or

```
https://192.168.18.7:33333/event.php?type=8&value=00:26:B9:DE:27:CB,http://
www.xyz.com/notices.jsp,2
```

The sample below attempts to change the redirection URL for the user of 00:26:B9:DE:27:CB to "http://xyz.com/notices.jsp". The session's authentication state is not changed.

```
http://192.168.18.7:33333/event.php?type=8&value=00:26:B9:DE:27:CB,http://
www.xyz.com/notices.jsp
```

or

```
https://192.168.18.7:33333/event.php?type=8&value=00:26:B9:DE:27:CB,http://
www.xyz.com/notices.jsp
```

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 8 "event.php" request.

Change the Role and Redirection Destination for a User Identified by Device IP Address

This request combines the functionality of service request type 3 and 7 in a single request. The user who is the subject of the request is identified by his or her device's IP address. The user's authentication state may be changed as well.

Service Number 9

Request Parameters:

1 IP address of the device belonging to the user whose role and redirection URL are to be changed.
2 The name of a role defined on the controller that is to be applied to the specified user's traffic.
3 The URL to which the user's browser will be redirected after a successful authentication.
4 The authentication state to assign to the user's session. This is an optional argument and can be absent. The table below shows how the authentication state argument is interpreted.

Code for Authentication State	Interpretation
Not present	The user's authentication state will not be changed.
1	Set the user's session to authenticated state.
2	Set the user's session to unauthenticated state.

Table 12: Primary Return Codes

Return Code	Meaning
1	Success.
3	General failure.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.
15	Authentication in progress. The user has been redirected to a captive portal during the last minute or so. Try again in a few minutes or disassociate the target user.
21	The role name in the request does not correspond to the name of a role defined on the controller.

Sample Request (No encryption)

The sample below attempts to change the access control role assigned to the user with IP address 10.10.10.15 to "default", his or her redirection URL to "http://www.xyz.com/notices.jsp" and set his or her session to unauthenticated state.

```
http://192.168.18.7:33333/event.php?type=9&value=10.10.10.15,default,http://
www.xyz.com/notices.jsp,2
```

or

```
https://192.168.18.7:33333/event.php?type=9&value=10.10.10.15,default,http://
www.xyz.com/notices.jsp,2
```

The sample below attempts to change the redirection URL for the user of 10.10.10.15 to “http://www.xyz.com/notices.jsp” and his or her access control role to “admin.” The session’s authentication state is not changed.

```
http://192.168.18.7:33333/event.php?type=9&value=10.10.10.15,admin,http://
www.xyz.com/notices.jsp
```

or

```
https://192.168.18.7:33333/event.php?type=9&value=10.10.10.15,admin,http://
www.xyz.com/notices.jsp
```

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 9 “event.php” request.

Change the Role and Redirection Destination for a User Identified by Device MAC Address

This request combines the functionality of service request type 4 and 8 in a single request. The target user is identified by the device’s MAC address. The user’s authentication state may be changed as well.

Service Number 10

Request Parameters:

1 MAC address of the device belonging to the user whose role and redirection URL are to be changed.
2 The name of a role defined on the controller that is to be applied to the specified user’s traffic.
3 The URL to which the user’s browser will be redirected after a successful authentication.
4 The authentication state to assign to the user’s session. This is an optional argument and can be absent. The table below shows how the authentication state argument is interpreted.

Code for Authentication State	Interpretation
Not present	The user’s authentication state will not be changed.
1	Set the user’s session to authenticated state.
2	Set the user’s session to unauthenticated state.

Table 13: Primary Return Codes

Return Code	Meaning
1	Success.
3	General failure.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.

Table 13: Primary Return Codes (continued)

Return Code	Meaning
15	Authentication in progress. The user has been redirected to a captive portal during the last minute or so. Try again in a few minutes or disassociate the target user.
21	The role name in the request does not correspond to the name of a role defined on the controller.

Sample Request (No encryption)

The sample below attempts to change the access control role assigned to the user with MAC address 00:26:B9:DE:27:CB to "default," the redirection URL to "http://www.xyz.com/notices.jsp" and set his or her session to unauthenticated state.

```
http://192.168.18.7:33333/event.php?type=10&value=00:26:B9:DE:
27:CB,default,http://www.xyz.com/notices.jsp,2
```

or

```
https://192.168.18.7:33333/event.php?type=10&value=00:26:B9:DE:
27:CB,default,http://www.xyz.com/notices.jsp,2
```

The sample below attempts to change the redirection URL for the user of 00:26:B9:DE:27:CB to "http://www.xyz.com/notices.jsp" and his or her access control role to "admin." The session's authentication state is not changed.

```
http://192.168.18.7:33333/event.php?type=10&value=00:26:B9:DE:
27:CB,admin,http://www.xyz.com/notices.jsp
```

or

```
https://192.168.18.7:33333/event.php?type=10&value=00:26:B9:DE:
27:CB,admin,http://www.xyz.com/notices.jsp
```

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 10 "event.php" request.

Blacklist a Client Device

This service adds a specified MAC address to a controller's blacklist. A device with a blacklisted MAC address cannot associate to any AP managed by the controller. Because the association fails, the device will not be able to send traffic through the controller's AP.

The blacklist is global to the controller, meaning that a device on the blacklist is blocked from associating to all APs managed by the controller, regardless of which VNS/SSIDs they serve.

A user with an active session at the moment his or her device is added to the blacklist will have his or her session terminated. The user will not be able to get back on the network because the device is now blacklisted.

Devices manually added to the blacklist, or added through this “event.php” facility stay on the blacklist until they are explicitly removed by an administrator. The Unsolicited Session Control interface does not define a request that can be used to take devices off the blacklist.

Service Number 5

Request Parameters:

1 MAC address of the device to be blacklisted.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=5&value=00:26:B9:DE:27:CB
```

or

```
https://192.168.18.7:33333/event.php?type=5&value=00:26:B9:DE:27:CB
```

The sample attempts to add MAC address 00:26:B9:DE:27:CB to the controller’s global blacklist.

Sample Response (No encryption)

Sample Response from event.php – No Encryption on page 52 is an example of a possible response to a type 5 “event.php” request.

Retrieve Information about a User’s Session Identified by Device MAC Address

This request collects information about the session of the device with the MAC address specified in the request’s value parameter. If the request is successful the response takes the form of an XML document containing the attributes of the identified device. The response to an unsuccessful request contains a status code or a string indicating why the query failed.

Table 14: Device Attributes Returned from a Successful Query

XML Element	Meaning
vns_id	The controller’s internal identifier number for the VNS.
ap_serial	The serial number that uniquely identifies the Extreme Networks Access Point to which the user is associated.
ssid	The SSID that the user’s device associated to.
ip_addr	The IP address of the identified user’s device.
mac_addr	The MAC address of the identified user’s device.
user	The user ID/user name of the identified user. It can be empty.
client_status	“Validated” or “Not Validated”. “Validated” means the user is considered authenticated. “Not Validated” means the user is considered unauthenticated.
session_start	The time at which the user session started. It is in the format DD MMMM YYYY HH:MM:SS where MMMM is the complete name of the month and HH:MM:SS is in military time format.
policy	The name of the role currently assigned to the identified station.

Table 14: Device Attributes Returned from a Successful Query (continued)

XML Element	Meaning
topology	If the default action of a role contains traffic to a VLAN, this will be the name of the topology defined on the controller for that VLAN.
ingress_rc (deprecated)	This is a deprecated attribute and will be removed in the future. The rate limits applied to the station can be determined by examining the role assigned to the user as named in the "policy" element.
egress_rc (deprecated)	This is a deprecated attribute and will be removed in the future. The rate limits applied to the station can be determined by examining the role assigned to the user as named in the "policy" element.

Service Number 6

Request Parameters:

1 MAC address of the device that is the subject of the query.

Table 15: Primary Return Codes

Return Code	Meaning
1	A client element is present in the response document. It may or may not contain the attributes of the requested device.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=6&value=00:26:B9:DE:27:CB
```

or

```
https://192.168.18.7:33333/event.php?type=6&value=00:26:B9:DE:27:CB
```

The above samples attempt to request from the controller a description of the device with the MAC address 00:26:B9:DE:27:CB.

Sample Response (No encryption)

```
Response to a Request with Invalid Query Parameters
<?xml version="1.0"?>
<response>
    <status>8</status>
</response>
```

Response to a Request with Valid Query Parameters – Subject Device Not Found

```
<?xml version="1.0"?>
<response>
    <client>
        Not Found
```

```
    </client>
    <status>1</status>
</response>
```

Response to a Request with Valid Query Parameters - Found Subject Device. User does not have a known identifier yet

```
<?xml version="1.0"?>
<response>
    <client>
        <vns_id>2</vns_id>
        <AP_serial>0500008162150300</AP_serial>
        <ssid>TheMall-Filtered</ssid>
        <ip_addr>11.11.11.252</ip_addr>
        <mac_addr>00:13:02:D0:F5:4E</mac_addr>
        <user></user>
        <client_status>Not validated</client_status>
        <session_start>10 June 2013 15:00:03</session_start>
        <policy>Unregistered</policy>
        <topology>VLAN_10</topology>
        <ingress_rc>N/A</ingress_rc>
        <egress_rc>N/A</egress_rc>
    </client>
    <status>
      1
</status>
</response>
```

The XML document has one of the following main forms:

- It may just contain the status element, as shown in Response from Event.php on page 52.This will happen if the list of values has too many or too few parameters in it. The return code in the document will be 8.
- It may contain a “client” element and a status element but the client element only contains the text “Not Found.” This happens when the controller cannot locate a session for the specified MAC address. Response to a Request with Valid Query Parameters - Found Subject Device. User does not have a known identifier yet on page 66 shows a sample of the type of document that will be returned in this case.
- The response can contain client and status elements and the client element contains elements describing the session of the identified user. Table 14: Device Attributes Returned from a Successful Query on page 64 lists the elements that will be returned by a successful lookup.

Note

If the input MAC address is valid but the controller can’t locate a session for it, the status code will be ‘1’. This usually indicates success, but in the case of this event.php service and the next one, it simply means that a “client” element is included in the document. The receiver must check the contents of the client element to tell if the request succeeded.

Retrieve Information about a User’s Session Identified by Device IP Address

This request collects information about the session of the device with the IP address specified in the request’s value parameter. If the request is successful the response takes the form of an XML document

containing the attributes of the identified device. The response to an unsuccessful request contains a status code or a string indicating why the query failed.

This request and the possible responses to it are almost identical to those described in Retrieve Information about a User's Session Identified by Device MAC Address on page 64.

Service Number 12

Request Parameters:

1 IP address of the device that is the subject of the query.

Table 16: Primary Return Codes

Return Code	Meaning
1	A client element is present in the response document. It may or may not contain the attributes of the requested device.
8	Too many or too few arguments have been passed in the value parameter. In this case, no arguments were passed or two or more were passed.

Sample Request (No encryption)

```
http://192.168.18.7:33333/event.php?type=12&value=10.10.10.15
```

or

```
https://192.168.18.7:33333/event.php?type=12&value=10.10.10.15
```

The above samples attempt to request from the controller a description of the device with the IP address 10.10.10.15.

Sample Response (No encryption)

Refer to Sample Response (No encryption) on page 65 in Retrieve Information about a User's Session Identified by Device MAC Address on page 64. The three types of response described in that section apply to service type 12 as well.

Note

The services discussed in the preceding this section and Service Number 6 on page 65 are intended to assist with decision making at strategic times during a user's session. For example, either query might be used by a help desk tool to collect information about the session of a user that is phoning for help.

These queries are not intended to be used to collect information about all currently associated users on a periodic basis. The overhead of collecting and reporting this information through this interface is too great.

5 Facility Reference—Firewall Friendly External Captive Portal

Overview

This section describes a variation on the wireless controller's External Captive Portal (ECP) support that addresses customers who use a public cloud-based ECP. An ECP is a web server that hosts a site that allows users to authenticate to the network. The web server is not hosted on the wireless controller. Instead, the wireless controller intercepts some of the user's HTTP messages and redirects them to the External Captive Portal web server.

Typically APs and wireless controller are located together behind a firewall. Increasingly, companies are turning to cloud-based firms to provide authentication services and related analytics. These firms typically are located on the unsecured side of the customer's firewall ("outside"). These firms need a way for their staff and guests to authenticate against the cloud service without requiring extra ports be open in their firewall. The Firewall-Friendly External Captive Portal (FF-ECP), introduced in release 9.12.02, addresses this requirement for an enterprise's wireless stations.

ECP authentication involves filtering the traffic of unauthenticated stations. When the station sends HTTP traffic, its browser is redirected to a website where the station's user can authenticate. The website is referred to as an ECP because it is located outside the wireless controller (which has its own 'internal' captive portal). The external captive portal authenticates the user in whatever way it sees fit, and then tells the controller whether the user is authenticated and what policy to apply to the user's session.

It is this last step that can be problematic when the controller and the external captive portal server are on opposite sides of a firewall. The ECP support implemented in releases prior to 9.12.02 required the

ECP to send unsolicited messages to the controller to tell it the authentication state and role to assign to the station. This forced the customer to open a port in the firewall for ingress. Even more ports might be needed if the customer has more than one controller using ECP authentication.

Release 9.12.02 addresses this FF-ECP issue by allowing all interactions with the ECP to be initiated by the user. The enterprise has to allow their staff and guest to egress through port 80 on the firewall to use the cloud-based ECP at all. This satisfies the pre-conditions for deploying the controller's FF-ECP solution.

The rest of this document describes how to configure and use the controller's FF-ECP support. The next section describes the message sequence that occurs when a station authenticates via FF-ECP support. The section highlights support for two different authentication flows:

- A simplified flow in which the controller accepts instructions from the ECP relayed through the station's web browser.
- A more complex flow in which the controller invokes RADIUS authentication to confirm the apparent authentication status of the station.

The following section describes how to configure FF-ECP support on the controller for the two FF-ECP variants. The final section covers how to program the actual external captive portal so that it can interact with the controller in these two FF-ECP authentication scenarios.

It is worth noting that the controller's ECP support before release 9.12.02 remains available and is a good choice for administrators whose ECP is on the same side of their firewall as their controller.

Firewall Friendly External Captive Portal Flow of Events

Figure 12: Firewall Friendly ECP Event Flow on page 70 illustrates the main flow of events that occurs when a wireless station authenticates against an ECP using the controller's FF-ECP feature. The main participants in the scenario are:

- The station being authenticated ('user').
- The Controller that manages the AP the user is communicating through.
- The Firewall between the user and controller on one side and the ECP on the other.
- The ECP that performs the actual authentication.

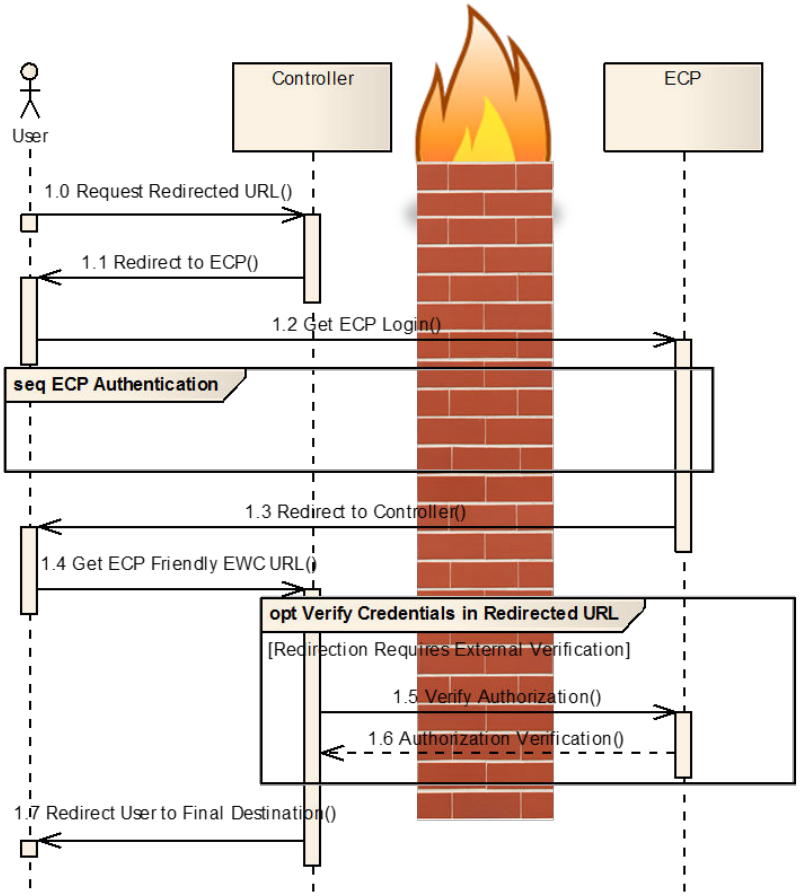


Figure 12: Firewall Friendly ECP Event Flow

FF-ECP Flow

When the user sends HTTP traffic (1.0 in Figure 12: Firewall Friendly ECP Event Flow on page 70) the controller spoofs the destination web server. It tells the station's browser that the resource it is requesting has temporarily been moved to another server (1.1 in Figure 12: Firewall Friendly ECP Event Flow on page 70). The other server is in fact the ECP. The controller can add parameters to the redirection, such as the user's MAC address and the BSSID that the station is associated with.

The station's browser normally follows the redirect automatically (1.2 in Figure 12: Firewall Friendly ECP Event Flow on page 70). The redirect contains the query parameters added by the controller. Because the ECP is located in the cloud, the user's request must be forwarded through his enterprise's firewall. Most companies allow requests for port 80 to pass through the firewall.

Typically the firewall serves also serves as a NAT. It records some state for the connection, replaces the IP address in the request, and forwards it to the ECP.

When the ECP receives the redirected request, it typically replies with a web page. The station's browser sends subsequent requests to the ECP to retrieve additional content needed to render the page. So

long as the NAT considers the connection alive, it will correctly forward responses from the ECP back to the client.

The ECP is free to authenticate the station in whatever way it chooses. The controller is not involved in this interaction, except to forward traffic between the ECP and the station. The interaction can be as simple or complex as necessary (represented by the box labeled “seq ECP Authentication” in Figure 12: Firewall Friendly ECP Event Flow on page 70). The ECP may simply display a set of terms and conditions that must be accepted before the user is assigned to a more liberal access control role.

At some point the ECP will decide that it wants to change the station’s authentication state and role. The ECP must wait until the station submits a get request to it. Then the ECP can reply by redirecting the station’s browser to a URI served by the controller (1.3 in Figure 12: Firewall Friendly ECP Event Flow on page 70). This URI can contain parameters, such as identification for the station, the name of a role to apply, or a VLAN ID to assign to the station. Because the ECP is responding to a request from the browser, the firewall should have no problem forwarding the response to the correct destination on the secure side of the firewall.

The station’s browser usually follows the redirection URI automatically (1.4 in Figure 12: Firewall Friendly ECP Event Flow on page 70). Assuming the URI passes basic validation, the flow proceeds in one of two possible ways, depending on the content of the URI. If the URI contains a signature (secure hash) and the hash is verified by the controller, the controller will accept the user as authenticated. If the URI contains the name of an access control role defined on the controller, it will apply that role to all traffic the station sends subsequently.

If the URI is unsigned and contains a user name and password, then the controller will attempt to authenticate the user against a RADIUS server. The WLAN Service that redirects to the ECP must have at least one RADIUS server configured for authentication or an error will be reported. This is shown as arrows labeled 1.5 and 1.6 in Figure 12: Firewall Friendly ECP Event Flow on page 70. The administrator must configure the controller with the address and the shared secret of at least one RADIUS authentication server.

The RADIUS server can reject the station’s access request or grant it. The response from the RADIUS server may also contain attributes, such as maximum session duration, the VLAN to which the station’s traffic should be assigned by default, and the name of an access control role to apply to the traffic the station sends subsequently. If the attributes in the response are valid, the controller applies them to the user’s session.

Once the user is deemed authenticated, it is typically assigned to a new role that does not redirect its HTTP traffic to the ECP. Some of the station’s HTTP traffic may be blocked by the role, but it will not be redirected. Because this is a function of the role the station gets assigned to, it is up to the administrator of the controller to define the authenticated role appropriately.

Because the station arrived at the controller as the result of a redirect, it is necessary for the controller to send the station an HTTP response. If the redirect from the ECP back to the controller contains a URL, the controller can be configured to redirect the station’s browser to it. The controller can be configured with a specific URL that all authenticated FF-ECP users get redirected to. The controller can be configured to redirect stations to a page served by the controller which allows them to manage their session. If an error occurs or the ECP or RADIUS server rejected the station, it will be redirected to an error page.

Assuming the station is authenticated at this point it is free to surf the web to the extent allowed by the authenticated role to which it is assigned.

Configuring FF-ECP

Configuring the Controller to Redirect to the External Captive Portal

The controller must be configured to redirect to the ECP the HTTP traffic of unauthenticated stations. HTTP traffic is redirected when:

- It is blocked by a rule or default action in an access control role, and
- The user that sent the traffic is in an unauthenticated state, and
- The traffic is carried on a tunneled (Bridged at Controller, Routed) topology, and
- Some type of captive portal authentication has been configured for the user's WLAN Service, including a destination for redirected traffic.

The controller needs to be configured to redirect user HTTP traffic and it needs to be told where to redirect that traffic as well as how to listen for instructions sent to it by the External Captive Portal web server. Policy is used to cause the controller to redirect unauthenticated HTTP traffic. Controller WLAN Service configuration determines where the redirected traffic goes and how communication between the controller and the External Captive Portal takes place.

Causing HTTP Traffic to be Redirected using Roles

To use HTTP redirection an access control role must be defined on the controller with the following characteristics:

- The role allows the station to use DHCP, DNS and ARP.
- The role allows the station to communicate with the external captive portal server using HTTP or HTTPS.
- For FF-ECP, the role must allow the station to send traffic to the controller's IP address on the VLAN containing the station's traffic.
- The role blocks at least some HTTP traffic, although not HTTP traffic destined for the External Captive Portal or the port on the controller to which the ECP will redirect authenticated stations.

The role must permit the station to send and receive DHCP, DNS and ARP traffic so that the station can function on an IPv4 network.

HTTP traffic to the external captive portal and to the interface on the controller receiving FF-ECP redirects must not be blocked by the role. If it is blocked, every HTTP request to that destination will trigger the controller to send a redirect back to the station. Because that redirect is back to the blocked ECP or controller IP address, an infinite loop could be created. Obviously, this must be avoided.

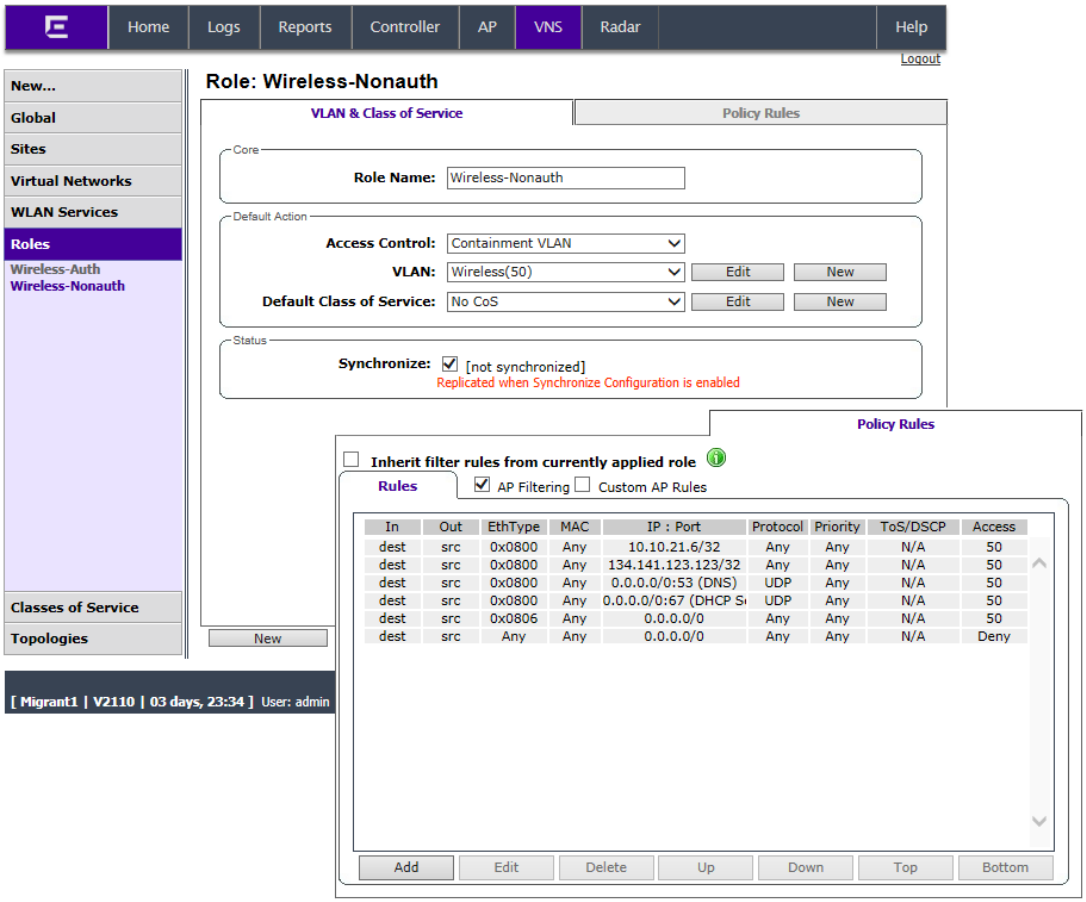


Figure 13: Sample Default Non-Authenticated Role that Redirects HTTP

In release 9.15, if the WLAN Service is configured for FF-ECP authentication and the station is unauthenticated then the controller will redirect any HTTP traffic denied by the station's role to the ECP.

The simplest way to assign the role to all unauthenticated stations is to create a VNS from the WLAN Service that uses FF-ECP and use a role that meets the previous requirements as the default non-authenticated role for the VNS. Figure 14: Sample VNS on page 74 shows the configuration of a sample VNSusing the "Wireless Nonauth" shown in the preceding figure as the role assigned by default to all associated unauthenticated stations. The VNS configured in Figure 14: Sample VNS on page 74 is suited for use with FF-ECP. The *ExtremeWireless User Guide* explains in more detail how to create a VNS using the controller GUI.

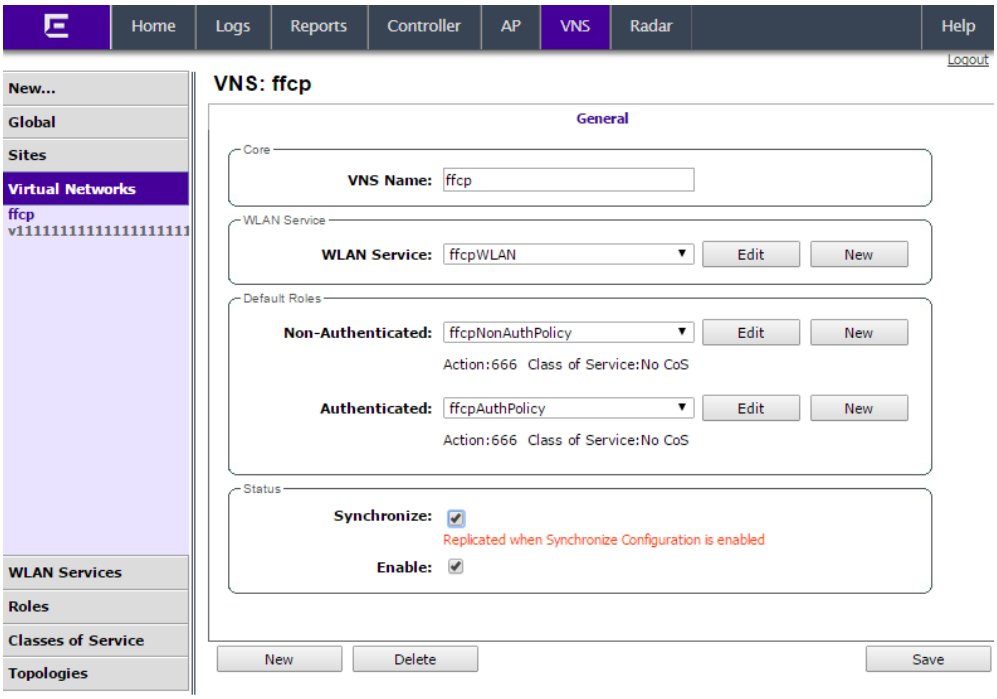


Figure 14: Sample VNS

Configuring a WLAN Service for Firewall Friendly External Captive Portal

Configuring FF-ECP authentication is similar to configuring ECP authentication. The first step is to create a new WLAN Service with the Service Type set to “Standard”. Help on creating the WLAN Service and on creating a VNS from the WLAN Service is available in the Extreme Wireless User Guide [1].

1 Once the WLAN Service exists, the FF-ECP configuration takes place on the Auth & Acct tab. First, select Firewall Friendly External as shown in in the following figure.

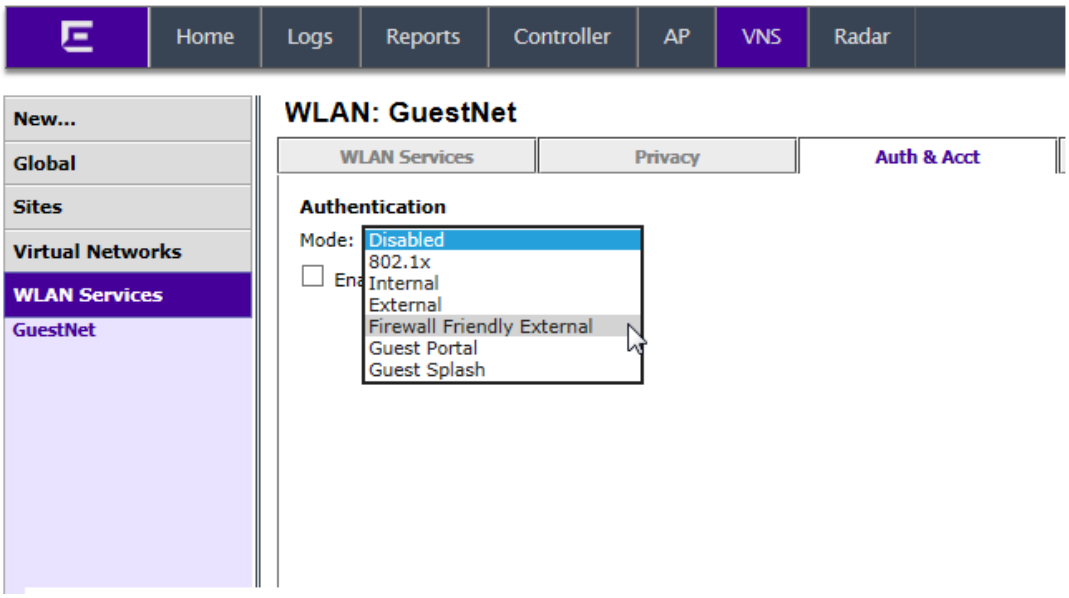


Figure 15: Selecting the "Firewall Friendly External" Support Option

2 If the controller will perform RADIUS authentication after receiving the web redirect from the ECP, configure the RADIUS server that will be used. A RADIUS server is required if the FF-ECP does not include all the following items in the redirection URLs it sends to authenticated clients:

- Signature – A secure hash computed over the body of the redirection URL.
- Timestamp – A timestamp indicates when the redirection URL was sent to the user.

If the above items are provided, the redirected station is considered fully authorized so RADIUS authentication will not be performed, even if the WLAN Service is configured with a RADIUS authentication server.

Up to three RADIUS servers can be selected for authentication and up to three servers can be selected for accounting. The *ExtremeWireless User Guide* provides instructions on how to configure RADIUS server details on the controller.

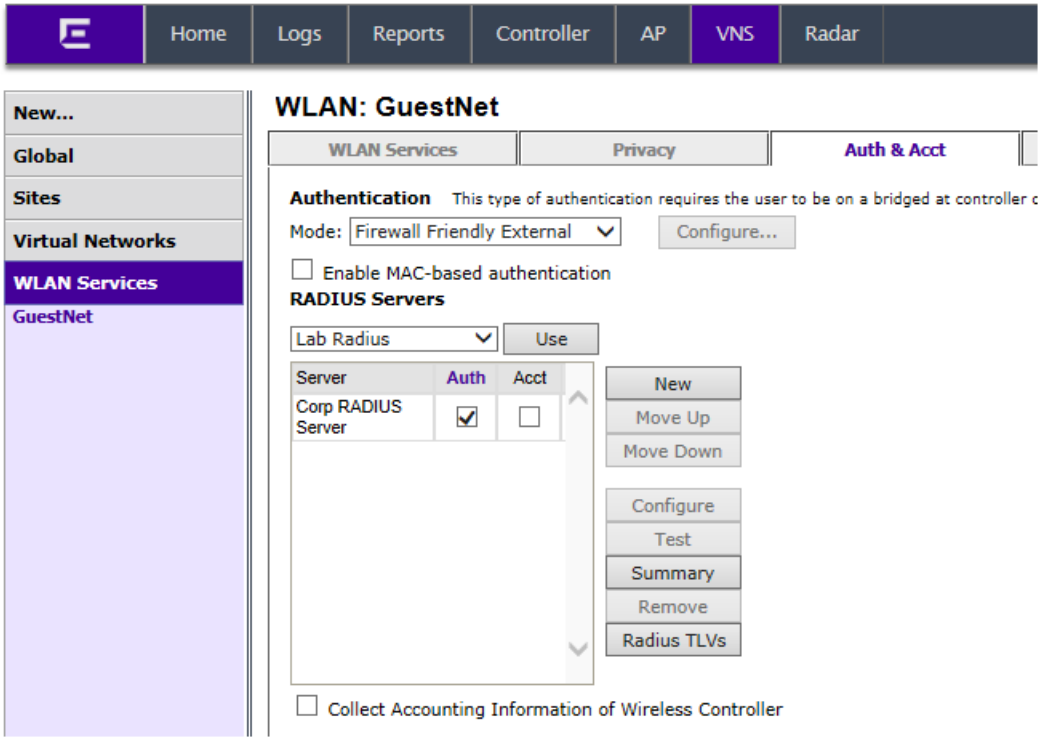


Figure 16: Configuring the RADIUS server to use for authentication

3 Save the WLAN Service. If the save is successful, the Configure... button will be enabled.

4 Configure the captive portal related addresses and the options that will be sent to the ECP. Start by clicking the Configure button".

A dialog similar to the one shown in Figure 17: Configure FF-ECP Options on page 77 should appear.

When the dialog is first opened for a newly created WLAN Servic,e the only option filled in or selected is Use HTTPS for User Connections. The example in Figure 17: Configure FF-ECP Options on page 77 has already been filled in by an administrator.

FF-ECP Options

Identity and Shared Secret

The identity and shared secret are optional, but they must be provided if the ECP will sign its redirection responses and wants the controller to sign its redirection responses.

The identity must be a printable (non-control code) ASCII alphanumeric string. The shared secret should:

- Be a printable (non-control code) ASCII string.
- Be between 16 and 64 characters. It can contain slashes, braces, and other printable ASCII symbols.
- Be exactly the same key must be configured on the ECP and associated with the same identity.

The identity and shared secret fields play a crucial role in signature generation. The identity field must be included in any signed redirected web request that redirects a user from ECP to the controller or vice versa. The identity tells the receiver which shared secret to use to validate the message signature.

The identity field and shared secret work together as a pair. Different WLAN Services on one controller can have the same identity configured, in which case the WLAN Services must have the same shared secret.

How the identity and shared secret are used to sign a redirection query string is covered in Creating the Signature and Verifying the Request on page 97.

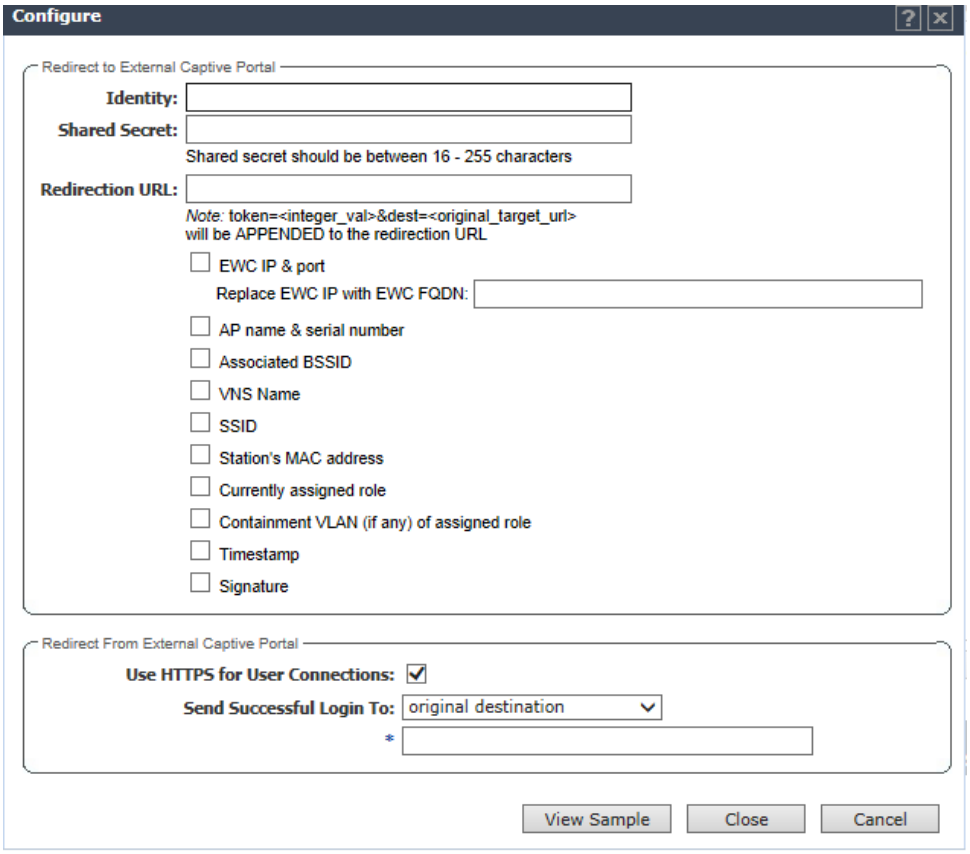


Figure 17: Configure FF-ECP Options

Redirection URL

The redirection URL is the URL that stations will have their HTTP traffic redirected to. This is a mandatory field. It should point to a page, script or program served by the ECP. Both HTTP and HTTPS are supported. The maximum length of the URL is 256 octets.

The redirection URL will have some parameters appended to it when it is received by the ECP. All information communicated with the ECP will be part of the redirection URL query string.

[8] Cookies and headers are not guaranteed to be forwarded by redirected browsers to the redirection destination. So the query string is the only option that can be used reliably to transfer data between the controller and the ECP via browser redirects.

At a minimum, the URL includes the following:

- A token—An identifier for the user-session. It must be included as a query string parameter in all ECP redirections back to the controller.
- A WLAN identifier—Helps the controller determine how to process the redirect back from the ECP. It must be included in the redirection sent by the ECP to the controller's web server.
- The original URL that the station was trying to access when it was redirected to the ECP, which may or may not be important. If the administrator wants all authenticated users to end the login sequence on one specific page (such as the site's news page) then the original URL can be ignored by the ECP. If the administrator wants the user to be redirected to a session management page after a successful login or if he wants the user to be sent to the original URL then the ECP must save the original URL and include it in the redirection that causes the user's browser to return back to the controller.

Optional Attributes the Controller can send to the ECP via Browser Redirection

EWC IP and Port

The controller can include its IP address and port in the redirection URL. This is optional, but necessary if the ECP interacts with more than one controller. Without these fields the ECP may not be able to compose the correct redirection URL to cause a station to complete authentication on the correct controller. The ECP should store the controller address and port with the token and other session details so that it is available throughout the authentication process. Currently only IPv4 addresses are used.

Unlike standard ECP, the administrator does not directly configure the controller IP address and port to which the ECP should redirect stations. Standard ECP makes use of a separate connection that is used by the ECP to send session control messages to the controller. FF-ECP does not have this connection because control messages are relayed from the ECP to the controller via redirecting the station's browser.

Because the station's browser conveys the commands to the controller, the station must have easy access to the controller interface and port that will receive the redirection. This will be the controller's address on the topology (VLAN) to which the user is assigned. The address to use is only known once a VNS based on the WLAN Service is fully configured.

The address will be difficult to learn in advance, and may change as a result of configuration, so it is useful to enable this option, even if the ECP interacts with a single controller.

Replace EWC IP with EWC FQDN

Sometimes it is convenient for the ECP to be able to redirect stations back to an FQDN belonging to a controller. This is necessary to avoid certificate warnings when the certificate contains FQDNs for identity instead of IP addresses. The EWC IP and Port option normally adds the IP address and port on the controller that the ECP should redirect clients to. If EWC IP and Port is enabled and the FQDN field is populated with a valid FQDN the controller will put the provided FQDN and port into the redirection it sends to the station.

This option only takes effect when the **EWC IP and Port** option is enabled.

AP Name and Serial Number

The AP Name is the name administratively assigned to the AP to which the authenticating station is associated. The serial number is the manufacturing serial number of the AP. If the administrator does not assign a name to the AP, the name defaults to the serial number.

The AP Name and Serial Number can be useful if the ECP needs to behave differently depending on the location of the station being authenticated. For example, the ECP might need to present the login page in different languages depending on the country the AP and station are located in. The ECP or its authentication server can make location-based decisions, such as when rendering the login page or even when authenticating the user.

The AP Name attribute will have the same value as the Siemens VSA called “Siemens-AP-Name”. The AP Serial Number attribute will have the same value as the Siemens VSA called “Siemens-AP-Serial”.

Associated BSSID

This is the Basic Service Set Identifier (BSSID) to which the station being authenticated has associated. It is a MAC address belonging to the AP to which the station has associated. The BSSID is the same identifier that the controller puts in the Called-Station-ID RADIUS TLV sent in Access-Requests and RADIUS accounting messages.

The associated BSSID will be sent in the form of a 12-character ASCII-encoded lowercase hex string. The string “00112233aabb” is an example of a BSSID formatted for the associated BSSID parameter.

VNS Name

This is the name of the VNS to which the “Associated BSSID” belongs. A VNS is a combination of a WLAN Service and one or two default roles. The value in the VNS Name attribute is the same as the value in the Siemens RADIUS VSA called “Siemens-VNS-Name”.

SSID

This is the Service Set Identifier (SSID) to which the station associated. While the BSSID identifies which interface of the AP the station is using the SSID identifies the overall service being used. The SSID typically is a human readable word, like “FreeWiFi”.

Station’s MAC Address

The station’s MAC address uniquely and globally identifies the station. More specifically, this is the MAC address of the station’s wireless interface that associated to the BSSID. This holds the same value that the controller puts in the Calling-Station-Id RADIUS attribute. The station MAC address will be sent in the form of a 12-character ASCII-encoded lowercase hex string. The string “00112233aabb” is an example of a MAC address formatted for the station MAC address parameter.

[9] Refer to Siemens VSA Dictionary in FreeRadius Format on page 156.

Currently Assigned Role

The “Currently Assigned Role” attribute contains the name of the access control role assigned to the station at the time its browser was redirected to the ECP. The contents of this attribute match the contents of the Siemens RADIUS VSA called “Siemens-Policy-Name”.

Containment VLAN (If Any) of Assigned Role

If the default action of the “Currently Assigned Role” is “Contain to VLAN,” then this contains the name of the topology/VLAN to which the station’s traffic is contained by default. Roles need not have a “Contain to VLAN” default action, in which case this attribute would be empty, even if requested.

Timestamp

This attribute contains the time on the controller at which the HTTP request that was redirected to the ECP was received. The timestamp is in UTC, in the form “seconds elapsed since 1970-01-01 00:00:00”. The timestamp will be included if the controller is configured to sign the redirection to the ECP, even if it was not explicitly requested. The timestamp is useful for preventing replay attacks of recorded redirected requests.

Signature

If this item is configured, the controller will compute a secure hash over select portions of the redirection response it sends. The request also will include a timestamp even if none was specifically requested.

A shared key and identity must be entered if a signature is requested. The identity will be included in the redirect to the ECP. The ECP can use this to look up the appropriate shared secret in order to validate the signature.

Selecting the timestamp and signature options does not cause the controller to expect signatures on the redirects from the ECP. However, if a WLAN Service that uses FF-ECP is not configured with at least one RADIUS authentication server, then the redirections from the FF-ECP to the controller must be signed and time stamped. The details of computing the signature are covered in Signing the Redirection to the Controller on page 104.

Note

Browsers do not necessarily accept redirects in response to posts and do not necessarily send to the redirection destination HTTP headers received with the redirection URL. This means that any data that has to be communicated between the controller and the ECP via redirects must be sent as query parameters.

The HTTP-related standards do not specify the maximum length of the query string. Most browsers can handle a query string of about 2000 characters but older browsers may not be able to deal with redirections that contain query strings that are 2000 characters long.

Use HTTPS for User Connections

This setting controls whether the controller listens for redirects at port 443 or port 80 and whether it expects to receive redirects from the ECP as HTTPS or HTTP.

The default is to use HTTPS as this is the most secure option. The controller has a self-signed certificate that it will use by default for HTTPS. However, most browsers will immediately warn the user that they are being redirected to a site with a self-signed certificate. If this service is to be used by a large number of users or by casual users, it is best to obtain a certificate from a CA that is trusted by all browser vendors. An administrator running multiple controllers with FF-ECP can obtain a wild card certificate that covers all the interfaces of all the controllers.

If a third-party certificate is installed, it must be installed on the topology that stations have direct access to (typically either the WLAN Service's default topology/VLAN or the station's role's default action's containment VLAN).

The *ExtremeWireless User Guide* describes how to install a certificate on the controller.

Send Successful Login To

Because the ECP redirected the station to an HTTP/HTTPS URL on the controller, the station's browser expects to receive something from the controller. If it does not, the browser will time out and display a message stating that the page in question could not be displayed. For a better user experience, the controller must send the user's browser to an appropriate web page, regardless of whether he passed or failed authentication.

If a station fails authentication (really only a possibility when RADIUS authentication is used as part of FF-ECP), the station's browser is directed to an error page. This page is not configurable at this time.

If the station passes authentication, it is redirected to one of these administrator-selected destinations:

- The original URL the browser was trying to receive when it was redirected to the ECP.
- A specific URL to which all successfully authenticated stations are sent. This could be a news page, or a sponsor's page or some other page that is useful to authenticated stations.
- A session management page. This page contains controls to tell how long the user's session has lasted and to gracefully terminate the user's session. It can also contain a link to the original URL the browser was trying to receive when it was redirected.

As mentioned earlier, the "original URL" option will only work if the ECP sends to the controller the original URL ("dest" parameter) that the controller sent to the ECP in the initial redirect. If the WLAN Service is configured to send successful logins to the "original destination" and the ECP does not return the original destination, then the station will be redirected to an error page. If the ECP does not return the original destination URL in the redirect to the controller and the WLAN Service is configured to

redirect the station to “Captive Portal Session Page,” the user will be redirected to that page, but it will not contain a link to the original destination.

Note

View Sample

The FF-ECP **Configure** dialog box includes a **View Sample** button. When pressed, the button opens a web page that contains an example of the format of the redirection URL the controller expects to receive (indirectly) from the ECP.

If the WLAN Service is part of a VNS or has a default topology, the server portion of the URL will contain the IP address to which the ECP should redirect the station’s browser. The query string will be populated with realistic but made-up data. This can be useful as a reminder when developing the ECP program.

Finishing Up On the Controller

Once the appropriate options have been selected on the **Configure** dialog box, click **Close**. This returns you to the WLAN Service **Auth & Acct** tab. The changes made in the **Configure** dialog box have not been saved. This is indicated by the message “* settings modified” that appears next to the captive portal configuration button. You must click **Save** on the **Auth & Acct** tab for the changes to take effect.

The figure below shows the WLAN Service’s **Auth & Acct** tab after changes were made in the FF-ECP configuration dialog box.

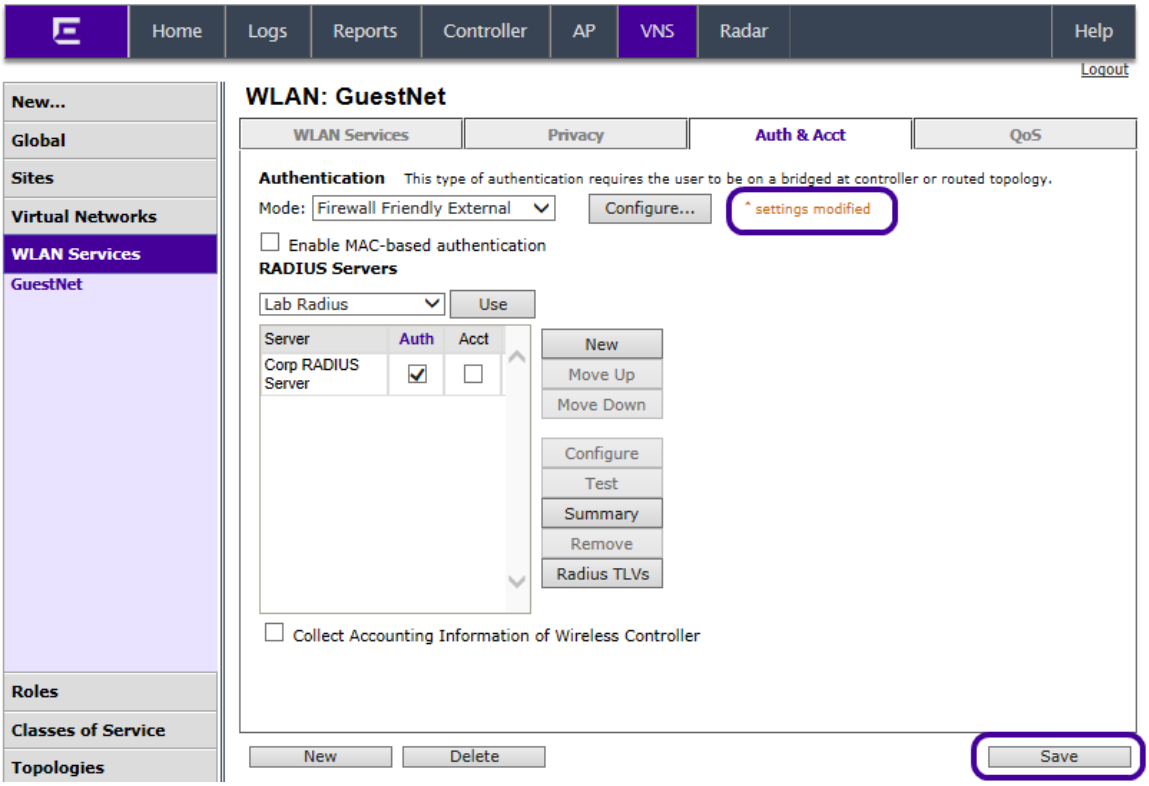


Figure 18: Saving the WLAN Service

Configuring the Firewall

The firewall must be configured to allow stations behind the firewall to forward traffic to port 80 destination on the insecure side of the firewall. Most sites configure this behavior by default. Firewall Friendly ECP can require the firewall to allow the controller to forward RADIUS requests (UDP) to an external server (typically at port 1812).

Configuring the ECP

The ECP essentially is a web server that runs an application allowing stations to change their authentication state, by providing credentials, credit card details, demographic information about themselves or acknowledging terms and conditions. The application can be written in any language the ECP provider chooses. The controller's web applications are implemented in PHP but they will interact with any programming language or library on the ECP or client that can generate valid HTTP.

If the ECP expects the controller to sign redirection responses, it is critical that the real time clocks on the controllers and ECP are synchronized. Signed redirection responses include timestamps to protect against replay attacks. The redirection responses should be trusted only for a limited period of time.

The easiest way to do this is to configure both the controller and the ECP to use NTP to manage the clock. The time zone also needs to be set correctly, both on the ECP and the controller. The *ExtremeWireless User Guide* describes how to configure the controller to use NTP.

The timestamps in signed redirection responses are in UTC ("Coordinated Universal Time"), so there is no need for the controller to know the ECP's time zone and no need for the ECP to know the controller's.

The signing algorithm is a slight variation on Amazon Web Service's (AWS) algorithm for signing requests using query string parameters. At this time AWS makes an SDK available that includes implementations of the signing algorithms in several different languages (notably Java and PHP). It may be helpful to obtain and use this SDK rather than re-implement the signing algorithm from scratch.

Processing Performed by the External Captive Portal

As discussed in FF-ECP Flow on page 70 the ECP must receive HTTP/HTTPS redirections from the wireless controller, provide some means for a station to become authorized and finally redirect the user back to a web server on the controller. Each of these steps is explored in more detail in this section.

Receiving the Redirected Request at the External Captive Portal

The script on the ECP that receives redirected requests has two responsibilities:

- Parse the redirection URL and preserve critical parameters for future use.
- Compose the web page that the user fills in to log into the network.

Parsing the Redirection URL Sent from the Controller

The request for the login page will take the form of an HTTP/HTTPS get request. All the arguments to the request are passed as query strings appended to the URL. Typically the web server or the back-end

runtime system will have parsed the query strings and made them available in a more convenient form for the back-end scripts.

The format of the redirected URL depends on the options con-figured on the External Captive Portal configuration page. The parameters are described in the table below.

Table 17: Parameters Available on the Redirection URL from the Controller to the ECP

Parameter Name	Parameter Value	Mandatory	Notes
ap		No	The configured friendly name of the AP to which the authenticating user has associated. The attribute is included in the redirection response from the controller only when the controller is configured to include it, in which case it must be present.
bssid		Alphanumeric String	The BSSID to which the authenticating station has associated. The BSSID is a MAC address belonging to the AP to which the station associated. The BSSID is in the format of six hex digits. The hex digits are “0123456789abcdef”. An example BSSID could be “00026fe9b568”. This is the same value that would be included in the Called-Station-ID field of a RADIUS Access-Request sent on behalf of this station. The attribute is included in the redirection response from the controller only when the controller is configured to include it, in which case it must be present.
ssid	A character string up to 32 bytes long	ASCII-encoded hex string	The SSID (Service Set Identifier) to which the station associated. The SSID is present only if the station has associated to a managed AP.
dest	Alphanumeric string	Yes	This is the original URL that the station’s browser was trying to receive when the request was redirected. The string is URI-encoded. For example, slashes in the URL are replaced by “%2F”.
hwc_ip	Numeric String	No	This is the IP address to which stations should be redirected to complete authentication. Typically a controller ends up with many IP addresses, but only one of them will map to the WLAN service’s FF-ECP implementation. Note that this address may not be accessible directly by the ECP itself. However it will be accessible to the station that is being authenticated. This attribute appears in the redirection response from the controller (and redirected request from the browser) only when the controller is configured to include it. If the controller is configured to include it then it must be present. Although this attribute is optional it is a good idea to include it. It may be the only way for an ECP dealing with multiple controllers to know which one to redirect the station to. A sample hwc_ip address is “10.10.21.6”.

Table 17: Parameters Available on the Redirection URL from the Controller to the ECP (continued)

Parameter Name	Parameter Value	Mandatory	Notes
hwc_port	ASCII-encoded numeric string	No	This the port on the controller interface to which the station should be redirected. If FF-ECP support is configured for HTTP then the hwc_port will be “80”, otherwise it will be “443”.
mac	ASCII-encoded hex string	No	The MAC address of the station that is being authenticated. A station could have multiple MAC addresses. This MAC address is the MAC address of the station’s wireless interface that it used to associate to the wireless network. The station MAC address is in the format of six hex digits. The hex digits are “0123456789abcdef”. An example “mac” could be “0023149032a8”. This is the same value that would be included in the Calling-Station-ID field of a RADIUS Access-Request sent on behalf of this station. The attribute is included in the redirection response from the controller only when the controller is configured to include it, in which case it must be present.
role	Alphanumeric String	No	The name of the access control role to which the authenticating station is assigned at the moment of redirection. The name will be the same as the name for the role in the controller GUI and if present, in Extreme Policy Manager’s GUI. This attribute is included in the redirection response from the controller only when the controller is configured to include it, in which case it must be present.
sn	ASCII-encoded hex string	No	The serial number of the AP to which the station being authenticated associated. The serial number is assigned to the AP at manufacturing time. It uniquely and globally identifies the AP. The serial number is a sequence of hex digits with the ‘alphabetic’ characters in lower case. “12b2694560000000” is an example of an AP serial number. This attribute is included in the redirection response from the controller only when the controller is configured to include it, in which case it must be present.
token	Alphanumeric String	Yes	An identifier for the user’s wireless session hosted on the controller that performed the redirection.

Table 17: Parameters Available on the Redirection URL from the Controller to the ECP (continued)

Parameter Name	Parameter Value	Mandatory	Notes
vlan	ASCII-encoded decimal number	No	The VLAN ID of the VLAN/topology to which the station is assigned at the moment of authentication. The VLAN ID is a number in the range 1 to 4094. The VLAN ID is the containment VLAN of the default action of the role to which the authenticating station is assigned. A role's default action does not have to be "contain to VLAN". If the default action is not "contain to VLAN" then this attribute will be empty or not present. The VLAN ID appears in the redirection response (and the redirected request from the browser) only if the controller is configured to include it and the role assigned to the station has a default action that "contains to VLAN".
vns	Alphanumeric String	No	The name of the Virtual Network Service (VNS) on which the station is authenticating. A VNS abstracts wireless network services away from the hardware that implements it. The *ExtremeWireless User Guide* describes how to create a VNS. The name of the VNS will appear as it does in the controller GUI. The VNS name appears in the redirection URL only when the controller is configured to include it, in which case it must be present.
wlan	ASCII-encoded decimal string	Yes	An internal identifier for the WLAN service on which the station is authenticating. The "wlan" attribute must be present in all redirection responses (and redirected requests) sent by the controller. The ECP must return the wlan attribute in the redirection back to the controller that it sends to the authenticating station's browser.
X-Amz-Algorithm	Alphanumeric String	No	The identifier for the algorithm used to compute the "X-Amz-Signature". Only present when the controller is configured to sign the redirection. This attribute must be present when the controller is configured to sign the redirection. The value of this attribute is "AWS4-HMAC-SHA256" and is not configurable. The signing algorithm and the role of the identifier in it are covered in more detail in section Verifying the Signed Request on page 91.
X-Amz-Credential	Alphanumeric String	No	The identifier for the account whose shared secret was used to compute the "X-Amz-Signature". Only present when the controller is configured to sign the redirection. If the controller is configured to sign the redirection then this field must be present. This is covered in more detail in section Verifying the Signed Request on page 91.

Table 17: Parameters Available on the Redirection URL from the Controller to the ECP (continued)

Parameter Name	Parameter Value	Mandatory	Notes
X-Amz-Date	Alphanumeric String	No	This is the time at which the controller prepared and sent the redirection back to the user's browser. The date and time are in ASCII-encoded UTC. This attribute is present if a timestamp or a signature is requested. It can be used to identify stale or replayed URLs. If the controller is configured to sign the request this must be included in the redirection response (and the browser's redirected request).
X-Amz-Expires	Numeric String	No	This is the maximum length of time in seconds to trust the request. In other words the web request is only good until X-Amz-Date + X-Amz-Expires. After that time the URL should not be trusted as it is highly likely to have been replayed. This attribute is present only when the controller is configured to sign the redirection to the ECP, in which case it must be present.
X-Amz-Signature	ASCII-encoded hex string	No	This is the signature computed over some of the HTTP headers and parts of the query string, presented as ASCII encoded-hex. The field is present only when the controller is configured to sign the request.
X-Amz-SignedHeaders	Alphanumeric String	No	Which of the headers in the HTTP request were included in the input to the calculation of the signature. This is present only when the controller is configured sign the redirection to the ECP, in which case it must be present.

Sample Configuration and Corresponding Redirection Responses Sent by the Controller

Date, BSSID, EWC address, Station MAC without a Signature

In this example the administrator configures the controller to send a timestamp, BSSID, the controller IP & port and the authenticating station's MAC address in the redirect request. Figure 19: FF-ECP Configuration Example 1 on page 88 illustrates the FF-ECP Configuration dialog box after the administrator has entered his choices. Figure 19: FF-ECP Configuration Example 1 on page 88 contains an example of the redirection URL sent by the controller to the station to get the station's browser to connect to the ECP.

In this particular example an identity and shared secret have been configured. Since a signature was not requested the controller did not use them to create a signature.

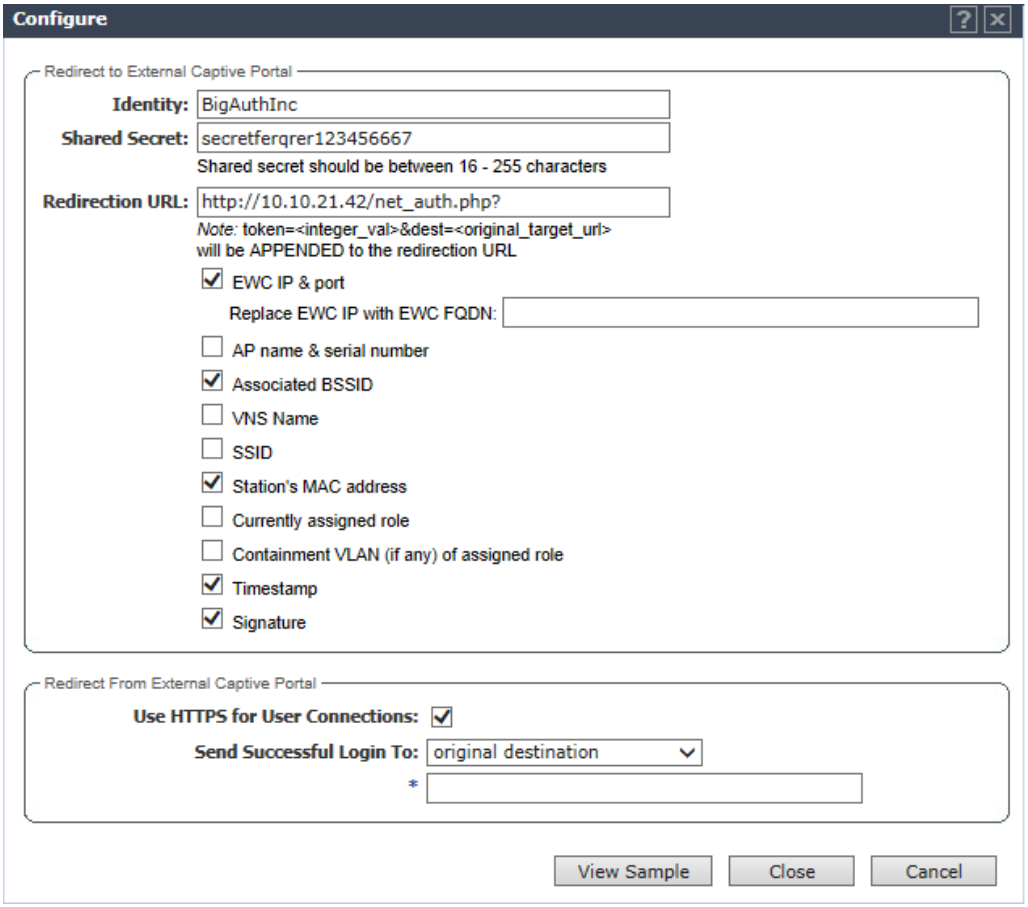


Figure 19: FF-ECP Configuration Example 1

```
http://10.10.21.42/net_auth.php?X-Amz-
Date=20140729T151045Z&bssid=00026fe9b560&dest=1.2.3.4%2Fnews.com&hwc_ip=10.10.
21.6&hwc_port=443&mac=001302d0f54e&token=T7vb1LdUZmsuY0q9V60Iww!!&wlan=1
```

Redirect Request with Parameters Requested for Example 1

As can be seen in the code snippet above, the date field is in the format YYYYMMDDThhmmssZ, where:

- YYYY is the 4 digit year.
- MM is the 2 digit month number (January is 01).
- DD is the 2 digit day of the month. The first of the month is represented as “01”.
- T – a literal that separates the date from the time.
- hh – two-digit hour, in military time format. 00 is midnight, 11 is 11 AM and 23 is 11 PM. ‘AM’ and ‘PM’ suffixes are not permitted.
- dd – two-digit hour, in military time format.
- ss – two-digit seconds field.
- Z – indicates that the time is in UTC.

This is an ISO 8601 compatible date format.

Date, BSSID, EWC address, Station MAC with a Signature

In the figure below, the administrator has configured the controller almost identically to the way it is configured in example 1, except that he has configured the controller to sign the request.

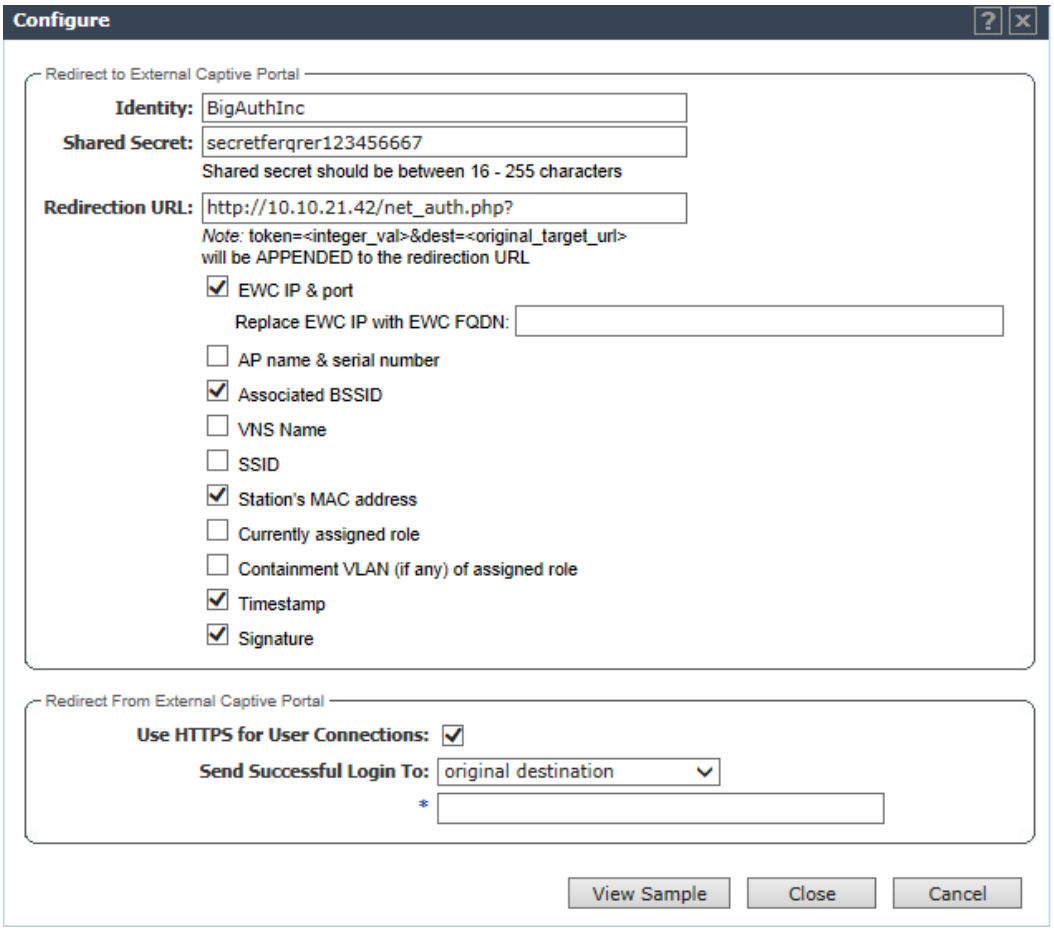


Figure 20: Configuration Example 2 – Example 1 plus a Signature

```
http://10.10.21.42/net_auth.php?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=BigAuthInc%2F20140729%2Fworld%2Fecp%2Faws4_request&X-Amz-
Date=20140729T152135Z&X-Amz-Expires=20&X-Amz-
SignedHeaders=host&bssid=00026fe9b560&dest=1.2.3.4%2Fnews.com&hwc_ip=10.10.21.
6&hwc_port=443&mac=001302d0f54e&token=T7vb1LdUZmsuY0q9V60Iww!!&wlan=1&X-Amz-
Signature=cede3b7c4638e505578e886f4d65600004249f15d7bb2d1ad7644ffd3c7617db
```

Sample URL Generated from Configuration in Example 2

The request for signatures causes a large number of signature-related fields to be added to the redirection. Verifying the Signed Request on page 91 covers how these fields work together to provide a method of authenticating the request and protecting against replays.

Include all possible data in the redirection to the ECP

In this example the administrator has configured the controller to send all possible data about the station in the redirection response to the browser.

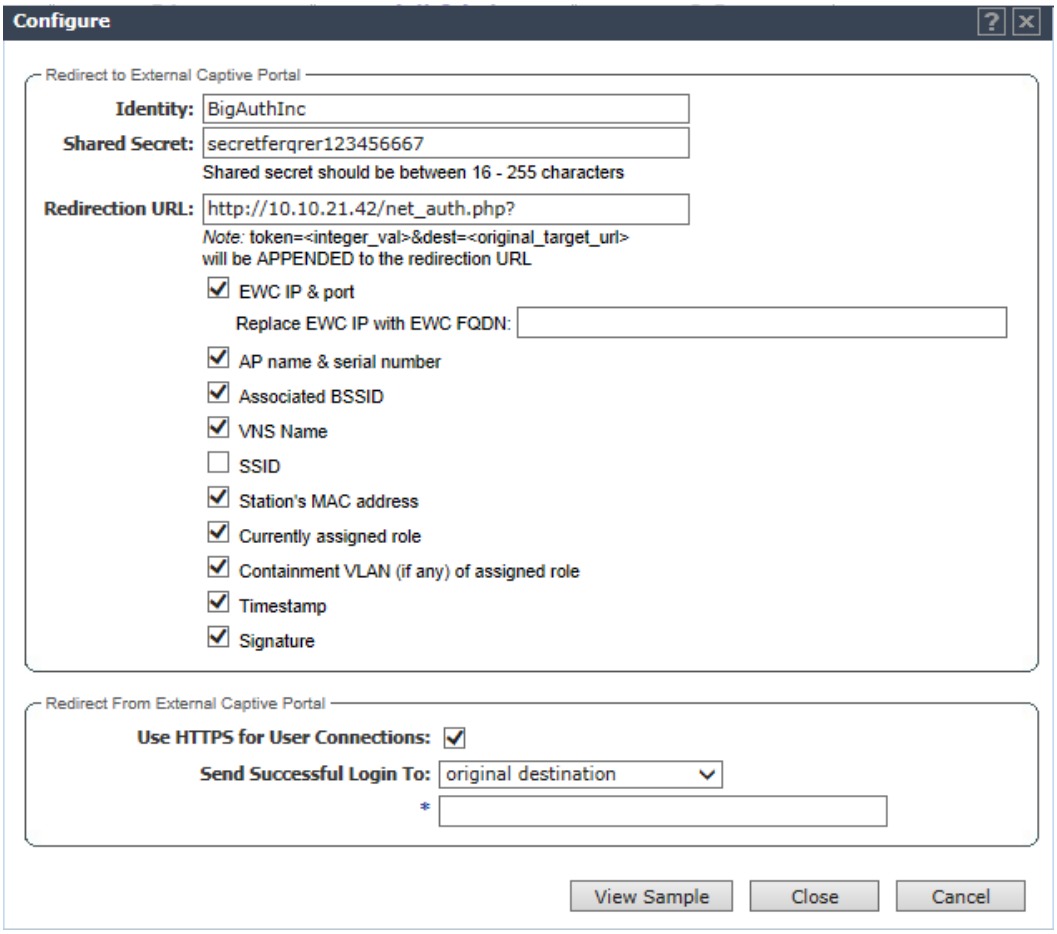


Figure 21: Configuration Example 4 – The Works

```
http://10.10.21.42/net_auth.php?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=BigAuthInc%2F20140729%2Fworld%2Fecp%2Faws4_request&X-Amz-
Date=20140729T152830Z&X-Amz-Expires=20&X-Amz-
SignedHeaders=host&ap=12b2694560000000&bssid=00026fe9b560&dest=1.2.3.4%2Fnews.
com&hwc_ip=10.10.21.6&hwc_port=443&mac=001302d0f54e&role=Wireless-
Nonauth&sn=12b2694560000000&token=T7vb1LdUZmsuY0q9V60Iww!!
&vlan=50&vns=GuestNet&wlan=1&X-Amz-
Signature=0226f763f4dcfc03f38acdec06b7a91867e2ad790f3804baacf7d6278955527d
```

Sample URL Generated from Configuration in Example 3

As can be seen in Figure 21: Configuration Example 4 – The Works on page 90, the redirection response is long. In fact, the Location header in the redirection response can reach lengths greater than 1000 characters. Most modern browsers can handle up to 2000 characters. Nonetheless it may be worth considering configuring the controller to only include the fields that are necessary for the implementation, rather than selecting everything.

[10] The station's browser builds the redirected request based on the contents of the Location header in the redirection response from the controller.

Verifying the Signed Request

As can be inferred from the sample redirection responses shown in the previous section, even if the controller is configured to include signatures, it is easy for the ECP to ignore them. The ECP simply extracts the information it is interested in from the provided attributes and ignores the rest.

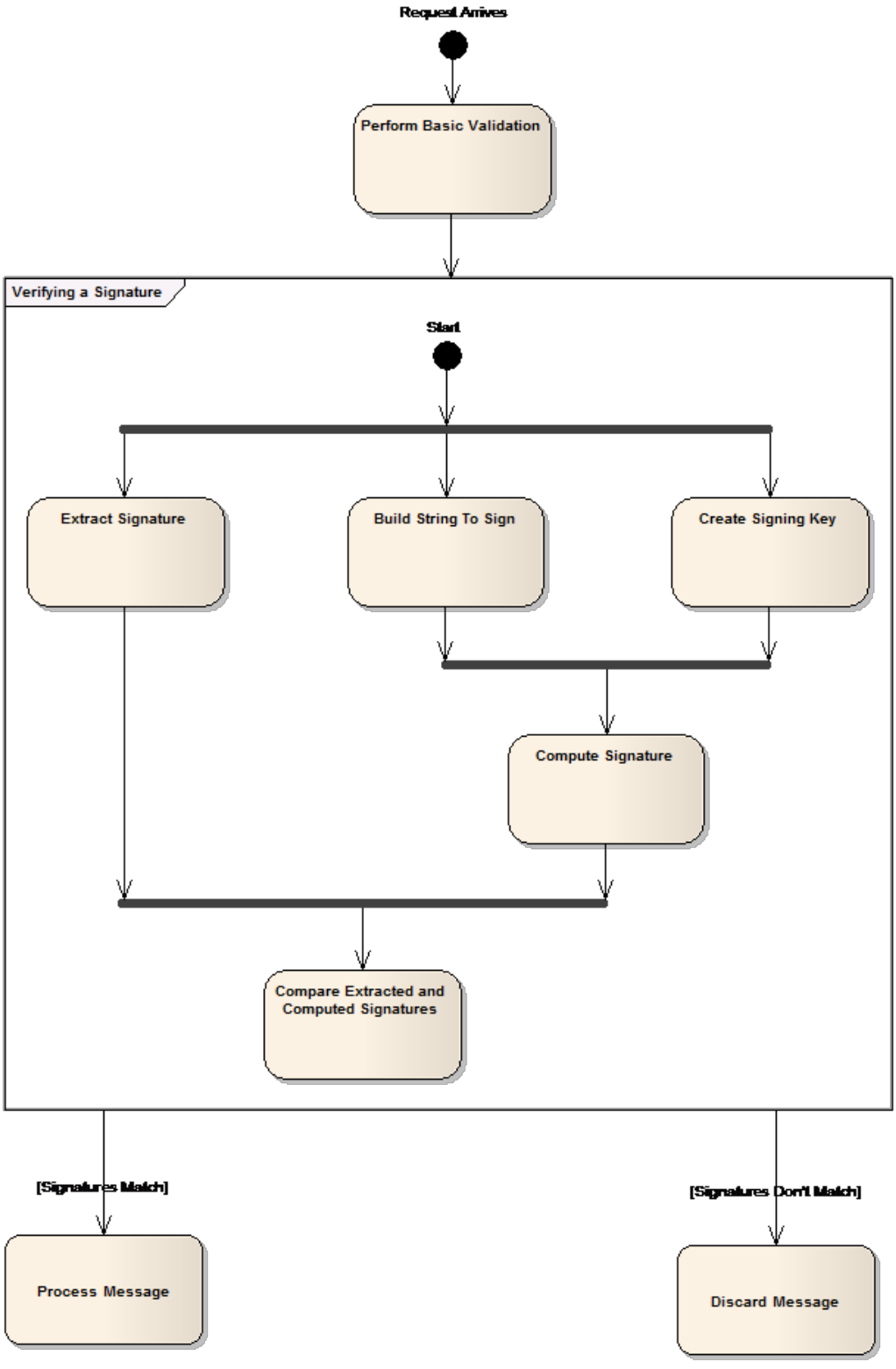
However, it is highly likely that an administrator that enables response signing wants to use the signatures to authenticate the redirected requests it receives. This section covers how to do that. The whole process is shown in Verifying a Signed Request Basic Validation Checks on page 92.

The algorithm used to sign the redirection response (and therefore the redirected request to the ECP) is based on Amazon Web Services API Signature Version 4, which is described in chapter “Authenticating Requests by Using Query Parameters (AWS Signature Version 4)” [2, p. 38]. AWS documentation refers to this approach as “Pre-signed URLs”.

Basic Steps

The basic steps for verifying the signature are:

1 Perform basic validation on the request message (are all required fields present, is the date current?). If these validations fail, there is no point in computing the signature.
2 Extract the signature from the received request.
3 From the received request, construct the string over which the signature will be computed. All but one component of this string come from the query parameters.
4 Generate the signing key. The shared secret is used to generate a signing key and is not itself the signing key.
5 Generate the signature using the signing key and the constructed string.
6 Compare the extracted signature (X-Amz-Signature) to the signature just computed. If they do not match, the request is invalid and should be discarded.



Verifying a Signed Request Basic Validation Checks

The following items can be considered when validating the redirect prior to computing the signature:

1 Does the request contain a token parameter, a WLAN parameter, and a destination URL? If not, the request either did not come from the controller or was tampered with en route.
2 If the request contains a timestamp, does the timestamp meet the following requirement:

```
timestamp <= now <= timestamp + x_amz_expires
```

Or if an allowance for clocks being out of sync is made,

```
timestamp - fuzz <= now <= timestamp + x_amz_expires
```

If not, the request is invalid, possibly the result of a user bookmarking the ECP landing page on a previous visit. The request should be rejected or discarded.

1 Are all parameters formatted in accordance with the descriptions in Table 5: Services Offered by Event.php on page 50?
2 Are all parameters required for the signature present in the request?

The first 1/3 of “verifyAwsUrlSignature” and the private method “validateQueryParms” in section crypt_aws_s4.php on page 209 provide examples of performing these types of checks in PHP.

Extracting the Signature from the Request

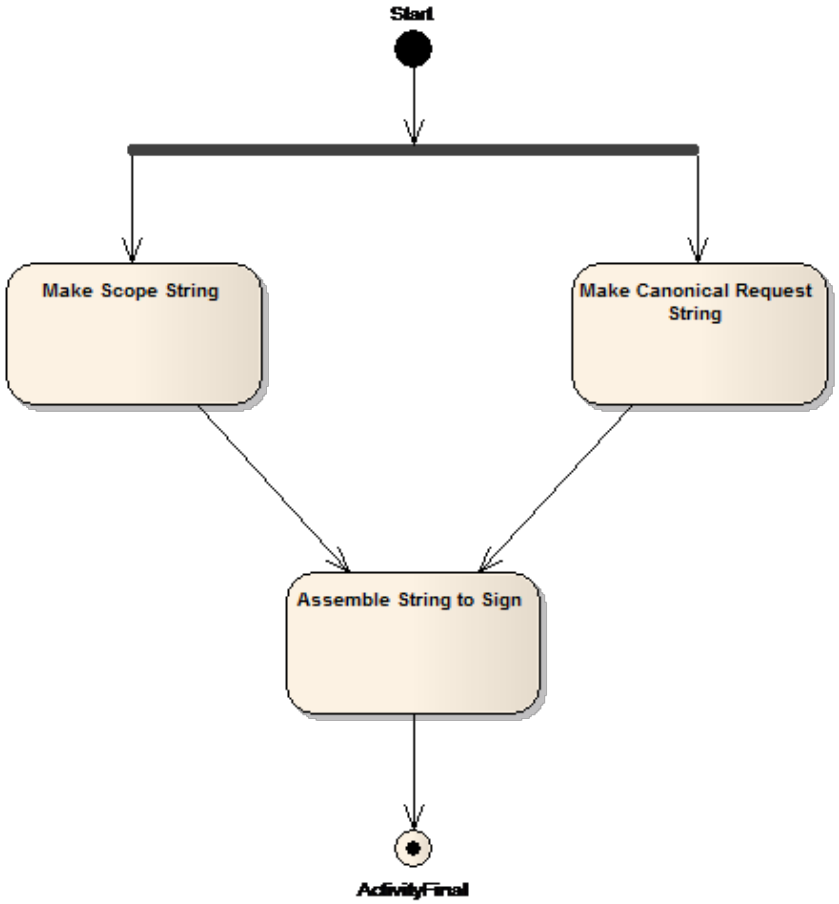
The signature is in the “X-Amz-Signature” query string parameter. Obviously the signature itself can’t be included in the computation of the signature so it must be removed from the request and set aside for later comparison. How the signature is removed from the request will depend on the program language and framework used to implement the external captive portal. The method “simpleaws::verifyAwsUrlSignature” in crypt_aws_s4.php on page 209 illustrates one way to remove the signature when the query parameters are in a PHP array.

Building the String to Sign

Figure 22: Steps in Building the String to Sign on page 94 shows the main actions required to build the string that will be signed out of the request:

1 Build the scope string.
2 Build a “canonicalized” version of the request.
3 Assemble the scope string, the canonicalized string, and some additional inputs to create the string to sign.

The scope string is easy to build out of a valid request. It is made from parts of the string in the “X-Amz-Credentials” parameter. If the credentials are valid then the scope string can be created by un-escaping the forward slashes it contains (i.e. replace ‘%2f’ with ‘/’), and then taking all the characters to the right of the first forward slash. The scope ends up being the fully qualified credential, less the identity string.



Note

Parts of the Scope

The fully qualified Amazon credential consists of:

- An identity string (the one configured in the controller GUI).
- The date portion of the X-Amz-Date.
- A region string. For a real Amazon application this is one of the geographic service regions defined by Amazon. The service region is not critical for the FF-ECP implementation so it is always set to ‘world’.
- A service identifier. The service is always set to ‘ecp’.
- The identifier ‘aws4_request’, which identifies the signature version.

For more information refer to citations [2] and [3].

The canonicalized request string has the format:

```
"GET\n"
.<URL-Path-Component>."\n"
.<URL-Query-Parameters>."\n"
.'host:'.<URL-Host>
."\n\nhost\nUNSIGNED-PAYLOAD";
```

Where:

- `GET` is the request type. For FF-ECP this will always be the literal “GET.”
- `<URL-Path-Component>` is the substring beginning with the ‘/’ at the end of the host or host-plus-port portion of the URL and either the end of the URL or the ‘?’ marking the beginning of the query parameter string. For example, the URL-Path-Component of `https://192.168.18.152:5825/adir/bdir/cdir/resource.htm?x=7&y=gg` is `/adir/bdir/cdir/resource.htm`
- `<URL-Query-Parameters>` is the substring following the ‘?’ character and extending either to the end of the URL or up to but not including the ‘#’ fragment character.
- `<URL-Host>` is the host portion of the URL string. It excludes any port number included in the URL. In the preceding URL, the URL-Host is 192.168.18.152.
- ‘`.`’ is the catenation operator.
- The remaining components are literals that should be added to the string as-is.

Finally the string that will actually be signed is composed as:

```
"AWS4-HMAC-SHA256\n"
.<Date>."\n"
.<scope>."\n"
.sha256(<canonicalized-request-string>)
```

where

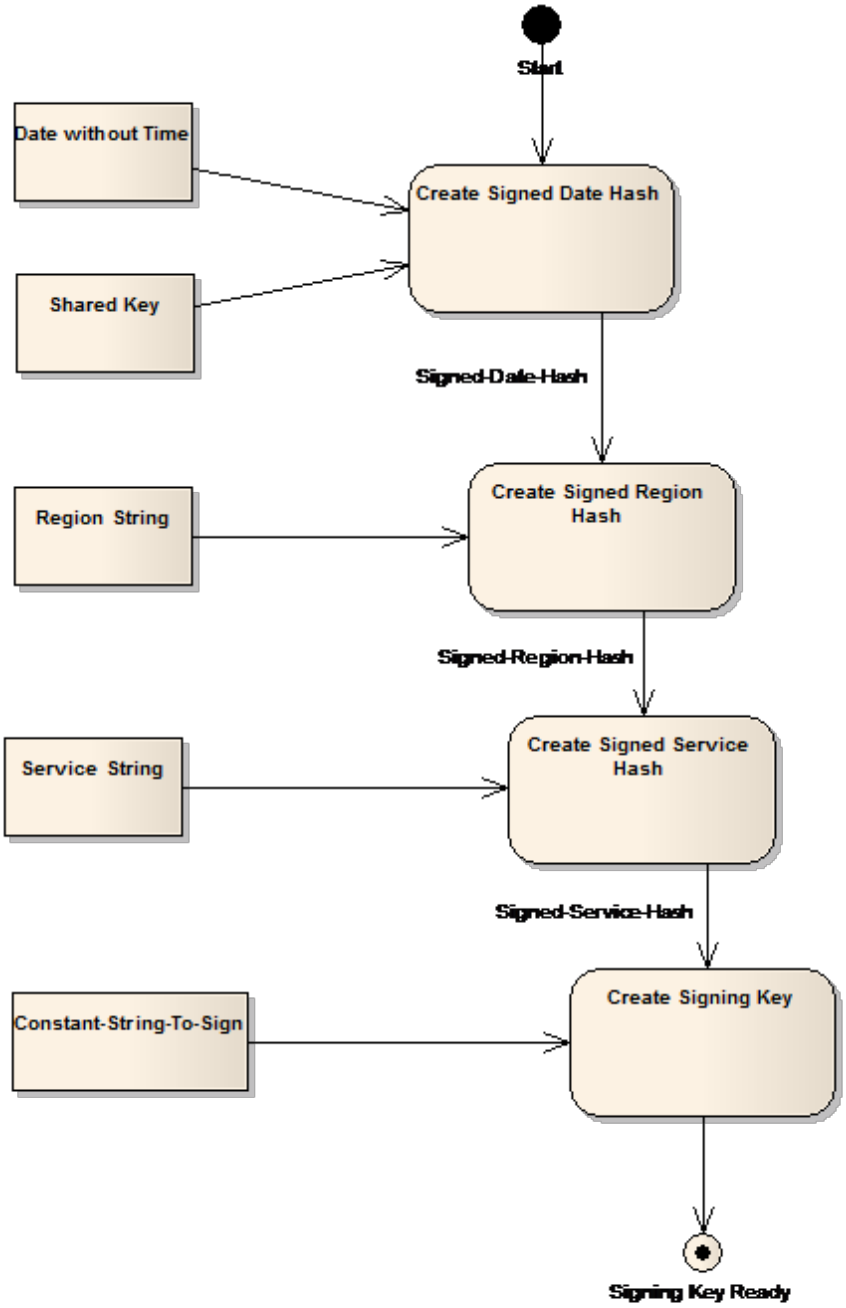
- `AWS4-HMAC-SHA256` is a literal identifying the overall signing algorithm being used.
- `<Date>` is the value of the “X-Amz-Date” parameter extracted from the redirected request.
- `<Scope>` is the scope string that was assembled as described above.
- `<canonicalized-request-string>` is the canonicalized request string assembled as described above.
- `sha256()` is a procedure that applies the standard sha256 algorithm to the canonicalized-request-string. Its output should be in the form of a string of lowercase hex digit characters.

Figure 22: Steps in Building the String to Sign

Creating the Signing Key

The process for generating signatures uses symmetric key encryption. The controller and the ECP use a shared key (the one configured on the controller’s WLAN Service’s captive portal configuration dialog) and the same encryption algorithm to generate and validate the signature.

The shared key is not used directly. Instead it is used to generate a secure hash (“HMAC”) that is then used as the key to sign the request. The process for creating the key is shown below in Figure 23: Creating the Signing Key on page 96.



In the above figure:

1 "Date without Time" is the first 8 characters in the "X-Amz-Date" attribute, which corresponds to the date only in "YYYYMMDD" format.

2 "Shared Key" is the shared key configured on the controller. It is the shared key that is paired with the identity used to create the "X-Amz-Credential" attribute in the redirected request.

3 "Region String" is the region component of the Scope string.

4 “Service String” is the service component of the Scope string.

5 “Constant-String-To-Sign” is the string “aws4_request”.

And each of the “Create...” actions consists of generating a secure HMAC using SHA256 from the inputs. The output secure hash is in binary format (not encoded as a hex character string). The output of each step acts as the signing key for the subsequent step. The signing key for the first step is the shared secret, pre-pended with the literal ‘AWS4’.

Note that for any given identity the correct signing key only needs to be computed once per day. If the calculations are cached the cache should include an entry for the previous day to cope with the request being sent just before midnight UTC. The previous day’s key only needs to be kept for a small overlapping period (perhaps 10 minutes at the most).

Figure 23: Creating the Signing Key

Creating the Signature and Verifying the Request

At this point the signature for the request is computed as a secure HMAC using SHA256. The signing key is created as described in Creating the Signing Key on page 95 and the string to sign is created as described in Building the String to Sign on page 93.

Verifying the signature in the request consists of standard string comparison between the transmitted and computed keys. If they aren’t identical the request is invalid. The client can be sent a web page containing a generic reject message or the request can be discarded silently.

Composing the Login or Splash Screen Page

How the login page is composed depends on the programming language and toolset used. This is largely outside the scope of this document. Any programming language that can be used for web development can be used to create an external captive portal.

The content on the login page will depend on the overall environment the ECP serves. It may contain little more than some terms and conditions and a button to indicate acceptance, or fields necessary to submit a user ID and password.

The redirected request will contain the attributes configured on the FF-ECP configuration dialog. Some of these could be used to decorate the login page. The other information can be input to the authentication process. For example the user may be considered authenticated only if he logs in from one of a specific set of APs.

Approving the Station

Typically, an ECP requires users to submit credentials for authentication.and credentials are submitted in an HTTP “post”. The post invokes a script on the ECP web server passing the user’s credentials to the script as arguments. The script is written by the customer and is adapted to the customer’s specific requirements.

The script file can have any name, but for example purposes the script is named “login.php”. The script can be written in any programming language that supports web development, but for this example the script is written in PHP.

The main job of the “login.php” script is to co-ordinate the station’s browser, the back-end authentication server, and the wireless controller. The “login.php” script takes the submitted credentials, sends them to an authentication server, and waits for the server’s reply. The exact steps taken here depend on the selected programming language, operating system, and the type of authentication server selected.

Once the authentication server has verified the user and potentially returned an access control role to assign to him or her, the script needs to tell the controller that the user is authenticated and, optionally the role to assign to the user. When the ECP uses the controller’s firewall friendly support it informs the controller by putting the information in the query string of a redirection response. The redirection response sends the station’s browser to a web server running on a specific interface and port of the controller that hosts the station’s session. The station’s browser normally sends a redirected request immediately and automatically.

The redirection response does not need to be signed. If it is not signed, the controller will not use any session attributes that happened to be included in the redirected request. Instead the controller will expect the redirected request to include a user ID and password. These credentials will be sent to a RADIUS server in a standard RADIUS Access-Request. The redirected request will be considered invalid if:

- The redirected request is not signed, and
  - The redirected request does not contain a user ID or password, or
  - The WLAN Service the station is using does not have at least one RADIUS server configured for authentication.

An invalid redirected request is sent to a standard error page. The error page cannot be configured at this time.

Composing the Redirection Response to Send the Browser back to the Controller

Case 1: When a RADIUS Server Will Authenticate the Station

In this scenario the redirection response must:

- Send the station to the correct interface and port on the controller. One way to ensure this is to configure the controller to include its port and IP address or FQDN in its redirection response to the station. The ECP can then cache this information and use it later to compose its redirection response.
- Include the token and WLAN ID that the controller put in its redirection response.
- Include a user name and password that will be treated as the user’s RADIUS credentials. These credentials must satisfy the standard requirements for RADIUS User-Name and User-Password attributes.

In order to trigger RADIUS authentication the redirection response must not be signed. The identity and shared secret should not be configured on the controller in this case.

If the controller is configured to redirect successfully authenticated stations to their original destination then the ECP must include in its redirection response, the “dest” parameter that was included in the controller’s redirection response.

The syntax of an unsigned FF-ECP redirect to the controller is:

```
[http | https]://<controller-IP-address-or-FQDN>{: <port>}/ext_approval.php?
token=<token>&wlan=<wlanid>&username=<userid>&password=<password>{&dest=<dest>
}
```

Where

- {...} denotes an optional component of the URL.
- [http | https] is either “http” or “https” depending on how the WLAN service’s captive portal is configured.
- :// is the literal string.
- <controller-IP-address-or-FQDN> is the controller’s IP address or Fully Qualified Domain Name. Since the controller will receive the redirect at the default HTTP or HTTPS port it does not need to be included in the redirect.
- {: <port>} is a literal colon, followed by the controller port to which the station is redirected. The port is optional and should only be included if the port is not port 80 or port 443.
- /ext_approval.php is the literal string. It is the name of the script that is invoked on the controller when the redirect is received there.
- <token> is the token taken from the redirect to the ECP.
- <wlanid> is the numeric identifier for the station’s WLAN Service as taken from the controller’s redirect to the ECP.
- <userid> is the name the captive portal would like the controller to send to the RADIUS server to authenticate this user.
- <password> is the password associated with the given user ID.
- <dest> is the original destination the station was trying to reach, as reported in the controller’s redirect to the ECP.

The order of the parameters in the query string is not important.

Examples of the redirection from the ECP to the controller expressed as a URL are:

```
https://10.21.15.42/ext_approval.php?token= OakRQ7uFYOH5E8dVD4PgvQ!!
&wlan=1&username=argon32&password=6Z*_aL40q!&dest=www.google.com
```

or

```
http://10.21.15.42/ext_approval.php?token= OakRQ7uFYOH5E8dVD4PgvQ!!
&wlan=1&username=argon32&password=6Z*_aL40q!
```

The parameters in the redirection response are summarized in the table below.

Table 18: Parameters that can be in the Redirection to the Controller, when RADIUS authentication is used

Parameter Name	Parameter Value	Mandatory	Notes
wlan	Numeric String	Yes	An identifier for the WLAN Service that the station is using to access the network.
username	Alphanumeric String	Yes	The user ID is mandatory even if the URL is signed. It is used to identify the station in reports and accounting messages, even if it is not used to authenticate the station.
password	Alphanumeric String	Yes	The password is mandatory if the station is to be authenticated using RADIUS. It must be the password that the authenticating RADIUS server associates with the user ID.
dest	URL	Conditional	The dest parameter is required only if the controller is configured to redirect the station to its original destination. The controller directs the station's browser to an error page if it is configured to redirect to the original destination and the dest parameter is not returned to the controller.

Case 2: When the FF-ECP is the Final Authority

If the ECP makes the final authentication and authorization decision, it must sign the redirection response it sends to the station's browser. If it signs the redirection it may also include options for the controller to apply to the authorized station's session including an access control role and the maximum duration for the station's session. Table 17: Parameters Available on the Redirection URL from the Controller to the ECP on page 84 lists all the parameters that can appear in a signed redirection response from the ECP, and which of them are mandatory in this case.

The syntax of an unsigned FF-ECP redirect to the controller is:

```
[http | https]://<controller-IP-address-or-FQDN>{: <port>}/ext_approval.php?
token=<token>&wlan=<wlanid>&username=<userid>{&dest=<dest>}{&role=<rolename>}
{&opt27=<max-seconds-duration>}&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=<Scoped-Credential>&X-Amz-Date=<YYYYMMDDThhmmssZ>&X-Amz-
Expires=<duration>&X-Amz-SignedHeaders=host&X-Amz-Signature=<signature>
```

Where

- {...} denotes an optional component of the URL.
- [http | https] is either "http" or "https" depending on how the WLAN service's captive portal is configured.
- :// is the literal string.
- <controller-IP-address-or-FQDN> is the controller's IP address or Fully Qualified Domain Name.
- {: <port>} is a literal colon (:), followed by the TCP/IP port number to which the station is redirected. The port is optional and should only be included if the port is not port 80 or port 443.
- /ext_approval.php is the literal string. It is the name of the script that is invoked on the controller when the redirect is received there.

- <token> is the token taken from the redirect to the ECP.
- <wlanid> is the numeric identifier for the station's WLAN Service as taken from the controller's redirect to the ECP.
- <userid> is the name the captive portal would like the controller to send to the RADIUS server to authenticate this user.
- <dest> is the original destination the station was trying to reach, as reported in the controller's redirect to the ECP.
- <rolename> is the name of a role defined on the controller that will be applied to the remainder of the station's session.
- <max-seconds-duration> is a positive integer representing the maximum duration of the station's session.
- X-Amz-Algorithm=AWS4-HMAC-SHA256 is a literal string embedded in the signed URL.
- <Scoped-Credential> is a credential in the format: <identity>/<YYYYMMDD>/world/ecp/aws4_request.
- <YYYYMMDDThhmmssZ> is the date and time at which the redirection response was sent by the ECP, in ISO 8601 compatible format.
- <duration> is a positive integer indicating the maximum duration after the X-Amz-Date that the request should be honored.
- X-Amz-SignedHeaders=host is a literal string constant.
- <Signature> is the actual signature computed over the redirection response.

The order of the parameters in the query string is not important.

The following is an example of a signed redirection response that assigns the user to a role called "Guest_Access" and limits the session duration to ten hours:

```
https://10.10.21.6/ext_approval.php?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=BigAuthInc%2F20140729%2Fworld%2Fecp%2Faws4_request&X-Amz-
Date=20140729T153754Z&X-Amz-Expires=60&X-Amz-SignedHeaders=host&dest=http%3A
%2F
%2F1.2.3.4%2Fnews.com&opt27=36000&role=Guest_Access&token=T7vb1LdUZmsuY0q9V60I
ww%21%21&username=test&wlan=1&X-Amz-
Signature=48389399c4b9e237ff64bbbd203a9abe272b8df513dff1eae8202df82ceb2c34
```

Table 19: Parameters that Can Be Included in a Signed Redirection Response from the ECP

Parameter Name	Parameter Value	Mandatory	Notes
dest	URL	Conditional	The "dest" parameter is required only if the controller is configured to redirect the station to its original destination. The controller directs the station's browser to an error page if it is configured to redirect to the original destination and the "dest" parameter is not returned to the controller.
opt27[11]	Base 10 Number	No	The maximum amount of time, in seconds, that the current session can last before being terminated. If not specified the default for the WLAN Service will be applied to the authenticated station.
role	Alphanumeric String	No	The name of an access control role defined on the controller. The controller will apply this role to the remainder of the authorized station's session. If a role parameter is not provided the controller will use the default authenticated role of the VNS that the authenticated station is accessing.
token	Alpha-numeric String	Yes	An identifier for the user's wireless session hosted on the controller that performed the redirection.
username	Alpha-numeric String	Yes	The user name is mandatory even if the URL is signed. It is used to identify the station in reports and accounting messages, even if it is not used to authenticate the station.
wlan	Numeric String	Yes	An identifier for the WLAN Service that the station is using to access the network.
X-Amz-Algorithm	Alpha-numeric string	Yes	The identifier for the algorithm used to compute the "X-Amz-Signature". This attribute must be present when the ECP is acting as the final authorizing authority. The value of this attribute is "AWS4-HMAC-SHA256" and is not configurable. The signing algorithm and the role of the identifier in it are covered in more detail in section Verifying the Signed Request on page 91.

[11] In the RADIUS protocol option number 27 is the Session-Timeout attribute.

Table 19: Parameters that Can Be Included in a Signed Redirection Response from the ECP (continued)

Parameter Name	Parameter Value	Mandatory	Notes
X-Amz-Credential	Alpha-numeric string	Yes	The identifier for the account whose shared secret was used to compute the "X-Amz-Signature". Mandatory if the ECP signs the redirection response in order to act as the final authorizing authority. The credential has the format: `<identity>/<YYYYMMDD>/world/ecp/aws4_request` where: • <identity> is the identity configured for the ECP on the controller in the WLAN Service's ECP configuration. • <YYYYMMDD> is the year, month, and day extracted from X-Amz-Date. • world/ecp/aws4_request is a constant literal string that scopes the request.
X-Amz-Date	Alpha-numeric string	Yes	This is the date and time at which the controller prepared and sent the redirection back to the user's browser. The date and time are in ASCII-encoded UTC and has the format: `YYYYMMDDThhmmssZ` This attribute must be present if the ECP signs the redirection response to indicate that it is the final authorizing authority.
X-Amz-Expires	Numeric String	Yes	This is the maximum length of time in seconds that the controller should trust the redirection response. In other words a signed redirection response from the ECP will be treated as valid only until X-Amz-Date + X-Amz-Expires. This attribute is mandatory if the ECP signs the redirection response.
X-Amz-Signature	ASCII-encoded hex string	Yes	This is the signature computed over some of the HTTP headers and parts of the query string, presented as ASCII encoded-hex. The field must be present if the ECP signs the request in order to act as the final authorizing authority.
X-Amz-SignedHeaders	Alpha-numeric String	Yes	Which of the headers in the HTTP request were included in the input to the calculation of the signature. This is present only when the controller is configured sign the redirection to the ECP, in which case it must be present.

Signing the Redirection to the Controller

Signing the redirection response is much the same process as calculating the expected signature for a URL that was received at the ECP. In fact it is the same algorithm, but the inputs to the algorithm are not taken from the request as the request is under construction.

There are only two steps involved in signing the redirection response from the ECP:

1 Compose the pre-signed redirection URL to be signed.

This step consists of building the request URL as described in Case 1: When a RADIUS Server Will Authenticate the Station on page 98 or Case 2: When the FF-ECP is the Final Authority on page 100 but leaving off the ***X-Amz-...*** parameters that are required for the signature.

2 Sign the URL, adding all parameters to the URL that are required to sign it.

This step consists of generating the signature, then appending all the ***X-Amz-...*** parameters used to the URL. This processing is described in Building the String to Sign on page 93, Creating the Signing Key on page 95, and Creating the Signature and Verifying the Request on page 97.

6 Facility Reference—RADIUS Authentication and Authorization

Overview

RADIUS is a standardized protocol for network session management. The protocol's format and behavior are defined in a series of Request for Comments (RFC) published by the Internet Engineering Task Force (IETF). The most important of the RADIUS RFCs are listed in the table below. ExtremeWireless is compliant with the RFCs listed in the table.

Table 20: Device Attributes Returned from a Successful Query

RFC Number	Title
2865	Remote Authentication Dial In User Service (RADIUS)
2866	RADIUS Accounting
2869	RADIUS Extensions
3579	RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)
3580	IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines
5176	Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)

ExtremeWireless support for RADIUS Accounting is discussed in Facility Reference—RADIUS Accounting on page 134. This chapter describes ExtremeWireless support for RADIUS authentication and authorization. It assumes basic familiarity with the RADIUS protocol. Different vendors' RADIUS servers are configured in very different ways so this chapter will not attempt to explain how to configure a server to work with a controller. Instead it will highlight RADIUS attributes that are useful for wireless session control. The chapter also assumes familiarity with the *ExtremeWireless Wireless User Guide*.

ExtremeWireless enables either the AP on its own or the AP and controller jointly to act as one RADIUS authenticator. When the AP and controller jointly act as the authenticator the controller sends the ACCESS-REQUEST to the RADIUS server and processes the ACCESS-CHALLENGE, ACCESS-ACCEPT

[12] A RADIUS authenticator mediates between the station that is trying to authenticate to gain network access and the authentication server which actually validates credentials against some sort of authorization database.

and ACCESS-REJECT responses from the server. This document refers to the scenario in which the AP and controller jointly act as the authenticator as the “joint authenticator” scenario.

When the AP acts as the sole authenticator it interacts directly with the RADIUS server. An AP must be a member of a site in order to act on its own as an authenticator. A site is a collection of APs that collaborate to manage user sessions. This permits seamless roaming among the APs of a site even when network access to the controller is unavailable. A site must be configured specifically to perform authentication at the AP. Otherwise site-based APs will require access to a controller to authenticate new user sessions.

The following figure shows the configuration for a typical site. The Local Radius Authentication check box determines whether the APs at the site serve as authenticators. Selecting this check box enables the site’s APs to act as RADIUS authenticators.

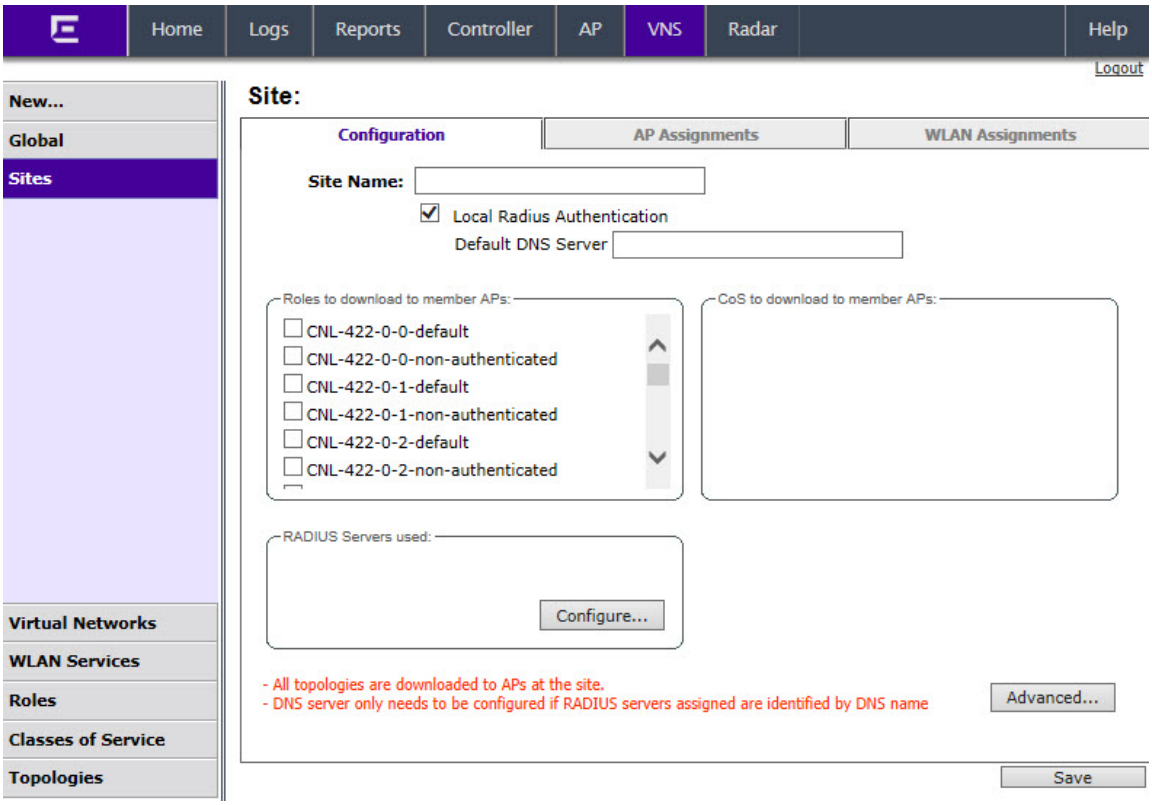


Figure 24: Configuring a Site for AP-based Authentication

ExtremeWireless supports both solicited (RFCs 2865, 2869, 3579. 3580) and unsolicited (RFC 5176) RADIUS session control. It also supports RADIUS server redundancy. These topics are discussed in more detail in the remainder of this chapter.

Note

Only the controller can receive unsolicited session control messages. The controller applies valid session control messages to the user so long as it has a network connection to the relevant AP. The controller will apply valid unsolicited RADIUS session control messages to users even if the AP was responsible for authenticating the user

Solicited/Synchronous RADIUS Authentication and Authorization

Solicited session control involves the authenticator sending an ACCESS-REQUEST message to the RADIUS server and applying the options in the server's ACCESS-ACCEPT or ACCESS-REJECT response.

There are two fundamental forms of RADIUS server-based authentication:

- RADIUS password-based authentication
- EAP authentication

An administrator must choose one of these methods when defining a WLAN Service that employs RADIUS-based authentication. The method chosen should depend on a number of factors, including:

- What authentication and authorization services are already in use in the enterprise.
- The degree of security required by the enterprise.
- Whether it is feasible for the enterprise administrator to configure the devices that will be accessing the wireless network.

RADIUS Password Authentication

Password-based authentication is the easiest authentication mechanism to administer. The RADIUS protocol intrinsically supports password-based authentication. The most common password-based authentication mechanisms in RADIUS are:

- Password Authentication Protocol (PAP)
- Challenge Handshake Authentication Protocol (CHAP)
- Microsoft Challenge Handshake Authentication Protocol (MS-CHAP)
- Microsoft Challenge Handshake Authentication Protocol version 2 (MS-CHAP v2)

ExtremeWireless natively supports these four authentication protocols. A joint authenticator scenario can use any of these protocols for captive portal authentication and MAC-based authorization. A site-based AP can use any of these protocols as part of MAC-based authorization.

Figure 25: Configuring a WLAN Service for RADIUS Password-based Authentication on page 108 shows one way to configure a WLAN Service for RADIUS-based authentication. In this case, internal captive portal is the chosen authentication mechanism. The controller in the example is configured to use a single RADIUS server (192-168-18-110) for authentication.

To use a RADIUS server for authentication:

1 Select the server from the drop-down list and click **Use**.
2 Select the **Auth** checkbox (for captive portal authentication)

[13] MAC-based authorization is a form of RADIUS password-based authentication in which a device's MAC address serves as its use ID and a single password is shared across all wireless devices. Because MAC addresses can be spoofed, MAC-based authorization should not be relied on as the sole type of authentication in networks requiring elevated security.

[14] A single WLAN Service can be configured to use up to three RADIUS servers for authentication. It only uses one server at a time but will fail over automatically to the next server in the list when the current server stops responding.

[15] If the desired server is not in the drop-down list, click **New** to add the server to the list of servers known to the controller.

The other way to use RADIUS password-based authentication is to enable MAC-based authorization. When MAC-based authorization is enabled a fourth column titled MAC appears in the list of RADIUS servers assigned to the WLAN Service. This column allows the administrator to indicate which RADIUS server is used for MAC-based authorization.

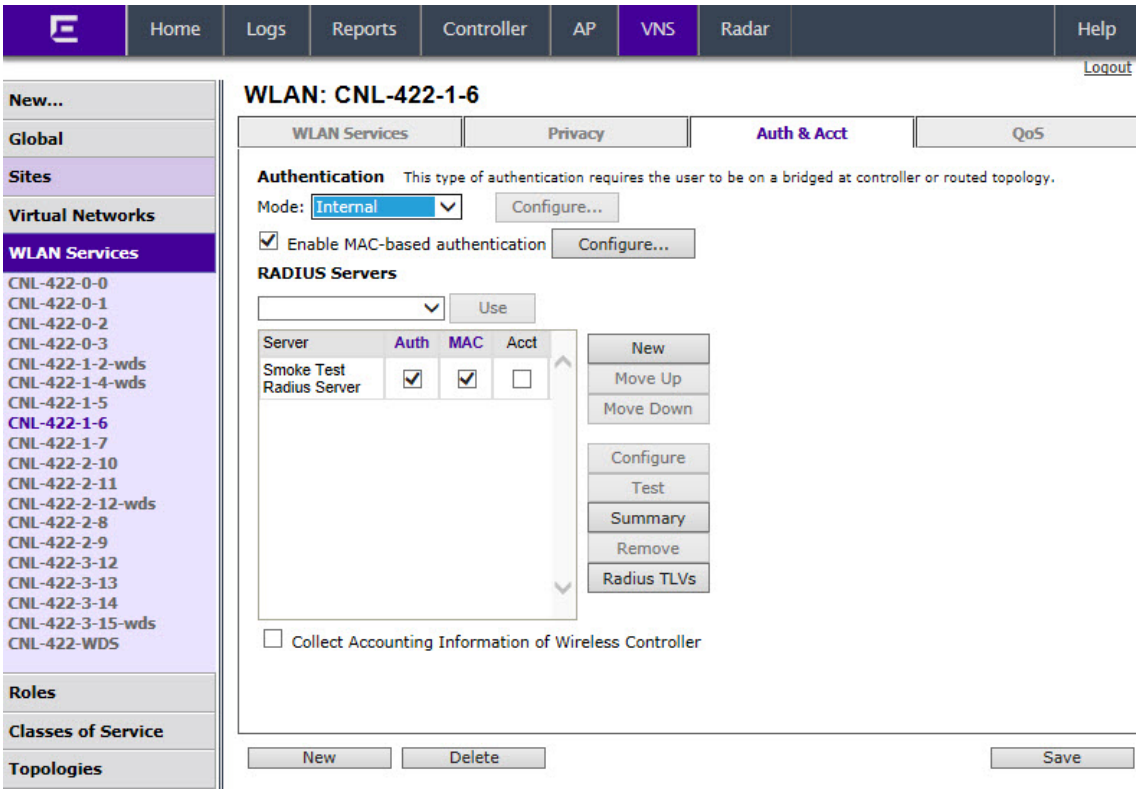


Figure 25: Configuring a WLAN Service for RADIUS Password-based Authentication

EAP Authentication

The Extensible Authentication Protocol (EAP) is more flexible than RADIUS password-based authentication. EAP is a kind of "envelope" that can contain messages for many different types of authentication. EAP can be used with certificate-based and pass-word-based authentication. EAP is a critical component of the 802.1x, WPAv1 Enterprise, and WPAv2 Enterprise standards.

There are many different EAP-encapsulated authentication protocol, including:

- EAP-AKA
- EAP-FAST
- EAP-TLS
- EAP-TTLS
- EAP-SIM
- EAP-MD5
- ZLXEAP

ExtremeWireless supports all EAP-encapsulated authentication protocols. In fact, the protocol inside EAP is transparent to ExtremeWireless.

Figure 26: Configuring Wireless for EAP-based Authentication on page 109 illustrates how to configure a WLAN Service to use EAP-based authentication. The WLAN Service's authentication mode must be set to 802.1x. The WLAN Service must be configured to use at least one RADIUS server for authentication.

EAP-TLS with mutual certificate authentication is one of the most secure authentication mechanisms, if not the most secure authentication mechanisms available, but does require a Public Key Infrastructure (PKI) and the ability to manage certificates on client devices.

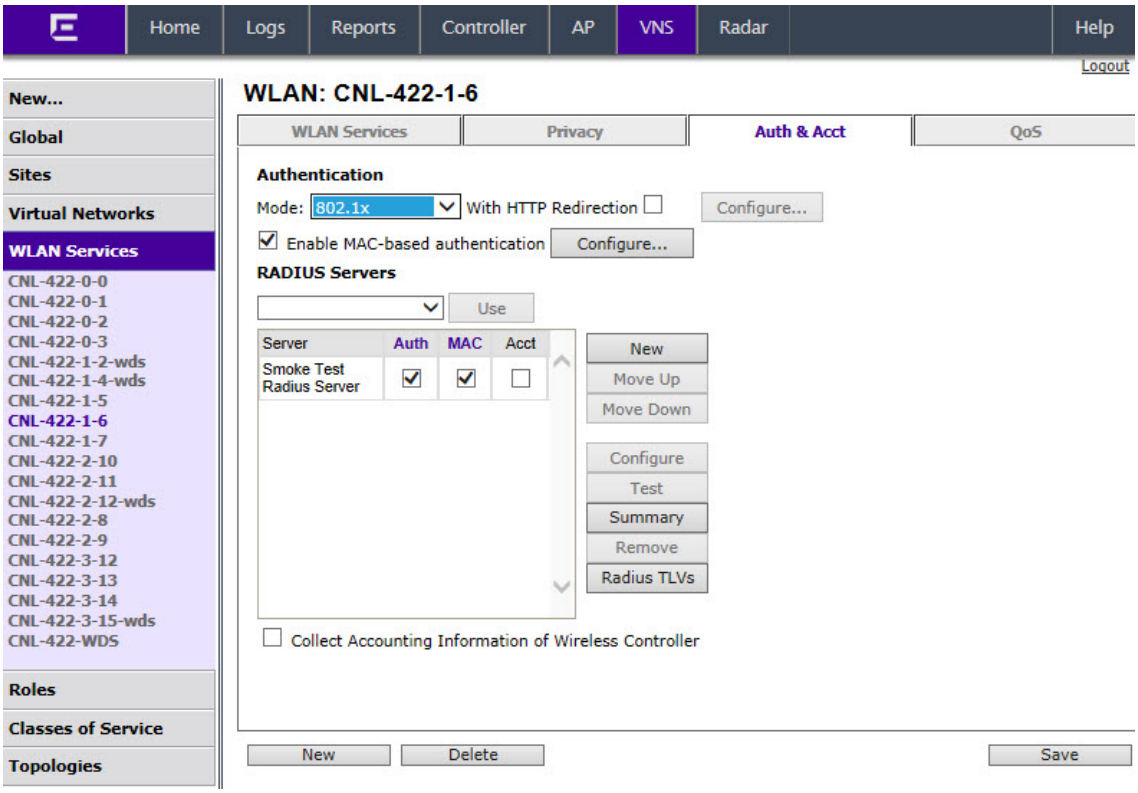


Figure 26: Configuring Wireless for EAP-based Authentication

Attributes Sent to a RADIUS Server

The authenticator can include a variety of information in the ACCESS-REQUEST message. Table 21: Attributes the Controller May Send in an ACCESS-REQUEST Message on page 110 below lists the RADIUS attributes that the controller can send to a RADIUS server in an ACCESS-REQUEST message. Many of the options can be included in or excluded from the request through WLAN Service configuration. The table indicates whether the attribute is configurable and if so the types of values it can take.

The controller and AP can send proprietary attributes (i.e., not defined in the RADIUS standard) to the RADIUS server. The RADIUS standards refer to these types of attributes as Vendor Specific Attributes (VSAs). Typically, a RADIUS server must have access to a "dictionary" defining the VSAs that it wants to use. Different RADIUS servers have varying degrees of support for VSAs.

The controller's VSAs are defined in "dictionary.siemens," which is included in Siemens VSA Dictionary in FreeRadius Format on page 156. The dictionary uses a syntax defined and supported by the FreeRADIUS server.

Table 21: Attributes the Controller May Send in an ACCESS-REQUEST Message

Attribute	Configurable	Controller / AP Sends	Notes
Acct-Session-ID	No	A string comprised of UTF-8 encoded 10646 [7] characters.	
Called-Station-Id	Yes	An octet sequence identifying the service being accessed.	By default this will be an octet sequence containing the ASCII encoding of the BSSID to which the device is associating. The format of the MAC address is controlled by the "MAC Address Format" attribute on the VNS module's Global Authentication page. The BSSID can be replaced with a Zone identifier. For more information please refer to Zones on page 113.
Calling-Station-Id	No	An octet string containing the ASCII encoding of the MAC address of the device to be authenticated.	One of twelve different MAC address formats can be specified on the VNS module's Global Authentication page. The same format will be used for all MAC addresses sent to all RADIUS servers that the controller or AP communicates with.
Chargeable-User-Identity	Yes	A string value that identifies a particular user. The RADIUS server determines the format and content of the string.	Identifies a user for any roaming transactions that take place outside of the network.
EAP-Message	No	An octet sequence containing one message in an EAP exchange. Sent from station to RADIUS server.	Expected and forwarded automatically when 802.1x authentication is enabled for a WLAN Service.
Location-Data	Yes		Enabling this attribute lets the RADIUS server use the user location data.
Location-Information	Yes.	A value of 2 -255	
Message-Authenticator	No	An eighteen octet sequence containing a signature that authenticates the message.	Must be used with EAP. Not to be confused with the RADIUS "Authenticator" field.
NAS-Identifier	Yes	An octet string representing a name.	This is either a string configured by the administrator or the name of the VNS to which the user is authenticating.

[16] Wireshark also uses the FreeRADIUS dictionary format to decode VSAs in RADIUS messages.

Table 21: Attributes the Controller May Send in an ACCESS-REQUEST Message (continued)

Attribute	Configurable	Controller / AP Sends	Notes
NAS-IP-Address	Yes	An IP address in the form of a sequence of four unsigned octets.[17]	The administrator can configure an arbitrary IPv4 address for the NAS-IP-Address. In joint authentication, if the NAS IP is not explicitly configured then if the controller has an IP address on the default topology of the default non-authenticated role of the user's VNS that address will be used as the NAS-IP address.
NAS-Port	No	A positive integer port number.	
NAS-Port-Type	No	A four-octet sequence containing the decimal value 18.	The number 18 represents the "Wireless-Other" NAS-Port-Type.
Operator-Name	Yes	A value of 0x34 - 0-xFE	Identifies the owner of an access network using a unique ID.
Service-Type	Yes	A positive integer of 1 - 19 indicating the type of service.	This is the only value that is sent "Framed".
Siemens-AP-Name	Yes	An octet sequence containing the name of the AP that is authenticating the user/device.	This is the name of the AP as configured in the AP section of the controller GUI. By default it is not sent.
Siemens-AP-Serial	No	An octet sequence containing the unique serial number of the AP that is authenticating the user/device.	The serial number of the AP is set at the factory and can't be altered. The Siemens AP Serial attribute is sent when ExtremeWireless is configured to send the Siemens AP Name in the ACCESS-REQUEST message. By default it is not sent.
Siemens-BSS-MAC	Yes	An octet sequence containing the BSSID to which the station associated.	By default this is not sent.
Siemens-Egress-RC-Name	Yes	The three-octet sequence 'n/a'.	Obsolete. A station can have many egress rate profiles applied by a single role. Use the role name to infer which rates are being applied to traffic sent to the station.
Siemens-Ingress-RC-Name	Yes	The three-octet sequence 'n/a'.	Obsolete. A station can have many ingress rate profiles applied by a single role. Use the role name to infer which rates are being applied to traffic sent by the station.
Siemens-Policy-Name	Yes	An octet sequence containing the name of the role currently applied to the station.	By default this is not sent. To have it sent, enable the **Role Name** option in the **VSAs** section of the **EWC's RADIUS Access-Request Message Options Dialog**.

[17] Octet is a synonym for byte. An octet can take any value in the range 0-255 inclusive.

Table 21: Attributes the Controller May Send in an ACCESS-REQUEST Message (continued)

Attribute	Configurable	Controller / AP Sends	Notes
Siemens-SSID	Yes	An octet sequence containing the SSID of the WLAN service to which the station associated.	By default this is not sent.
Siemens-Topology-Name	Yes	The name of the topology to which the station's traffic is contained.	This attribute is deprecated and may contain 'n/a' in the future. As of release 9.01, a station's traffic can be split across multiple VLANs at once. This field will only have a value if the default action of the role assigned to the user is 'Contain to VLAN'. In that case this attribute contains the name of the topology defined on the controller for this VLAN. By default this is not sent.
Siemens-VNS-Name	Yes	An octet sequence containing the name of the VNS to which the station is authenticating.	This is the name given to the VNS in the VNS configuration section of the Virtual Networks module of the controller GUI. By default this is not sent.
User-Name	No	Identifier to be authenticated, as entered by user.	For MAC-based authorization, the user ID is the device's MAC address.
User-Password	No	Password, as entered by user.	For MAC-based authorization the password is part of the WLAN Service configuration and is shared by all authenticating devices.

Attributes like the VNS Name or Called-Station-ID can be used to implement more advanced access control policies. For example, an enterprise might require each user to access the network through one specific VNS. Different types of users must use different VNSs. In this case, a user's credentials aren't sufficient to determine whether the user should be authorized to use the network they are authenticating to. ExtremeWireless can be configured to include the Siemens-VNS-Name VSA in the ACCESS-REQUEST to allow the RADIUS server to detect when a user is authenticating to the wrong service.

As another example, an enterprise might want to assign its users to different roles or different VLANs depending on where each user connects to the network. Enterprise policy might prohibit wireless users from accessing the Internet from inside the data center and permit access when they are outside it. Again, the user's credentials aren't sufficient to determine the access policy to apply to the user. The Called-Station-Id and the Siemens-AP-Serial VSA can be used to identify the AP that the user is associated to during authentication. This information locates the user for the purpose of applying location-specific policy.

VSA configuration is on a per-WLAN Service basis, so that ACCESS-REQUESTs sent for different WLAN Services can contain different VSA combinations. Figure 27: Configuring the Controller to Send Siemens VSAs on page 113 illustrates how to configure ExtremeWireless to send Siemens VSAs in the RADIUS ACCESS-REQUEST message.

1 Click the **RADIUS TLVs** button on the WLAN Service **Auth& Acct** tab.

2 In the dialog box that appears, select the VSAs that should be included in the ACCESS-REQUEST message.

3 Click OK and then Save.

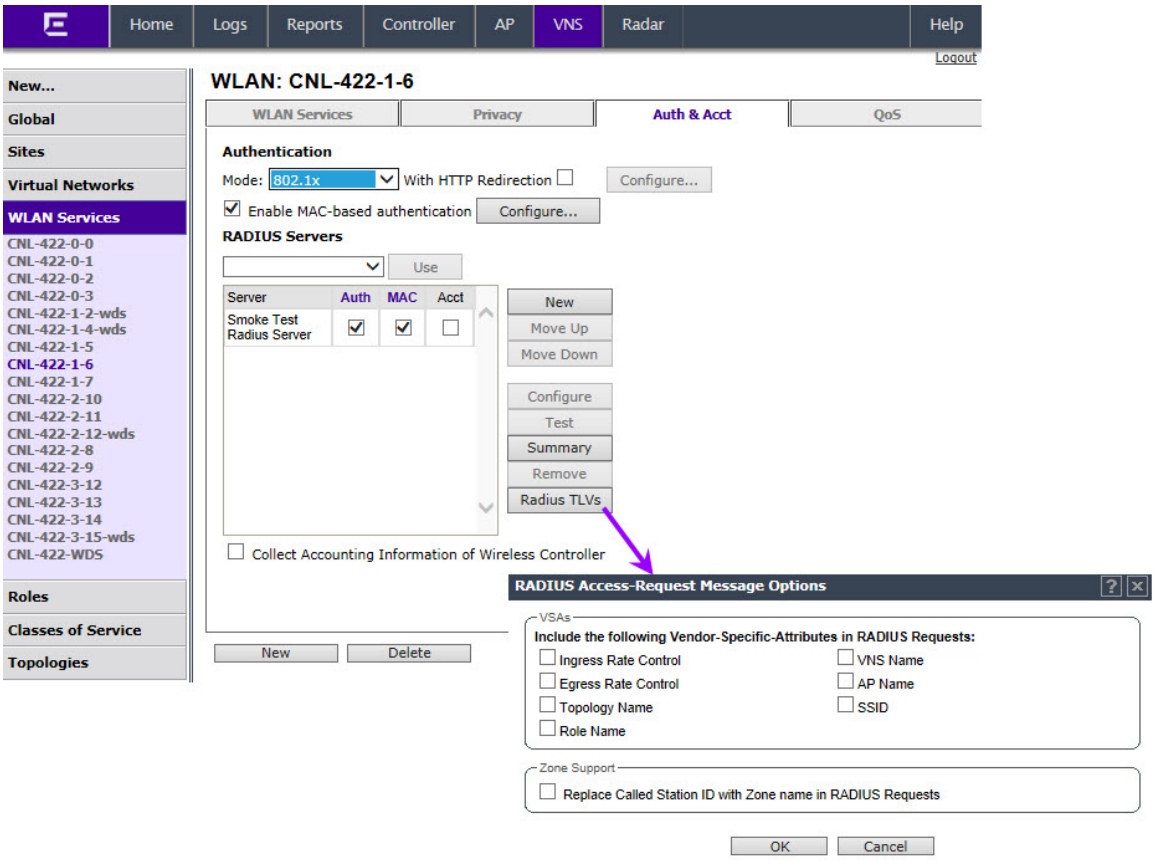


Figure 27: Configuring the Controller to Send Siemens VSAs

The AP Name option controls both the AP Name and AP Serial number VSAs.

Zones

A zone is a group of APs that share a common zone label, which is an arbitrary text string that allows the administrators to group APs for purposes of authentication and authorization. ExtremeWireless can be configured to send the zone label in the Called-Station-Id field of an ACCESS-REQUEST message instead of a BSSID.

Zone labels are targeted to deployments that use location-based authorization policies. As mentioned earlier, by default ExtremeWireless puts the BSSID that the station is associating to in the Called-Station-Id field. Because BSSIDs are unique, a single BSSID can be at a single location, so the BSSID could be used to identify the location of the user.

The problem with using BSSIDs to locate users is that a typical deployment has many of them. A single AP can have 16 BSSIDs (16 WLAN Services, 8 per radio). A single location might have 6 to 10 APs and an enterprise network can have dozens or hundreds of locations. This very quickly adds up to a lot of RADIUS server configuration.

Using zone labels instead of BSSIDs reduces the configuration effort for location-based policy by an order of magnitude. Also, the zone label can be a human-readable, meaningful string like “Atherton Hall” or “Dafoe Library.” Using zone labels greatly reduces the risk of incorrectly configuring the RADIUS server.

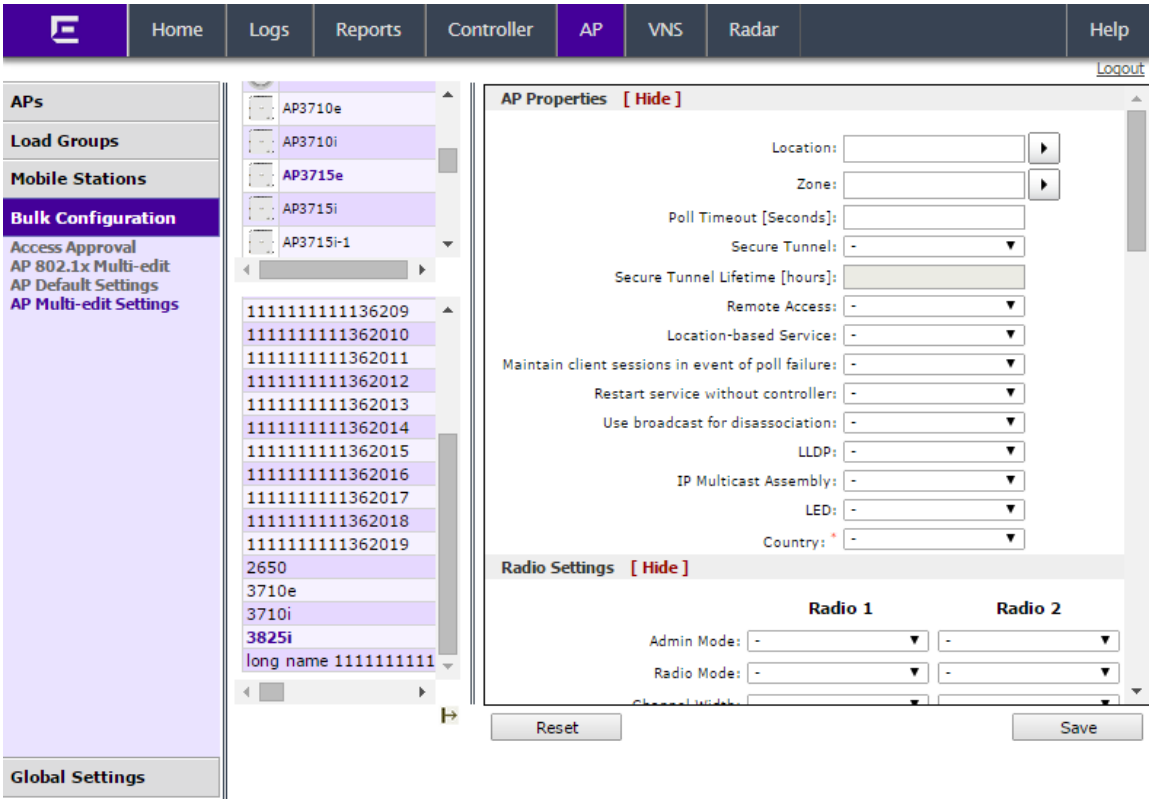


Figure 28: Assigning Zone Labels to APs Using Multi-edit

Zone labels can be created and applied on the individual AP configuration page or on the AP Multi-edit Settings page. Figure 29: Configuring a WLAN Service to Send the Zone Label on page 115 above illustrates the application of a zone label using the multi-edit page. After selecting a group of APs to edit, either type a new label into the Zone field or select one of the entries in the drop-down list. Saving the multi-edit page applies the entered label to the selected APs.

The AP and controller RADIUS clients must be configured to send the zone label to the RADIUS server. This is configured on a per-WLAN Service basis. So some WLAN Services might be configured to send the BSSID while others are configured to send the zone label.

To configure a WLAN Service to send the zone label to the RADIUS server:

1 Open the Auth& Acct tab of the WLAN Service to be configured.
2 Click the RADIUS TLVs button. A dialog box like the one in Figure 29: Configuring a WLAN Service to Send the Zone Label on page 115 will appear.
3 Select the Replace Called Station ID with Zone name in RADIUS Requests checkbox.
4 Click OK and then Save.

[18] The drop-down list will be empty until the first zone label is assigned to an AP.

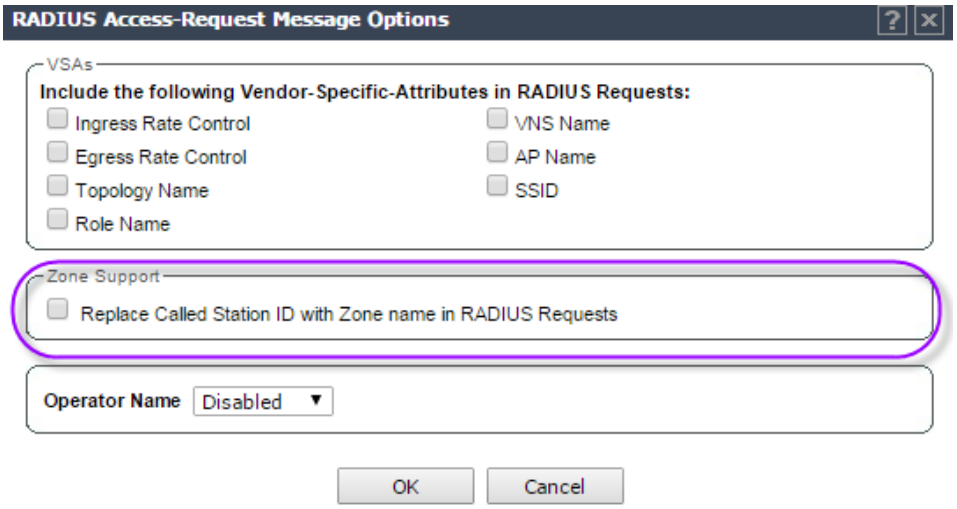


Figure 29: Configuring a WLAN Service to Send the Zone Label

The RADIUS server will need to be configured to make authorization decisions based on the Called-Station-Id field. Please consult the RADIUS server's documentation to determine how to do this.

Attributes Received from a RADIUS Server

ExtremeWireless will work with basic ACCESS-REJECT and ACCESS-ACCEPT responses. An ACCESS-REJECT causes ExtremeWireless to terminate the user's session. An ACCESS-ACCEPT response without optional attributes causes ExtremeWireless to allow the target user onto the network in a fully authenticated state and assigned to the default authenticated role of the user's VNS. This requires relatively little RADIUS server configuration and works very well when the enterprise only needs to distinguish authenticated users from unauthenticated ones.

Many enterprises have more complex access policies. These enterprises have many classes of network user, each with its own access privileges and restrictions. ExtremeWireless accepts a variety of RADIUS attributes in the ACCESS-ACCEPT response that can customize access for each specific user. below lists the attributes that ExtremeWireless accepts in an ACCESS-ACCEPT response.

Table 22: RADIUS Attributes Accepted by ExtremeWireless

Attribute	Controller / AP Expects	Notes
Acct-Interim-Interval	A four-octet sequence interpreted as the number of seconds between interim accounting reports for this user.	Overrides the default value configured on the controller for each RADIUS server.
Chargeable-User-Identity	A string value that identifies a particular user. The RADIUS server determines the format and content of the string.	Identifies a user for any roaming transactions that take place outside of the network.
Class	An octet string at least one octet in length	

Table 22: RADIUS Attributes Accepted by ExtremeWireless (continued)

Attribute	Controller / AP Expects	Notes
Event-Timestamp	A four-octet value field containing the number of seconds since January 1, 1970 00:00 UTC.	The controller expects the Event-Timestamp in RFC 3576 (Disconnect, Change-of-Authorization) requests.
EAP-Message	An octet sequence containing one message in an EAP exchange. Sent from station to RADIUS server.	Expected and forwarded automatically when 802.1x authentication is enabled for a WLAN Service.
Filter-Id	An octet string containing the name of a role defined on the controller. The role will be applied to the authenticated user's session.	If the AP is the authenticator then the Filter-Id must be the name of a role that is assigned to the site to which the AP belongs.
Idle-Timeout	A positive integer representing the maximum number of consecutive seconds that a user can be idle before the user's session is terminated.	If the Idle-Timeout is not specified in the ACCESS-ACCEPT response the controller or AP will use the idle timeout value configured in the Advanced Settings of the user's WLAN Service.
Login-LAT-Group	Obsolete	Formerly used by ExtremeWireless to designate a Child VNS on which to place the user.
Login-LAT-Port	Controller expects either integer 0 or integer 1.	If the attribute is set to 1, the user is considered fully authenticated. If the value is set to 0, the user will be allowed onto the network but will be treated as unauthenticated. If captive portal authentication is enabled on the VNS the user is accessing then denied HTTP traffic will cause the user's browser to be redirected to the captive portal. Refer to Login-LAT-Port on page 121 for details.
Message-Authenticator	An eighteen-octet sequence containing a signature that authenticates the message.	Must be used with EAP. Not to be confused with the RADIUS "Authenticator" field.
MS-MPPE-Recv-Key	A session key for use by the Microsoft Point-to-Point Encryption Protocol (MPPE).	Used with 802.1x authentication.
MS-MPPE-Send-Key	A session key for use by the Microsoft Point-to-Point Encryption Protocol (MPPE).	Used with 802.1x authentication.
Reply-Message		Returned in success packets and notification messages.
Service-Type	A four-octet value field containing one of the integers 6, 7 or 8.	Only used for RADIUS-based administrative login to a controller. Refer to RADIUS-based Administrative Login on page 125 for details.

Table 22: RADIUS Attributes Accepted by ExtremeWireless (continued)

Attribute	Controller / AP Expects	Notes
Session-Timeout	A positive integer representing the maximum length of the authenticated user's session, expressed in seconds.	The maximum number of seconds of service to be provided to the user. The effect of the session timeout may depend on the Terminate-Action attribute. The normal action upon reaching the session-timeout is to terminated the user's session. If it is not specified then the Session Timeout configured for the WLAN Service on the controller is used for this session.
Siemens-URL-Redirection	If present in the response, it must contain a valid HTTP or HTTPS URL.	This can be used to override where the station is redirected. The URL can be up to approximately 9000 characters in length. Multiple instances of the VSA in a single response are concatenated to form a single URL.
Termination-Action	A four-octet string containing either the value 0 or 1.	If not present in the response or if it has the value zero, ExtremeWireless simply terminates the user's session when the Session-Timeout is reached. If the Termination-Action is set to 1 and 802.1x authentication is configured for the WLAN Service, ExtremeWireless will initiate re-authentication.
Tunnel-Medium-Type	Three octets containing the integer 6, representing all 802 media.	This is the only supported tunnel medium type for ExtremeWireless. It is only relevant when a Tunnel-Private-Group-ID containing a VLAN ID is returned.
Tunnel-Private-Group-ID	Either an octet sequence containing the name of a role defined on the authenticator or a valid integer VLAN ID (1-4094). The VLAN ID is encoded as a string.	
Tunnel-Type	The integer 13, indicating that the tunnel type is a VLAN.	This is the only type of tunnel that is supported. Only relevant when a Tunnel-Private-Group-ID containing a VLAN ID is returned.

Filter-Id

The Filter-Id attribute, if present, must contain the name of a role that is defined on the controller or AP that is being asked to apply the role. The Filter-Id attribute can have either of two formats:

1 The Filter-Id contains an ASCII-string that exactly matches the name of a role defined on the controller or site-based AP being asked to apply the role. The name is case sensitive so "ADMINISTRATOR" and "Administrator" would refer to two different roles.

2 The Extreme Networks “Decorated Filter-Id” format, which is formatted like one of the following:

```
Extreme Networks:version=<n>:policy=<role-name>
Extreme Networks:version=<n>:mgmt=<management-access>
Extreme Networks:version=<n>:mgmt=<management-access>:policy=<role-name>
```

where:

- `<n>` is replaced by a positive integer version number. Currently, the correct version number is “1”.
- `<role-name>` is replaced by an ASCII string that exactly matches the name of a role defined on the controller or site-based AP being asked to apply the role
- `<management-access>` is replaced by one of the following:
  - `ro` – read-only access
  - `rw` – read-write access
  - `su` – super-user access

An example of a valid Decorated Filter Id

```
Extreme Networks:version=1:policy=Administrator
```

ExtremeWireless only uses the `policy=<role-name>` portion of the decorated Filter-Id. Other Extreme Networks equipment use the `mgmt=<management-access>` portion of the decorated Filter-Id to identify the level of management access to the device that should be granted to the authenticating user. The ExtremeWireless Controller uses the RADIUS Service-Type attribute for this purpose. The controller simply ignores the `mgmt=<management-access>` part of the Filter-Id.

Regardless of which format is used, the entire string is placed in the value portion of the RADIUS Filter-Id Tag-Length-Value (TLV) attribute.

Dealing with Invalid Filter-Id Assignments

In some cases it is possible that the RADIUS server will send a Filter-Id to the controller that cannot be used. This is almost always a configuration issue, either on the controller or the RADIUS server. Regardless, the controller or AP must apply some role to the user’s session. ExtremeWireless implements a configurable “Invalid Role Action” that determines the role applied to the user when the one named in the RADIUS response can’t be used. The following figure shows how to configure the invalid role action using the controller GUI.

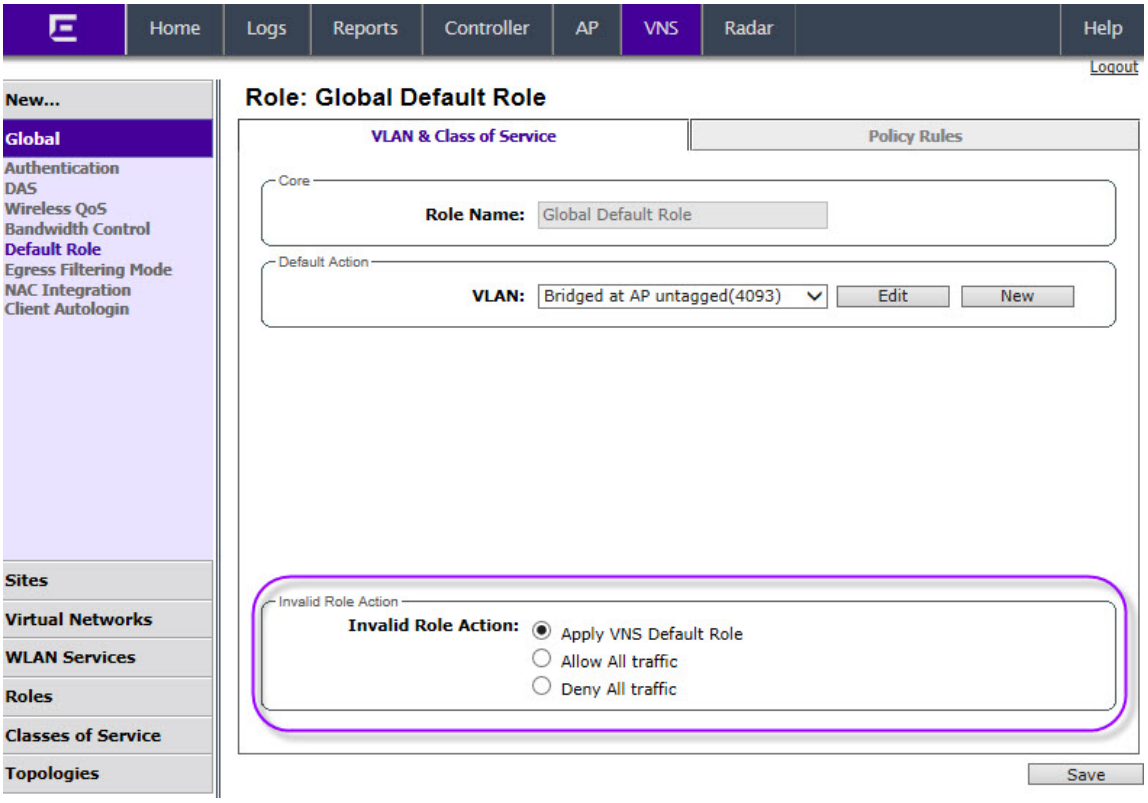


Figure 30: Configuring the Invalid Role Action

These actions are available for dealing with invalid roles:

- Apply VNS Default Role

  This is the default invalid role action. Every VNS must have one or two default roles associated with it. When an invalid role is received and this option is selected, the authenticator substitutes the VNS' default authenticated role or its default non-authenticated role for the invalid role. Which role is applied depends on whether the target user is fully authenticated.

- Allow All traffic

  This will allow all traffic to or from the user-assigned role to the invalid role. Obviously this can be very insecure. It may be an acceptable option for networks that enforce only a few restrictions in their policies and which might have to operate for a prolonged period before a detected problem can be corrected.

- Deny All traffic

  The user will be assigned to a role that blocks all traffic to or from the user. While the user will be able to associate to the wireless network, he or she will not be able to send traffic to the network or receive any from it. This is the most secure option, although it will certainly result in support calls.

Tunnel-Private-Group-ID

ExtremeWireless uses the Tunnel-Private-Group-ID in two ways:

1 As an alternative to the Filter-Id TLV.

2 To identify the VLAN/Topology to which the authenticated user's traffic is contained by default.

The authenticator (AP or controller) will treat the Tunnel-Private-Group-ID like a Filter-Id if:

- The Tunnel-Private-Group-ID contains an ASCII string.
- That string exactly matches the name of a role defined on the authenticator.
- The RADIUS response does not contain a Filter-Id.

The authenticator (AP or controller) will apply the Tunnel-Private-Group-ID as a VLAN ID if the TLV contains a single valid VLAN ID.

The exact way in which the Tunnel-Private-Group-ID TLV is used depends on configuration on the controller. This is covered in more detail in the next section.

Combining the Filter-Id TLV with the Tunnel-Private-Group-ID TLV

A single RADIUS ACCESS-ACCEPT response can contain both a Filter-Id and a Tunnel-Private-Group-ID attribute. The user interface allows the administrator to configure whether and how each attribute is used. Figure 31: Configuring How the Tunnel-Private-Group-ID and Filter-Id are used on page 120 below shows the tab on which the exact use of the Filter-Id and Tunnel-Private-Group-ID are configured. The configuration is global, applying to all WLAN Services defined on the controller.

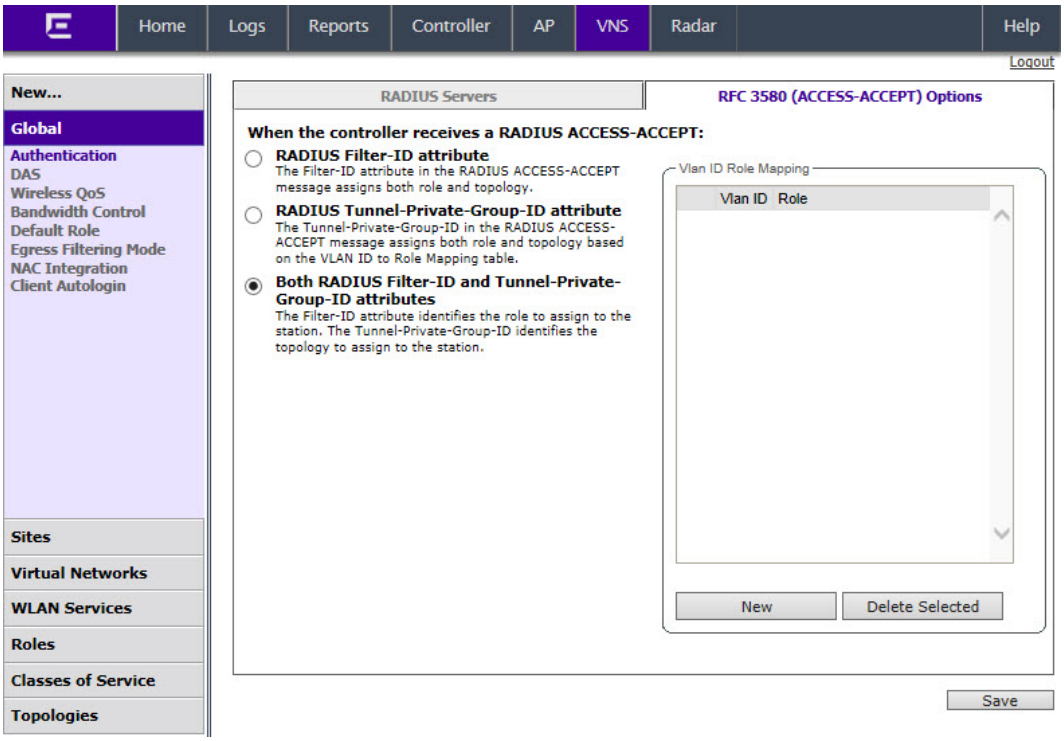


Figure 31: Configuring How the Tunnel-Private-Group-ID and Filter-Id are used

These options are available:

- RADIUS Filter-ID Attribute

  In this mode the response is expected to contain a Filter-Id TLV and the TLV must contain a string exactly matching the name of a role defined on the authenticator. If the Tunnel-Private-Group-ID is present, it is ignored.

- RADIUS Tunnel-Private-Group-ID

  In this mode the Tunnel-Private-Group-ID is expected to map to a role definition. The mapping is done using the VLAN ID-to-Role mapping table that can be seen in Figure 31: Configuring How the Tunnel-Private-Group-ID and Filter-Id are used on page 120. The table is ignored unless this option is enabled. Any Filter-Id TLV in the response is ignored.
- Both RADIUS Filter-Id and Tunnel-Private-Group-ID attributes

  In this configuration the controller or AP expects the ACCESS-ACCEPT response to contain a Filter-Id and optionally contain a Tunnel-Private-Group-ID. If the response only contains a Filter-Id and the Filter-Id exactly matches the name of a role defined on the authenticator then the role is applied to the user's traffic. The role's default action is applied to all of the user's traffic that does not specifically match a policy rule in the role. This is the factory-default setting for this attribute.

The *ExtremeWireless User Guide* describes exactly how to configure this setting on the controller.

Note
This feature is sometimes referred to as "RFC 3580 Support" because RFC 3580 defined how the RADIUS protocol uses tunnel attributes to assign users to specific VLANs.

Login-LAT-Port

The RADIUS standard originally intended Login-LAT-Group and Login-LAT-Port TLVs to manage access to DEC Local Area Transport (LAT) networks. Since LAT is not in wide use, ExtremeWireless re-purposed the TLVs for managing wireless network access. It no longer uses the Login-LAT-Group TLV. ExtremeWireless continues to use the Login-LAT-port TLV as a means of explicitly controlling a user's authentication state.

A RADIUS server can indicate that a set of credentials are invalid by returning an ACCESS-REJECT response. The RADIUS server can indicate that the user successfully authenticated by returning an ACCESS-ACCEPT. ExtremeWireless always terminates the user's session when it receives an ACCESS-REJECT response for the user and always allows the user onto the network when it receives an ACCESS-ACCEPT response.

ExtremeWireless does not automatically assume that the ACCESS-ACCEPT means the station is authorized for full network access. The ACCESS-ACCEPT allows the user onto the network but he or she is not necessarily considered authenticated. How the ACCESS-ACCEPT affects the user's authentication state depends on the type of authentication that occurred:

- If the ACCESS-ACCEPT was sent in response to MAC-based authorization then the user is considered unauthenticated. If the user is on a WLAN Service that uses captive portal authentication, the user's denied HTTP traffic is redirected to the configured captive portal.
- If the ACCESS-ACCEPT was sent in response to captive portal or 802.1x authentication then the user is considered fully authenticated. The user's denied HTTP traffic is dropped.

The Login-LAT-Port attribute allows the RADIUS server to override this default behavior. If the Login-LAT-Port setting is not included in the response then ExtremeWireless applies this default behavior.

As mentioned in Table 21: Attributes the Controller May Send in an ACCESS-REQUEST Message on page 110, if the Login-LAT-Port has a value of 1 then the controller or AP regards the user as fully authenticated. If the TLV has a value of 0 then the controller or AP regards the user as unauthenticated.

This mechanism is useful when two-phase authentication is required. Cases in which two-phase authentication are required include:

- Only authorized devices are allowed access to the network and a pool of those devices is shared among different users. MAC-based authorization or 802.1x authentication can be used to authenticate the device and captive portal authentication can be used to authenticate the user.
- The IT department periodically needs to redirect all authorized users to a notification page and ensure that all networks users must have seen the page before using the network. Non-authorized users should not be allowed on the network at all and so will not see the notification.
- All authorized devices need to be redirected to a "scanning station" such as a Network Access Controller (NAC) to ensure compliance with corporate software policy. Devices only need to be scanned periodically and not every time they arrive on the network.
- Customers buy time on a network. The first time they access the network they must be directed to a page where they can purchase network access. Subsequently, customers should not have to see this page again until they run out of access credits.

In each of these situations it is necessary to have some preliminary identification of the device accessing the network in order to decide what to do with it.

MAC-based authorization always occurs as soon as the device associates to the network. MAC-based authorization is completely transparent to the user. The authentication server can use MAC-based authorization as an opportunity to consult a database to determine things like whether the user's device is due for another security scan or whether the user has enough credits for another network session. If the authorization server is satisfied then it can return a Login-LAT-port setting of one to the controller. The controller will consider the user fully authenticated and not redirect his or her blocked HTTP traffic to a captive portal.

If the authorization server determines that the user needs further processing before he or she can get full network access it can return a Login-LAT-port setting of zero. The controller will redirect the blocked HTTP traffic to the configured web server address. When the server is satisfied that the user is allowed full network access, it can use either the HTTP or RADIUS unsolicited session control APIs to assign the user to a new policy and mark the user as fully authenticated.

The Login-LAT-Port TLV can be used to override the default handling of an ACCESS-ACCEPT response when the WLAN Service uses 802.1x authentication. 802.1x authentication can be combined with captive portal authentication by selecting 802.1X authentication, enabling the With HTTP Redirection option and using the Configure button to specify a redirection URL. The figure below shows where to configure these settings on the WLAN Service's Auth & Acct configuration page.

Figure 32: Combining 802.1x and Captive Portal Authentication

802.1x authentication, like MAC-based authorization always occurs before captive portal authentication. If the user passes 802.1x authentication, the RADIUS server sends an ACCESS-ACCEPT to the ExtremeWireless authenticator. ExtremeWireless considers the user to be fully authenticated when it

receives an ACCESS-ACCEPT response for 802.1x authentication. The user's denied HTTP traffic simply is dropped.

The RADIUS server can override this behavior by including the Login-LAT-Port TLV with a value of zero in the ACCESS-ACCEPT response. In this case the user will be considered "partially authenticated." The user will not need to undergo another 802.1x authentication as a result of this setting. Any of the user's denied HTTP traffic will be redirected to the configured URL. This means that authorized users can be redirected to the captive portal sometimes and not others based on details available to the RADIUS server. It also means that some users can be redirected to the captive and not others based on information available to the RADIUS server at authentication time.

Siemens-Redirection-URL

The Siemens-Redirection-URL VSA allows the RADIUS server to control where the denied HTTP traffic of unauthenticated or partly authenticated users is redirected. The precise way that the URL is used depends on whether it is returned in an ACCESS-ACCEPT response for MAC-based authorization or for internal captive portal authentication.

If the Siemens-Redirection-URL is returned in the response for a MAC-based authorization then the controller saves the URL with the user's session details. If the user is not fully authenticated, the denied HTTP traffic is converted to redirections to the URL returned by the RADIUS server. In effect, this allows the RADIUS server to decide on a per-user basis which captive portal will be used for authentication.

If the Siemens-Redirection-URL is returned in response to internal captive portal authentication, how it is used depends on how the internal captive portal is configured. The ExtremeWireless internal captive portal can be configured to do one of the following actions after a successful authentication:

- Redirect the user to the URL he or she was originally trying to access when the user was redirected to the captive portal.
- Redirect the user to a URL specified in the WLAN Service's captive portal configuration. All successfully authenticated users accessing the WLAN Service are redirected to the same destination. This is useful if all users need to be directed to an announcements or instructions page.
- Redirect the user to a page that contains controls for managing the user session. This page has controls for displaying session status, including usage time and for explicitly logging out of the session. By default the page also includes a link to the destination the user was trying to reach when redirected to the captive portal. The user can click on the link to continue on to his original destination.

  This option is useful when the user is being billed for usage and so needs precise control over when his session terminates. This page is implemented by controller software and is configurable using the controller's captive portal page editor.

In the first two cases, the Siemens-Redirection-URL overrides the normal redirection behavior. If the RADIUS server does not return a Siemens-Redirection-URL then the user will be redirected according to controller configuration. If the RADIUS server returns a Siemens-Redirection-URL then that will override the configured default for that specific user.

In the third case, the link to the URL the user was originally trying to access is replaced by a link to the URL returned in the Siemens-Redirection-URL VSA.

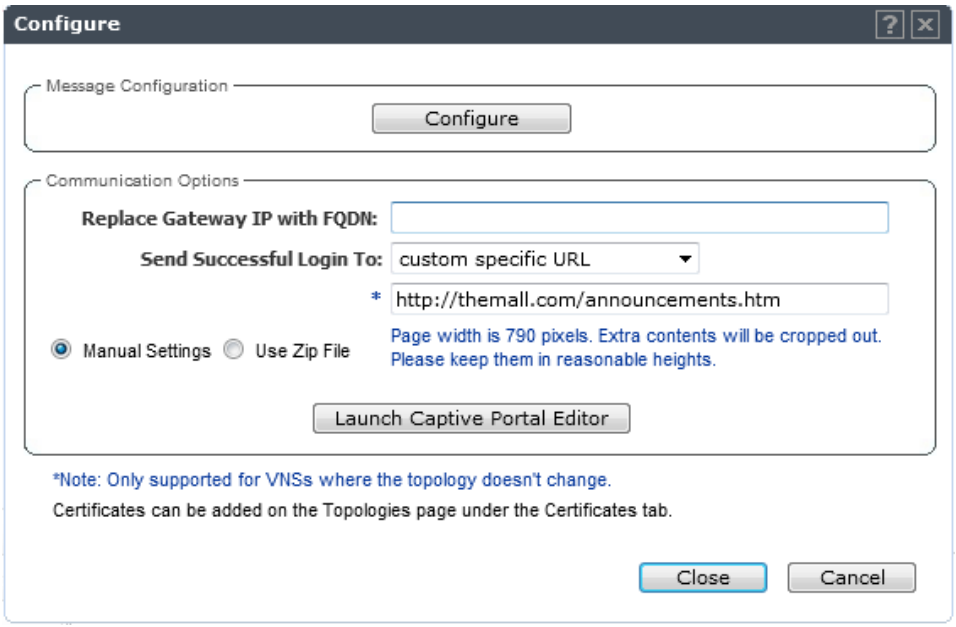


Figure 33: Configuring a WLAN Service to Redirect Successfully Logged In Users to a Specific Page

The URL has no effect when the WLAN Service is configured for External Captive Portal. The ECP is free to redirect the user as it sees fit. The VSA has no effect if it is returned in the response for MAC-based authorization and the same response includes a Login-LAT-Port TLV with a value of one.

Note

Multiple instances of the Siemens-Redirection-URL VSA can be included in a single response message. In this case, the values of the VSA instances are concatenated to produce a single redirection URL. This is useful if a long redirection URL is required. It also means that the administrator must ensure that the RADIUS server does not return two or more unrelated URLs in a single response. If it does, they will be concatenated, probably resulting in an invalid URL. The user will be redirected to the invalid URL causing the user's browser to report an error.

Termination-Action

The Termination Action is only relevant for 802.1x authentication. If the TLV is missing from the response or if it is set to 0, when the user's session times out ExtremeWireless terminates the user's session and cleans up. The user must initiate the entire authentication process again. If the Termination-Action is set to 1 and 802.1x authentication was used, then the controller or AP will set a re-authentication timer as well as a session timer. In this case, the two timers have essentially the same setting. When the re-authentication timer starts, the AP to which the user is associated will initiate re-authentication of the device.

The Termination-Action is useful when integrating with NAC-like products. At least some of these products consign policy violators to a quarantine network for a period of time. At the end of that time the user is re-authenticated. If the user's device is compliant at the time of re-authentication, it is moved to another network. Because the AP initiates re-authentication, the user does not have to remember to do so.

RADIUS-based Administrative Login

ExtremeWireless implements three classes of administrative users, which are listed in Table 23: Types of ExtremeWireless Administrators on page 125 below. The *ExtremeWireless Convergence User Guide* explains how to create accounts of each type on the controller.

The controller can be configured to use RADIUS servers to authenticate administrators logging onto it. As shown in Figure 34: Configuring RADIUS-based Administrative Login on page 126, administrative login is configured on the RADIUS Authentication tab of the Login Management page of the Wireless Controller GUI. The controller can be configured to:

- Authenticate administrators only against the list of authorized credentials stored on the controller.
- Authenticate administrators only against a RADIUS server.
- Authenticate administrators against a remote RADIUS Server, falling back to the controller's own authentication database.
- Authenticate administrators against the controller's local authentication database, following back to RADIUS server authentication.

Table 23: Types of ExtremeWireless Administrators

Type of Administrator	Privileges
Full (Read/Write)	Full access to the controller's administration GUI and CLI. This type of administrator can configure any aspect of the controller or its APs, including managing other administrator accounts.
Read-Only	Read-only access to the controller. Cannot edit or create controller or AP configuration. Cannot see any passwords defined on the controller. Has access to a restricted CLI. Cannot create or manage other administrator accounts.
Guest	Can only use the Guest Portal administration GUI page. Has no CLI access. Can create and manage Guest Portal user accounts, including managing constraints on time of day and total amount of account usage time.

The controller configured in Figure 34: Configuring RADIUS-based Administrative Login on page 126 is set up to first try to authenticate a user against the RADIUS server and, if that fails, against the local authentication database. The controller can be configured to use more than one RADIUS server for administrative login authentications. The controller will use only one of the configured servers at any time, failing over to the next highest priority server when the initial server fails to respond. This is exactly the same approach used for 802.1x and captive portal authentication of end-users. This is covered in more detail in section RADIUS Server Redundancy on page 131.

From the RADIUS server point of view, authenticating administrators accessing a controller is just another instance of RADIUS password-based authentication. The server must be configured appropriately, including:

1 Having the shared secret that the controller uses to authenticate its messages to the RADIUS server.
2 Having the same RADIUS password-based authentication selected as the controller. The choices are:
   - PAP
   - CHAP
   - MSCHAP
   - MSCHAPv2
3 Having access to an authentication database containing the administrator's credentials.

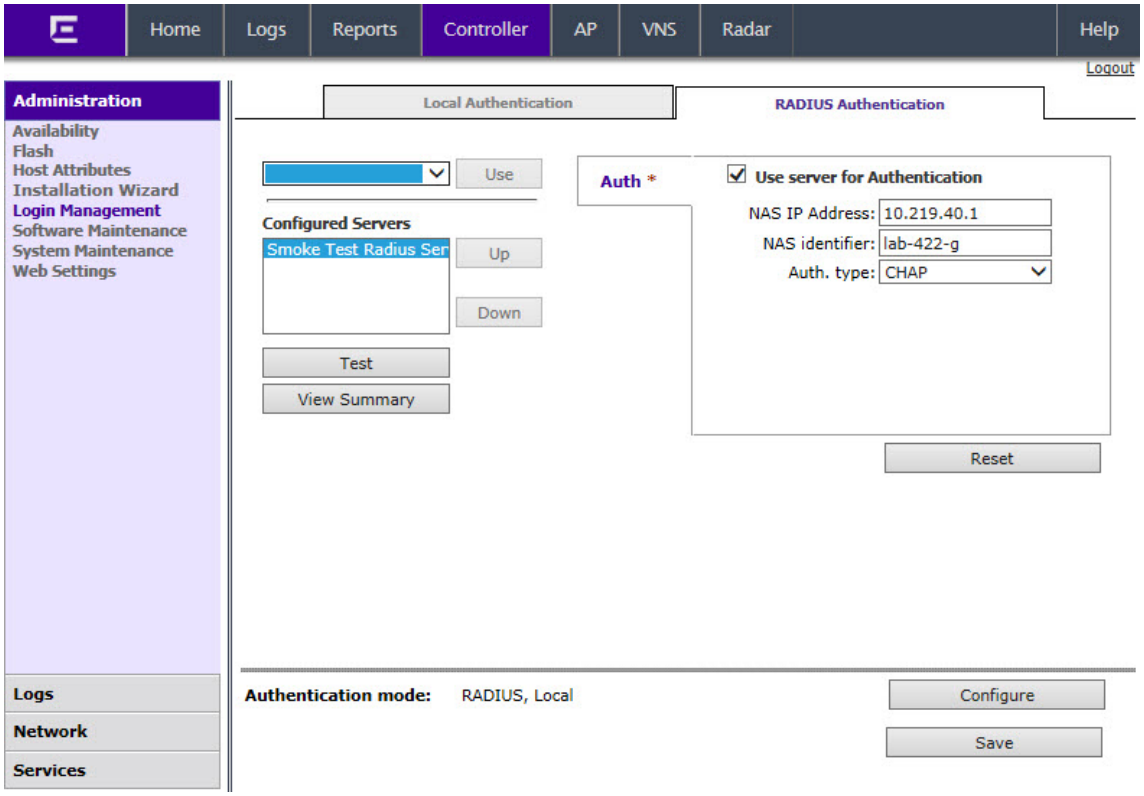


Figure 34: Configuring RADIUS-based Administrative Login

The RADIUS server is required to tell the controller what type of access to grant the administrator. The Service-Type attribute in the ACCESS-ACCEPT response conveys this information. The controller expects the Service-Type to have one of three values for an administrative authentication response as shown in the following table.

Table 24: Service Types for Administrative Login

Service-Type Value	RADIUS-Assigned Meaning	Meaning for Administrative Login
6	Administrative	Supplicant belongs to the fully privileged administrators group.
7	NAS Prompt	Supplicant belongs to the read-only privileged administrators group.
8	Authenticate Only	Supplicant is a guest portal account administrator.

Different RADIUS servers configure these items in different ways, so consult the RADIUS server's documentation for details.

Unsolicited/Asynchronous RADIUS Session Control

RFC 3576 defines a set of RADIUS messages that can be used to terminate a user's session or assign the user to a different role without terminating his or her session. The ExtremeWireless controller supports many of the functions defined in RFC 3576, in particular:

- Disconnect Message (DM) Request and Response

- Change of Authorization (CoA) Request and Response

The controller acts as the Dynamic Authorization Service (DAS) server, listening for DM and CoA Requests and sending the responses. RADIUS servers and other types of servers that send DM and CoA requests are referred to as DAS clients.

This part of the RADIUS protocol is pretty simple. To disconnect a user, send a DM request that identifies the user's session to the NAS (controller) managing that user's session. The server terminates the user's session, possibly generating a RADIUS accounting record, and replies to the requester indicating whether the disconnect operation was successful. Similarly, to change the role of a user or to set the user's session to "not authenticated," send a CoA message to the NAS (controller) for the target user. The message must identify the target user and the new policy settings to apply. The NAS applies the changes to the user's session and sends a CoA response to the requester to indicate that the changes were made.

RFC 3576 defines an "Authorize Only" mode for processing DM and CoA requests. In essence, a request with a Service-Type attribute set to "Authorize Only" triggers the receiver to send an Access-Request message to the configured RADIUS server. The RADIUS server then "dynamically" changes the authentication state or role by replying with an ACCESS-REJECT or an ACCESS-ACCEPT message containing a newly assigned role. ExtremeWireless does not support the "Authorize Only" mode for DM and CoA requests.

Controller Configuration

The servers sending DM and CoA messages need not be the servers used by WLAN Services for authentication. The server sending a DM or CoA request for a station does not have to be the RADIUS server that was used to authenticate the station. In fact the DAS client (the sender of DM and CoA requests) does not even have to be a RADIUS server, so long as it speaks the RFC 3576 piece of the RADIUS protocol.

DM and CoA requests are only accepted from servers listed in the controller's global list of RADIUS servers. The list is managed on the **Authentication** page of the VNS Configuration GUI, as shown in Figure 35: The Global RADIUS Servers List Includes DAS Clients on page 128 below. The DAS clients only need to be in this list and do not need to be assigned to any WLAN Services. Servers in the RADIUS server list share a common secret with the controller, which helps the controller to authenticate the source of the DM or CoA request. The controller drops requests that come from servers not in its RADIUS server list; it also drops requests from servers in the list if the Authenticator field in the message does not match the value computed by the controller.

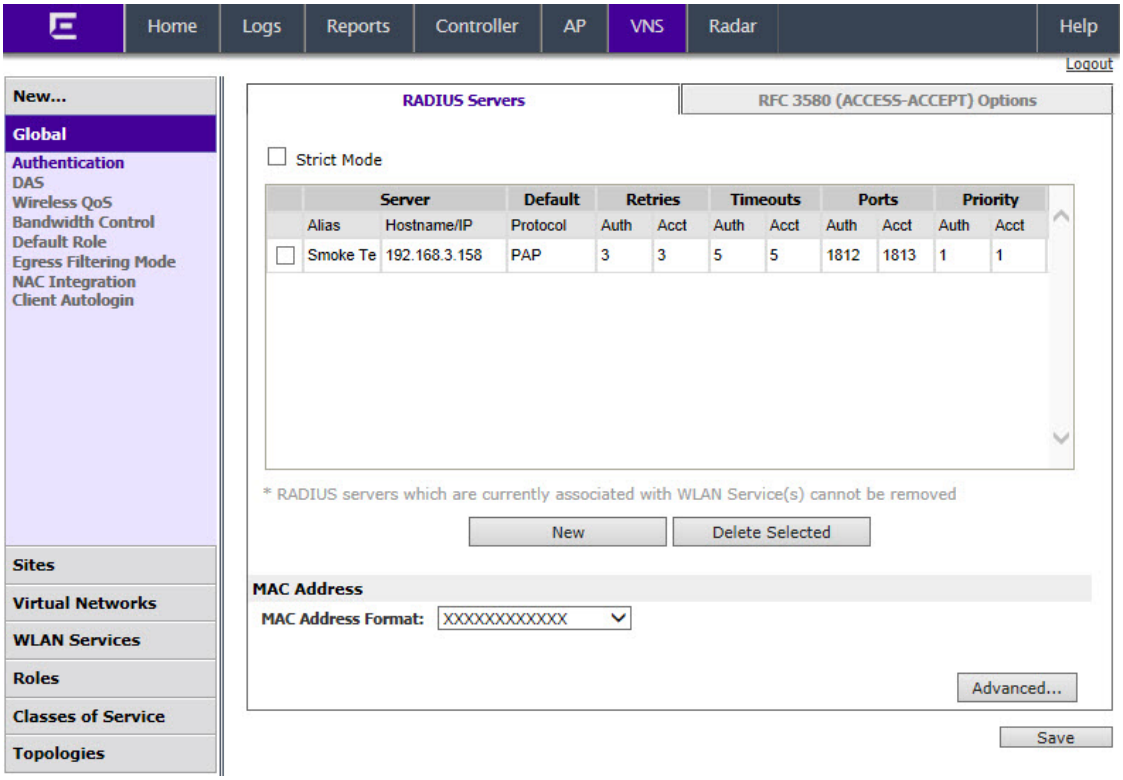


Figure 35: The Global RADIUS Servers List Includes DAS Clients

The controller's DAS server itself has just two configurable items:

- The UDP port number that the server listens on. All DAS requests must be sent to the controller at this port regardless of which WLAN Service the target user is on. The default for the port is 3799, which is the port specified for DAS in RFC 3576.
- The replay interval. RFC3576 defines basic protection against replay attacks. The DAS server will check the Event-Timestamp attribute against its own clock. The replay interval is the maximum absolute difference that will be tolerated between the DAS server's clock reading and the Event-Timestamp value in the request. If the interval between when the message appears to have been sent and when the message is received is too big, the DAS server quietly discards the request. The default value for the replay interval is 300 seconds, as recommended by RFC 3576. Replay protection can be turned off by setting the replay interval to 0.

Note

Replay protection means that the controller and DAS client need to have synchronized clocks. One way to achieve this is to configure the controller and DAS client to synchronize clocks with an NTP server.

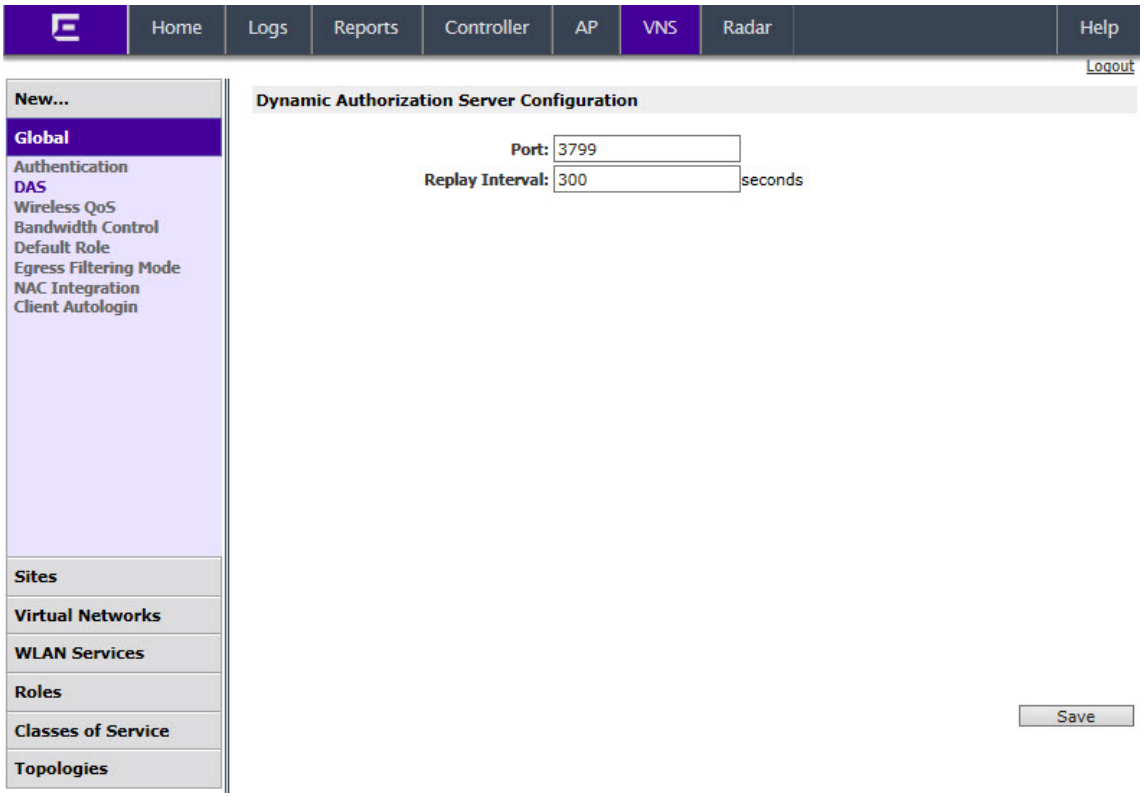


Figure 36: Dynamic Authorization Server Configuration

Terminating a Session

To terminate a session, send a Disconnect-Request RADIUS message to the controller. If the request is valid, the controller will send back a Disconnect-Response message indicating whether the disconnect request succeeded.

The controller expects to receive a Calling-Station-Id attribute in the Disconnect-Request. The value of the Calling-Station-Id is an octet string containing the MAC address of the device that is to have its session terminated. This is the only attribute the controller accepts as an identifier for the session to be terminated. The MAC address can be in any of the following formats:

- `xx:xx:xx:xx:xx:xx`
- `xx-xx-xx-xx-xx-xx`
- `xx.xx.xx.xx.xx.xx`
- `xxxx.xxxx.xxxx`
- `xxxxxxxxxxxx`

The hex digits ‘A’, ‘B’, ‘C’, ‘C’, ‘E’ and ‘F’ can be in either upper or lower case.

The controller requires the Disconnect-Request to contain an Event-Timestamp RADIUS attribute. If it does not, then the controller silently discards the request. The Event-Timestamp is specified in RFC 2869. Its value is the number of seconds since January 1, 1970 00:00 UTC.

The standard requires that the request contain a one-octet identifier. The identifier is copied to the response to simplify matching of responses to requests in the DAS client.

If an Authenticator attribute is included in the request, it must be valid. Note that the Authenticator is calculated the same way as the RADIUS accounting message Authenticator, as specified in RFC 2866.

The request may contain a standard RADIUS User-name attribute, which should be set to the user identifier for the user using the device identified by the Calling-Station-ID field.

The standard requires the controller to silently discard requests containing certain types of errors (such as messages missing the Event-Timestamp attribute). ExtremeWireless implements this aspect of the standard.

If the controller is allowed to reply, the response message will contain:

- The value in the Identifier field of the RADIUS request.
- An Event-Timestamp for the time at which the response is composed.
- A response authenticator, but only if the corresponding request had the request authenticator attribute.
- The Code field which will contain Disconnect-ACK if the request succeeded or a Disconnect-NAK if it failed.
- An Error-Cause attribute, but only if the message is a Disconnect-NAK. The error cause provides some information about why the request failed.

Changing a Station's Role or Authentication State

Asynchronously changing a station's default VLAN, role, or authentication status involves sending to the controller a CoA-Request (a RADIUS Request message with the Code field set to 43 – CoA-Request) and waiting to receive the response. If the request is valid, the controller will send back a RADIUS response indicating whether the CoA operation completed successfully.

The controller expects to receive the following attributes in a CoA request:

- Request Identifier octet. This is copied to the response message if it is sent.
- Event-Timestamp. This is used the same way for both DM and CoA requests.
- Calling-Station-Id. This contains the MAC address of the device that is to have its policy changed. The same MAC address formats can be used in the CoA request as in the Disconnect-Request.

The controller will accept any of the following optional attributes in a CoA request:

- Message authenticator. If present this must be valid or the request is discarded.
- Filter-Id. This is the same attribute that can be present in an ACCESS-ACCEPT response. The controller will accept either the standard Filter-Id format or the Extreme Networks decorated Filter-Id format.
- Tunnel-Private-Group-ID. This is the same attribute that can appear in an ACCESS-ACCEPT response. However, it can only contain a VLAN ID when transmitted in a CoA request.
- Tunnel-Type and Tunnel-Medium. These two attributes must be present when Tunnel-Private-Group-ID is present in the CoA request. Tunnel-Type must be VLAN (integer 13) and Tunnel-Medium must be 802 (integer 6). They must precede Tunnel-Private-Group-ID in the CoA request.
- Login-LAT-Port. This is the same attribute that can appear in the RADIUS ACCESS-ACCEPT response. It is used in the same way. An authenticated user can be set back to unauthenticated state

without terminating his session by sending a CoA-Request containing a Login-LAT-Port set to 0. This can be done at any time during the user's session.

- User-Name. This is the same attribute that appears in ACCESS-REQUEST messages.

These attributes were covered earlier in the chapter.

As was the case for Disconnect-Requests, there are scenarios in which the standard prohibits the controller from replying to the CoA-Request. If the controller is allowed to reply, the response will contain:

- The value in the Identifier field of the RADIUS request.
- An Event-Timestamp for the time at which the response is composed.
- A response authenticator, but only if the corresponding request had the request authenticator attribute.
- The Code field, which will contain Disconnect-ACK if the request succeeded or a Disconnect-NAK if it failed.
- An Error-Cause attribute, but only if the message is a Disconnect-NAK. The error cause provides some information about why the request failed.

Please consult RFC 3576 and the documentation for your RADIUS server for more details about using the Disconnect-Request and CoA-Request messages.

Note

The CoA-ACK, CoA-NAK, DM-ACK and DM-NAK contain the Identifier field of the corresponding request message. They do not contain any identifiers for the user, device, or session the action was performed on. Be sure to keep track of which request identifier applies to which user session.

RADIUS Server Redundancy

ExtremeWireless supports up to three RADIUS servers for 802.1x or Captive Portal authentication, up to three RADIUS servers for MAC-based authorization and up to three servers for RADIUS accounting. This means that each WLAN Service can use up to nine different RADIUS servers when RADIUS accounting and two types of authentication (MAC-based and one other one) are enabled. Obviously this is not mandatory or common. A single RADIUS server could be used for all three functions.

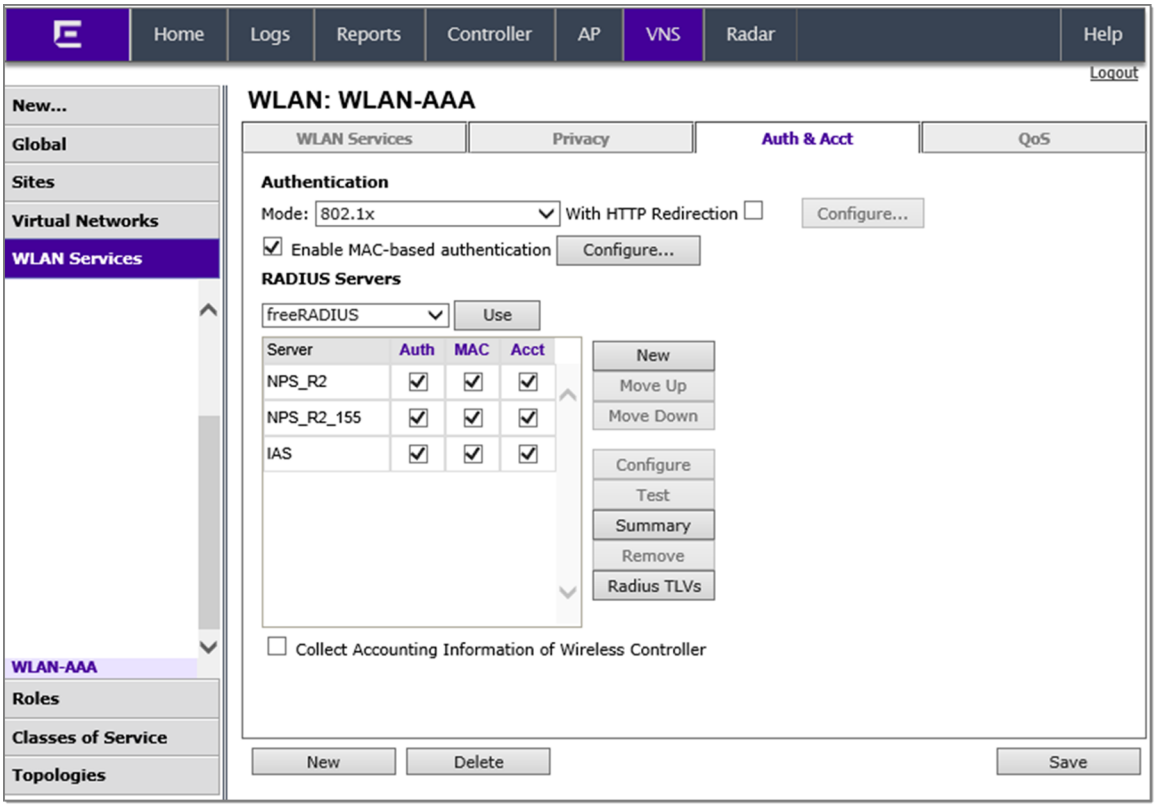


Figure 37: RADIUS Server Redundancy: Up to three usage types per server and up to three servers per usage type.

The controller uses additional RADIUS servers differently, depending on whether it is using the server for accounting or authentication and authorization. The controller sends copies of each RADIUS accounting message to each RADIUS server the WLAN Service is configured to use. This allows maintaining truly independent and redundant copies of accounting records to mitigate RADIUS accounting server failures.

The controller only uses one RADIUS server per authentication type (MAC-based authorization, 802.1x authentication, captive portal authentication) per WLAN Service at any one time. When the WLAN Service initializes it uses the highest priority server for authentication and authorization. When that server stops responding, ExtremeWireless switches to the RADIUS server with the next highest priority. If ExtremeWireless is using the lowest priority RADIUS server and it stops responding, the controller or AP will switch back to using the highest priority server.

The highest priority RADIUS server for each WLAN Service can be different, permitting a simple kind of load balancing across different RADIUS authentication servers.

For a WLAN Service the highest priority RADIUS server is the one nearest the top of the WLAN Service's "Used RADIUS servers" list with a check mark in the relevant column. In Figure 37: RADIUS Server Redundancy: Up to three usage types per server and up to three servers per usage type. on page 132, three RADIUS servers have been configured for use with MAC-based authorization. Server 192-168-192-110 has the highest priority for MAC-based authorization as it is nearest the top of the list. Server 192-168-192-192 is second highest priority and server 192-168-192-21 has the lowest priority for MAC-based authorization. Server 192-168-192-192 has the highest priority for Captive Portal authentication.

The relative priority of the servers can be adjusted by highlighting the server in the list of used servers, then clicking Move Up or Move Down until the server appears in the correct place in the list.

7 Facility Reference—RADIUS Accounting

Overview

Session accounting refers to creating and retaining records describing critical attributes of network users' sessions. ExtremeWireless implements two session accounting facilities:

1 RADIUS Accounting-
2 Call Detail Record (CDR) Accounting

Both types of accounting are optional and are disabled by default. Accounting functions can be enabled on a "per-WLAN Service" basis, meaning that some WLAN Services on a controller record accounting information about their users while others on the controller do not.

RADIUS accounting and CDR accounting collect the same information. The main difference between the two facilities is that CDR accounting records accounting details to files stored on the controller, while RADIUS accounting uses the protocol defined in RFC 2866 to transmit accounting details to one or more remote RADIUS servers.

Accounting functions are available only for WLAN Services that use some form of MAC-based authorization, 802.1x authentication or captive portal authentication. Only CDR accounting is available for WLANs using Guest Portal and Guest Splash authentication. This is because those types of authentication do not use RADIUS authentication. Consequently, some of the identifiers required for RADIUS accounting records are not available.

RADIUS accounting is the preferred method for collecting accounting information. Since CDRs are stored on the controller and the controller has limited persistent storage, CDR collection is only appropriate for WLAN Services that will have a low number of sessions per day. CDR files are kept only as long as space is available for them and the space is not needed for newer CDR records. Older CDR files are deleted automatically as needed. If CDRs are used to collect data about WLANs hosting large numbers of sessions there is a risk that some accounting information will be lost.

CDR records can be accessed only by running a backup task on the controller. The backup task can be scheduled to run periodically at a convenient time. The task can be configured to send the backup directly to an FTP server. Consult the *Extreme Networks User Guide and Command Line Interface Guide* for configuration details.

RADIUS accounting records are sent off the controller as soon as they are generated. RADIUS accounting records can be sent to multiple servers, providing protection against one of the servers failing.

CDR accounting and RADIUS accounting are enabled on the **Auth & Acct** tab of the WLAN Service configuration page. CDR collection is enabled by checking off the option labeled **Collect Accounting Information of the Wireless Controller**. The option will not be available when the WLAN Service's authentication mode is "Disabled". The *ExtremeWireless Convergence User Guide* describes configuring CDR collection and the format of CDR records in more detail.

Enabling RADIUS accounting is much like enabling RADIUS authentication. This topic is covered in Facility Reference—RADIUS Authentication and Authorization on page 105,which also describes how the controller uses multiple RADIUS accounting servers assigned to a single WLAN Service to enable reliable accounting record collection.

Note

Only the ExtremeWireless Controller acts as a RADIUS accounting client. When a site-based AP authenticates a user through RADIUS it sends the details to the controller. The controller will send accounting records to one or more RADIUS servers if the WLAN Service the user is accessing is configured for RADIUS accounting.

RADIUS Accounting Attributes Sent in the Start Accounting-Request

Table 25: RADIUS Attributes Sent in the Start Accounting-Request Message on page 136 below lists the attributes that can appear in a RADIUS Accounting Request message sent by the controller at the start of a user's session. This record is sent after authentication completes. The same information is recorded in CDRs when that facility is enabled.

If MAC-based authorization is enabled and if RADIUS accounting for MAC-based authorization is enabled, then the accounting start record is sent as soon as MAC-based authorization completes. MAC-based authorization can be used in conjunction with captive portal authentication. If RADIUS accounting is enabled but not for MAC-based authorization, the accounting start record is sent after captive portal authentication completes successfully.

Table 25: RADIUS Attributes Sent in the Start Accounting-Request Message on page 136 also lists the attributes that the controller can include in a RADIUS ACCOUNTING-REQUEST for an accounting start record. Not all of the attributes are necessarily included in all accounting start records.

Table 25: RADIUS Attributes Sent in the Start Accounting-Request Message

Attribute Name	Attribute Value	Notes
Acct-Authentic	A two-octet sequence interpreted as an unsigned integer in the range 1-6.	The value indicates how the user was authenticated. In the case where a user must undergo multiple authentications to get full network access, the value of this attribute indicates the type of authentication that triggered the accounting start record. ExtremeWireless uses this field in a non-standard compliant way. The possible values ExtremeWireless can send are: 1 – Indicates 802.1x authentication was used. 2 – Indicates that internal captive portal authentication was used. 3 – Indicates that external captive portal authentication was used. 4 – Indicates that Guest Portal authentication was used. This value can only appear in CDRs since RADIUS accounting is disabled in this case. 5 – Indicates that Guest Splash Screen authentication was used. This value can only appear in CDRs since RADIUS accounting is disabled in this case. 6 – Indicates that MAC-based authorization was used. This can only happen when RADIUS Accounting is enabled for MAC-based authorization.
Acct-Delay-Time	A four-octet sequence interpreted as the number of seconds the controller has been trying to send this record.	Computed as the difference between the official start time of the session and the second at which this message is composed.
Acct-Interim-Interval	A four-octet sequence interpreted as the number of seconds between interim accounting reports for this user.	This is configured per RADIUS server. The default is 1800 seconds (30 minutes).
Acct-Session-Id	An octet sequence that is used to identify all the accounting records for a particular user-session.	
Acct-Status-Type	A four-octet sequence interpreted as an integer that identifies the type of accounting record transmitted	Will be set to 1 to indicate an accounting start record.
Called-Station-Id	Octet string containing a BSSID / MAC address or a text sequence.	By default, this TLV contains the BSSID of the AP the station is associated to at the moment the start record is sent. If the WLAN Service is configured to send Zone labels in RADIUS Access-Requests, it will also send the Zone label instead of the BSSID in the Called-Station-Id.
Calling-Station-Id	An octet string containing the ASCII encoding of the MAC address of the device to be authenticated.	This is always reported as a colon-separated MAC address. In other words, the MAC address format selection on the global VNS authentication configuration GUI applies to ACCESS-REQUEST messages but not to ACCOUNTING-REQUEST messages.

Table 25: RADIUS Attributes Sent in the Start Accounting-Request Message (continued)

Attribute Name	Attribute Value	Notes
Chargeable-User-Identity	A string value that identifies a particular user. The RADIUS server determines the format and content of the string	Identifies a user for any roaming transactions that take place outside of the network.
Class	An octet string at least 1 octet in length	This attribute contains whatever class string was assigned to the user in the ACCESS-ACCEPT response from the RADIUS authentication server. RFC 2865 places few restrictions on the contents of the string. The controller does not interpret the attribute. It simply forwards it in accounting records for the user.
Connect-Info	An octet string containing either “802.11a” or “802.11b/g.”	The string identifies the radio to which the user is associated at the time the start record is created. The string “802.11a” means the user is connected to the 5 GHz radio. The string “802.11b/g” means the user is connected to the 2.4 GHz radio. These strings do not indicate the actual protocol the user is transmitting. For example, if the user is transmitting 802.11n on the 5 GHz band, this attribute will say “802.11a.”
Filter-Id	String	The name of the role that is being applied to the subject user’s traffic
Framed-IP-Address	An IP address in the form of a sequence of four unsigned octets.	The IP address assigned to the station at the moment the session start record is created. Interim accounting records will be sent if the IP address changes during the user’s session. The user may not have an IP address at the time the start record is sent. This can happen if MAC-based authorization or 802.1x is enabled on the WLAN Service the user is accessing.
Login-LAT-Group	Obsolete	ExtremeWireless used this attribute to identify the “Child VNS” to which the user’s session was assigned. The concept of “Child VNS” is no longer supported.
NAS-Identifier	An octet string representing a name.	This is either a string configured by the administrator or the name of the VNS to which the user is authenticating.
NAS-IP-Address	An IP address in the form of a sequence of four unsigned octets.	The administrator can configure an arbitrary IPv4 address for the NAS-IP-Address. In joint authentication, if the NAS IP is not explicitly configured, then if the controller has an IP address on the default topology of the default non-authenticated VNS role, that address will be used as the NAS-IP address.
NAS-Port-Type	A single octet with a decimal value of 18.	The number 18 represents the “Wireless-Other” NAS-Port-Type.
Operator-Name	A value of 0x34 - 0-xFE	Identifies the owner of an access network using a unique ID.
Session-Timeout	A positive integer representing the maximum length of the authenticated user’s session, expressed in seconds.	

Table 25: RADIUS Attributes Sent in the Start Accounting-Request Message (continued)

Attribute Name	Attribute Value	Notes
Siemens-AP-Name	An octet sequence containing the name of the AP that is authenticating the user/device.	This is the name of the AP as configured in the AP section of the controller GUI. By default it is not sent.
Siemens-AP-Serial	An octet sequence containing the unique serial number of the AP that is authenticating the user/device.	The serial number of the AP is visible in the controller GUI. It is set at the factory and can't be altered. By default this is not sent. The Siemens-AP-Serial attribute is only sent when the Siemens-AP-Name is also sent.
Siemens-SSID	An octet sequence containing the SSID of the WLAN service to which the station associated.	
Siemens-VNS-Name	An octet sequence containing the name of the VNS to which the station is authenticating.	This is the name given to the VNS in the VNS configuration section of the controller's Virtual Networks GUI page. By default this is not sent.
User-Name	String	Identifier the user authenticated with. In some cases this may not be available at the start of the session. In particular, if MAC-based authorization is enabled and accounting records are enabled for MAC-based authorization, then it is very likely the user will not have authenticated with a user name by the time the start record is sent.

Note

Siemens VSAs are sent in start, interim, and stop ACCOUNTING-REQUEST messages only when they are configured to be sent in the ACCESS-REQUEST message. Only the specific VSAs included in the ACCESS-REQUEST are included in the ACCOUNTING-REQUEST messages. The Siemens-AP-Serial number VSA is sent only when the Siemens-AP-Name VSA is configured to be included in ACCESS-REQUEST messages.

Aside: Enabling Reporting Fast Failover Events as Interim Accounting Records

A user's home controller normally is the one that manages the first AP he associated to during his or her session. However, in a special case it is possible for the user's home controller to change mid-session. This can happen when the user's session is managed by an availability pair that has fast failover enabled. In this case, if a user on a bridged at AP or bridged at controller topology roams to an AP managed by the availability partner, then that controller becomes the user's home controller. If the user's session terminates while his home is the availability partner, that controller will send the Stop Accounting-Request to the RADIUS server. So in this case, it is possible for an Accounting server to receive the Start Accounting-Request from a controller and interim and stop Accounting Requests for the same user from a different controller.

The controller can be configured to send an interim ACCOUNTING-REQUEST when it takes over a user's session from its availability partner. When the option is enabled, the controller sends the interim Accounting-Request to the RADIUS accounting server at the moment it takes over the user's session. This allows the RADIUS server to keep track of which controller currently hosts the user. This is useful

information especially if the server is using unsolicited/asynchronous session control. Only the controller currently hosting a user's session will process session control requests for that user.

Figure 38: Enabling Interim Accounting Records for Fast Failover Events on page 140 shows the dialog box used to enable sending interim accounting records when a user's session moves to the availability partner. The attribute is configured on a per-WLAN-Service-per-RADIUS-server basis. This means that a single WLAN Service could send the interim "Session Moved" report to some of the accounting servers it uses but not others. It also means that the event can be sent for some of a controller's WLAN Services and not others.

To get to the dialog box in Figure 38: Enabling Interim Accounting Records for Fast Failover Events on page 140:

1 Open the WLAN Service configuration GUI for the WLAN Service.
2 Open the WLAN Service's **Auth & Acct** tab.
3 Select at least one RADIUS server from the list.
4 Click **Configure**.
5 Once the dialog box opens, select the row labeled "Acct". If one does not appear, then the server is not configured as a RADIUS accounting server on the controller.

   The "Send Interim Accounting Records for" options appear. The dialog box will now look like it does in Figure 38: Enabling Interim Accounting Records for Fast Failover Events on page 140.
6 Enable the setting by selecting the **Fast Failover Events** option and clicking **OK**.
7 Click **Save** to save the changes.

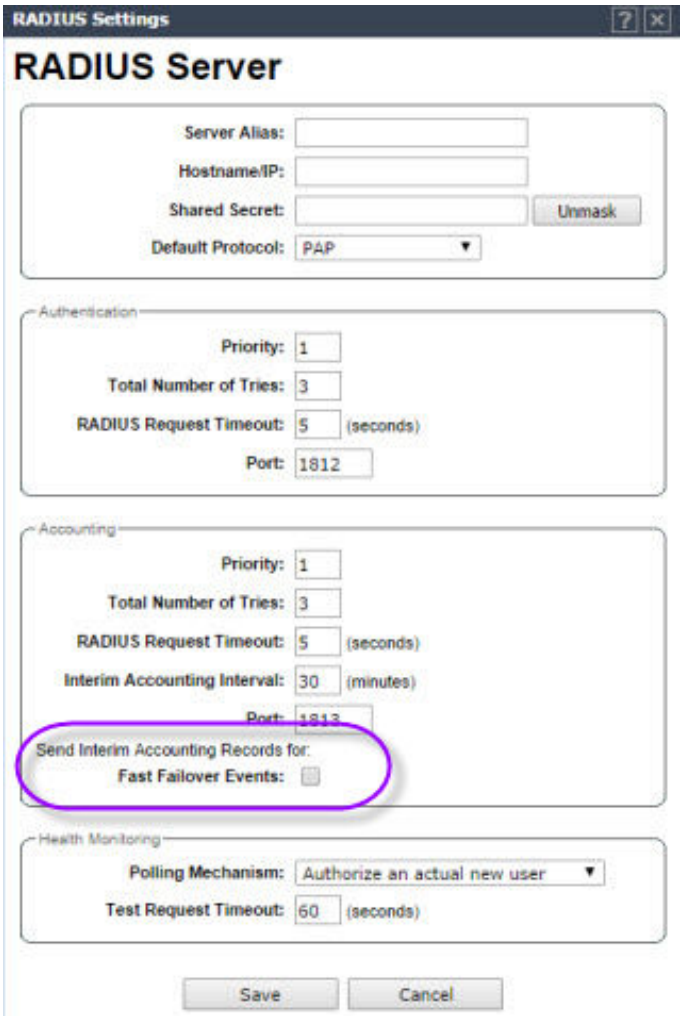


Figure 38: Enabling Interim Accounting Records for Fast Failover Events

RADIUS Accounting Attributes Sent in the Interim and Stop Accounting-Requests

Stop records are sent as soon as the user's session terminates. The reason for the termination is indicated in the Accounting-Request message sent from the controller.

Interim records are sent periodically (by default every 30 minutes) while the subject user has a network session. Interim records are also sent for some significant session events such as:

- The subject user changes his or her IP address while the session is in progress.
- The user's session is moved to the availability partner and the option to report this event as an interim update is enabled, as described in the preceding section.

The table below lists the attributes that can appear in a RADIUS Accounting Request message sent by the controller in interim reports and at the end of a user's session.

Table 26: RADIUS Attributes Sent in Interim & Stop Accounting Request Messages

Attribute Name	Attribute Value	Notes
Acct-Authentic	A two-octet sequence interpreted as an unsigned integer in the range 1-6.	The value indicates how the user was authenticated. In the case where a user must undergo multiple authentications to get full network access, the value of this attribute indicates the type of authentication that triggered the session start record. ExtremeWireless uses this field in a non-standard compliant way. The possible values for this field are: 1 – Indicates 802.1x authentication was used. 2 – Indicates that internal captive portal authentication was used. 3 – Indicates that external captive portal authentication was used. 4 – Indicates that Guest Portal authentication was used. This value can only appear in CDRs since RADIUS accounting is disabled in this case. 5 – Indicates that Guest Splash Screen authentication was used. This value can only appear in CDRs since RADIUS accounting is disabled in this case. 6 – Indicates that MAC-based authorization was used. This can only happen when RADIUS Accounting is enabled for MAC-based authorization.
Acct-Delay-Time	A four-octet sequence interpreted as the number of seconds the controller has been trying to send this record.	
Acct-Input-Octets	The number of octets/ bytes received from this user so far during his or her session.	
Acct-Input-Packets	The number of packets received from this user so far during his or her session.	
Acct-Interim-Interval	A four octet sequence interpreted as the number of seconds between interim accounting reports for this user.	This is configured per RADIUS server. The default is 1800 seconds (30 minutes).
Acct-Output-Octets	The number of octets/ bytes sent to this user so far during his or her session.	
Acct-Output-Packets	The number of packets sent to this user so far during his or her session.	
Acct-Session-Id	An octet sequence that is used to identify all the accounting records for a particular user-session.	

Table 26: RADIUS Attributes Sent in Interim & Stop Accounting Request Messages (continued)

Attribute Name	Attribute Value	Notes
Acct-Session-Time	A four-octet sequence interpreted as an unsigned integer containing the total number of seconds of service the user has received so far in his or her session.	
Acct-Status-Type	A four-octet sequence interpreted as an integer that identifies the type of accounting record transmitted.	Will be set to 2 for a Stop record or 3 for an Interim-Update.
Acct-Terminate-Cause	A four-octet sequence interpreted as an unsigned integer that identifies the reason the session terminated.	Some of the more common termination codes returned by the controller are 1 – User Request – the internal captive portal provides a way for the user to cleanly log out of the network. The terminate cause will be set to 1 for any session that is terminated this way. 4 – Idle Timeout – user's session was inactive for at least the duration specified for his session 5 – Session Timeout – user's session reached its maximum configured duration. 6 – Admin Reset – the administrator uses the controller GUI, CLI or session control APIs to terminate the user's session before it timed out. 11 – NAS Reboot – indicates that the session was terminated due to a controlled shutdown of the controller.
Called-Station-Id	An octet string containing a BSSID / MAC address	By default this TLV contains the BSSID of the AP the station is associated to at the moment the start record is sent. If the WLAN Service is configured to send Zone labels in RADIUS Access-Requests, it will also send the Zone label instead of the BSSID in the Called-Station-Id.
Calling-Station-Id	An octet string containing the ASCII encoding of the MAC address of the device to be authenticated.	This is always reported as a colon-separated MAC address. In other words, the MAC address format selection on the global VNS authentication configuration GUI applies to ACCESS-REQUEST messages but not to ACCOUNTING-REQUEST messages.
Class	An octet string at least one octet in length	This attribute contains whatever class string was assigned to the user in the ACCESS-ACCEPT response from the RADIUS authentication server. RFC 2865 places few restrictions on the contents of the string. The controller does not interpret the attribute. It simply forwards it in accounting records for the user.

Table 26: RADIUS Attributes Sent in Interim & Stop Accounting Request Messages (continued)

Attribute Name	Attribute Value	Notes
Connect-Info	An octet string containing either “802.11a” or “802.11b/g.”	The string identifies the radio to which the user is associated at the time the interim or stop record is created. The string “802.11a” means the user is connected to the 5 GHz radio. The string “802.11b/g” means the user is connected to the 2.4 GHz radio. These strings do not indicate the actual protocol the user is transmitting. For example if the user is transmitting 802.11n on the 5 GHz band this attribute will say “802.11a.”
Filter-Id	String	The name of the role that is being applied to the subject user’s traffic.
Framed-IP-Address	An IP address in the form of a sequence of four unsigned octets.	The IP address assigned to the station at the moment the session interim or stop record is created.
Login-LAT-Group	Obsolete	ExtremeWireless used this attribute to identify the “Child VNS” to which the user’s session was assigned. The concept of “Child VNS” is no longer supported.
NAS-Identifier	An octet string representing a name.	This is either a string configured by the administrator or the name of the VNS to which the user is authenticating.
NAS-IP-Address	An IP address in the form of a sequence of four unsigned octets.	The administrator can configure an arbitrary IPv4 address for the NAS-IP-Address. In joint authentication, if the NAS IP is not explicitly configured then if the controller has an IP address on the default topology of the default non-authenticated VNS role, that address will be used as the NAS-IP address.
NAS-Port-Type	A single octet with a decimal value of 18.	The number 18 represents the “Wireless-Other” NAS-Port-Type.
Session-Timeout	A positive integer representing the maximum length of the authenticated user’s session, expressed in seconds.	
Siemens-AP-Name	An octet sequence containing the name of the AP that is authenticating the user/device.	This is the name of the AP as configured in the AP section of the controller GUI. By default it is not sent.
Siemens-AP-Serial	An octet sequence containing the unique serial number of the AP that is authenticating the user/device.	The serial number of the AP is visible in the controller GUI. It is set at the factory and can’t be altered. By default this is not sent.
Siemens-SSID	An octet sequence containing the SSID of the WLAN service to which the station associated.	

Table 26: RADIUS Attributes Sent in Interim & Stop Accounting Request Messages (continued)

Attribute Name	Attribute Value	Notes
Siemens-VNS-Name	An octet sequence containing the name of the VNS to which the station is authenticating.	This is the name given to the VNS in the VNS configuration section of the Virtual Networks module of the controller GUI. By default this is not sent.
User-Name	String	Identifier the user authenticated with. In some cases this may not be available at the start of the session. In particular if MAC-based authorization is enabled and accounting records are enabled for MAC-based authorization then it is very likely the user will not have authenticated with a user-name by the time the start record is sent.

8 Facility Reference—Batch Mode Station Location Publishing

Overview

This chapter describes how to configure and use the controller's ability to publish files of station location information. This feature is referred to as Batch Mode Station Location Publishing. Once the controller is configured for Location Services and configured to publish locations, it will post a file of location data to between one and five destinations periodically.

The controller continues to publish this information until:

- Batch mode location publishing is disabled.
- The location feature is disabled.
- The controller is shut down.

If batch mode location publishing is enabled then the controller will resume publishing results after it recovers from a restart or power outage.

The file of location data is formatted as XML. The file complies with the schema published in DeviceLocations.xsd on page 216.

This document assumes familiarity with Extreme NetSight OneView's Maps application. Refer to “Advanced Map Features” and “Create and Edit OneView Maps” in [1] for details.

This document also assumes familiarity with the Extreme Wireless Controller's location calculation feature. The controller's location calculation feature must be enabled in order to use the location publishing feature described in this technical note. Refer to “Configuring the Location Engine” in [2] and [3] for details.

Controller Location Calculations

The controller can generate location reports for:

- Stations associated to one of the controller’s managed APs or one of the controller’s availability partner’s APs.

- APs that represent a threat to the controller's wireless environment. These include non-managed APs advertising SSIDs or BSSIDs that belong to the controller and rogue APs. Radar WIDS-WIPS must be enabled to receive these location reports.
- MAC addresses of devices that the administrator is interested in. The administrator must manually enter these addresses. These addresses will be tracked whenever they are visible to a managed AP, even if the station owning the MAC address does not associate to one of the managed APs. This is useful for things like triggering an alert when a possibly stolen device shows up in the area.

In release 9.15, the controller generates two types of location calculations:

- CellId– RSS: This type of location is reported as a two tuple:
  - The identity of the AP that reports seeing the station.
  - An estimate of the distance between the AP and the located station. The distance is estimated from the strength of the station's transmissions as received at the AP (RSS).

    In effect, the location of the station is (most probably) within a circle centered on the location of the AP with a radius equal to the associated distance estimate.
- Triangulated Location: This type of location is reported as a four tuple:
  - The hierarchical name of the general location in which the station is (most probably) located. The hierarchical name is the hierarchical name for a floor plan for a location at which managed APs are deployed. The name is assigned to the general location/floor plan by the OneView Map editor.
  - An estimate of the distance that the station is from the location represented by the left-hand edge of the floor plan identified by the hierarchical name ('x').
  - An estimate of the distance that the station is from the location represented by the top edge of the floor plan identified by the hierarchical name ('y').
  - The probability that the station is at the location identified by the {hierarchical location name, x, y}.

    This location algorithm takes the RSS readings for the station from multiple APs and computes the probability of that pattern of RSS readings being received, given the construction represented by the floor plan and the location of the AP. The floor plan is divided into a rectangular grid. The probability calculation is made for each grid cell. The location file contains for each station the grid cell that has the highest probability of containing the station.

The batch location report file may contain both types of location report. A triangulated location is both more accurate and precise than a CellId-RSS location. The CellId-RSS is used when:

- The controller has reached its system limit on the maximum number of triangulated locations it can track.
- The controller has not received for the subject station RSS reports of sufficient strength from a sufficient number of APs to generate triangulated locations.

Configuring Publishing of Station Locations

Before station locations can be published, the controller must be configured to calculate station locations. This is a three-step process:

1 Enable the location service on the controller.
2 Enable "Location Batch Reporting" on the controller.

3 Configure one or more locations into OneView Maps.

Enabling Location Service on the Controller

Figure 39: Enabling the Location Engine on page 147 below illustrates the Main Location Engine configuration page, before the engine is enabled.

To reach this page:

1 Click **Radar** from the top menu.
2 Click **Location Engine** from-the left-hand sub-menu.
3 To enable the engine, select the **Location Engine** check box.

The page will now display more configuration options. Be sure to enable **Locate Active Sessions** as shown in Figure 40: Enabling Automatic Tracking of User Sessions on page 148.

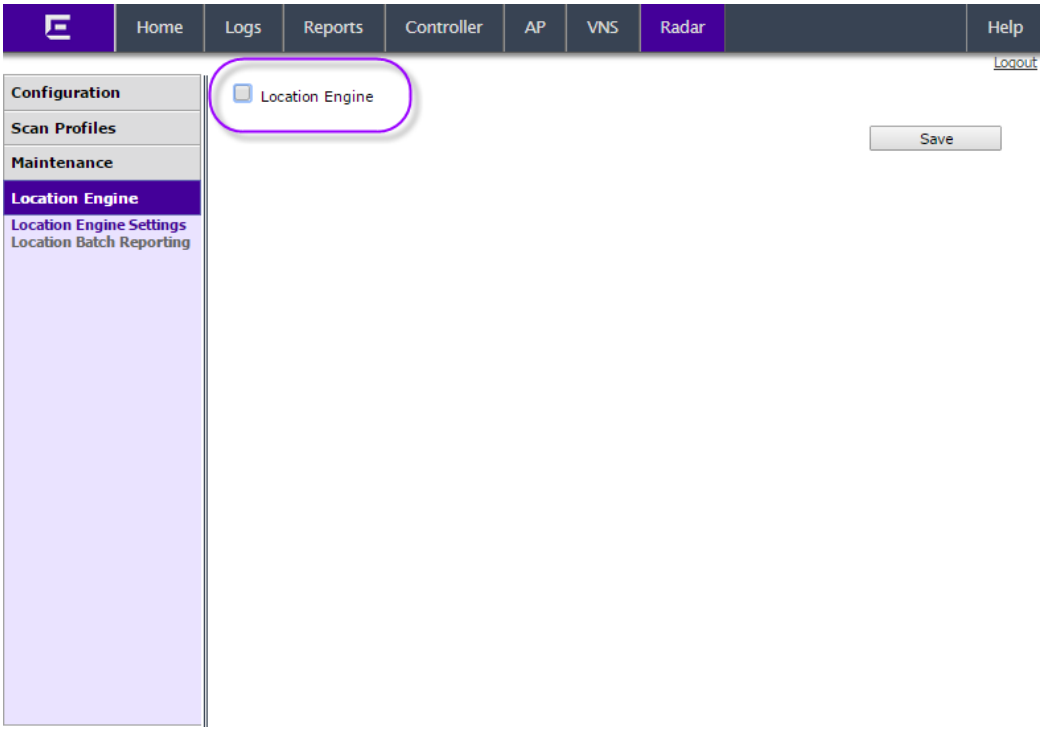


Figure 39: Enabling the Location Engine

The page contains other options that can be changed if appropriate. Refer to [2] for a description of those options.

[19] The purple rounded rectangle is a highlight added for this document and not present on the actual web page.

4 Click Save once the desired configuration changes have been made.

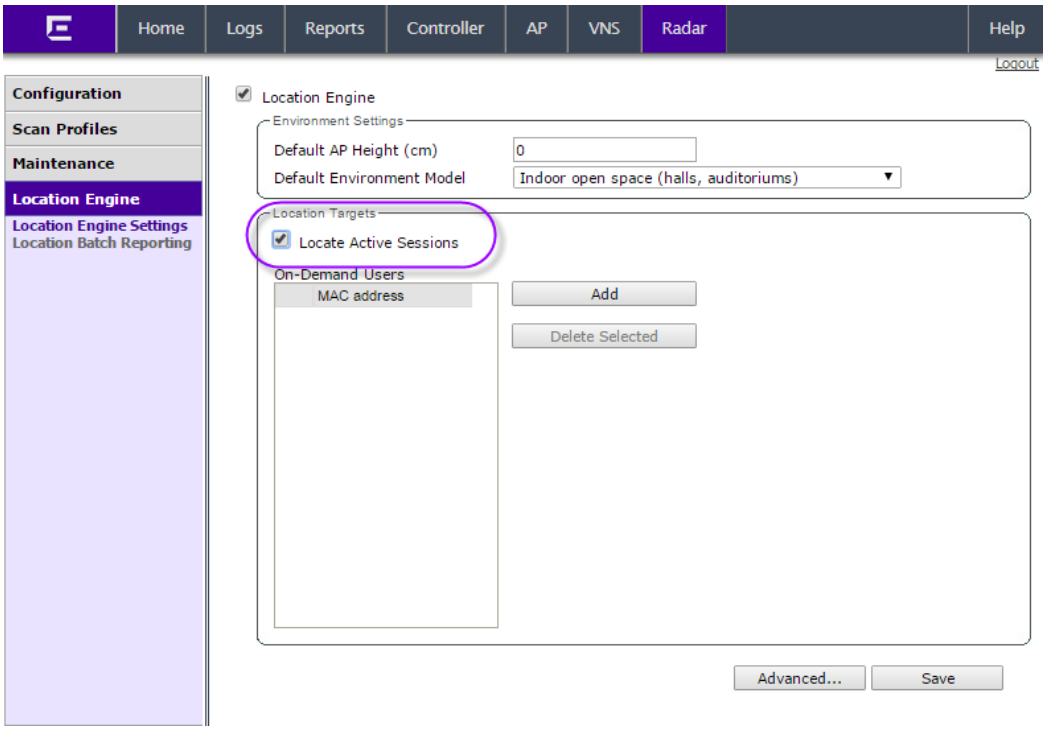


Figure 40: Enabling Automatic Tracking of User Sessions

Enabling Location Batch Reporting on the Controller

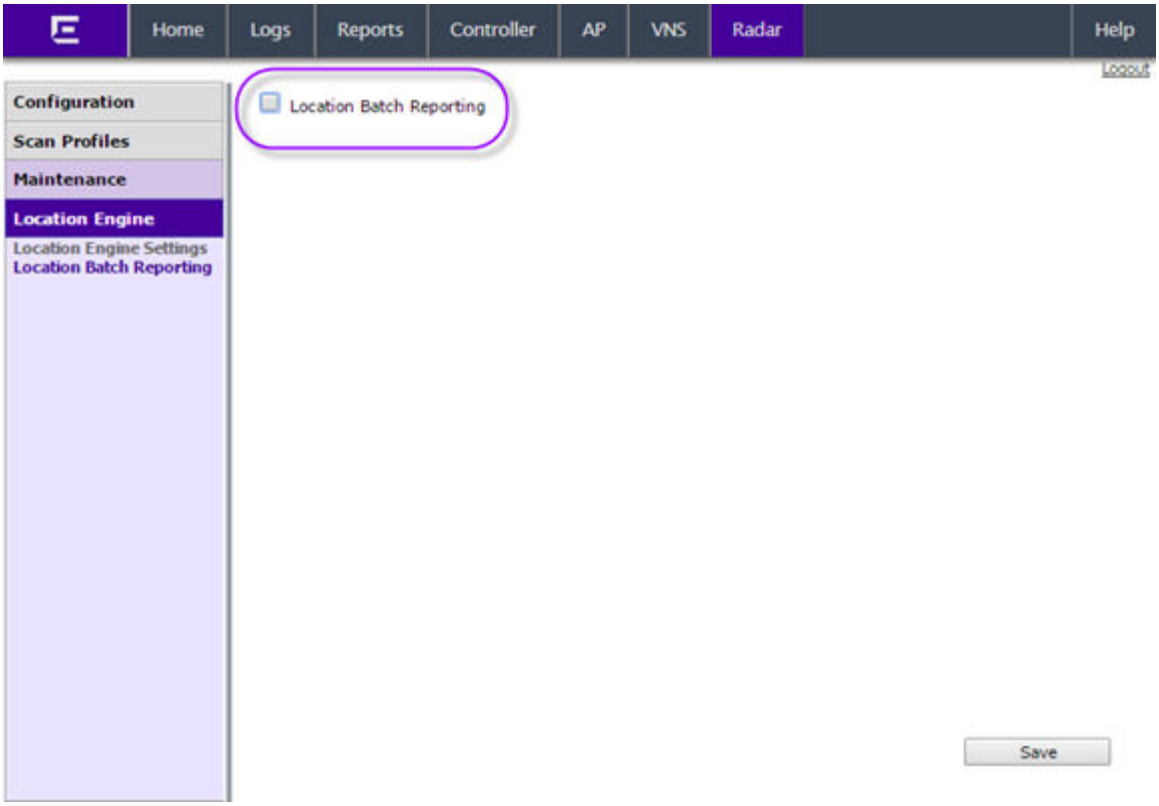
To enable location publishing:

1 Click Location Batch Reporting in the left-hand menu (see Figure 40: Enabling Automatic Tracking of User Sessions on page 148).

A page similar to the one shown in Figure 41: Enabling Location Batch Reporting on page 149 will appear.

2 Select the Location Batch Reporting checkbox.

Figure 41: Enabling Location Batch Reporting



3 Configure the following options:

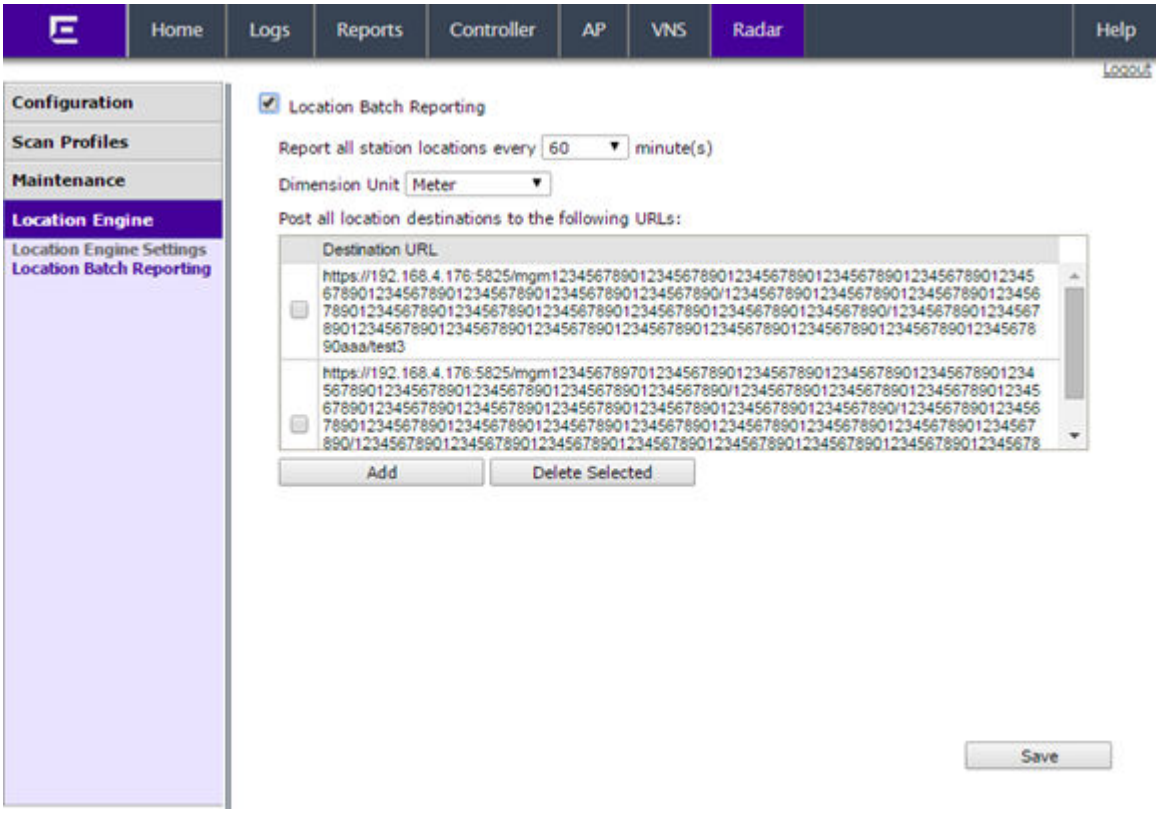
- Report all station locations every, which determines the interval between station location reports, in minutes. Choose from of the following in the drop-down list:
  - 1 minute
  - 2 minutes
  - 5 minutes
  - 10 minutes
  - 20 minutes
  - 30 minutes
  - 60 minutes
  - 120 minutes
  - 240 minutes
- A list of up to five different URLs to which the location report will be posted.

Note

The controller populates the location file with entries for the sessions it knows about at the moment it creates the report. It does not buffer location data for sessions that terminate before the controller starts generating the next report. Consequently, the list of station MAC addresses in a location file generated every four hours may not be the same as the list of MACs that used the network in the four-hour interval since the last report was generated.

4 To configure a destination click Add.

Figure 42: Location Batch Reporting Configuration Page, After Enabling the Feature



5 In the resulting dialog box, enter a valid URL into the Destination URL field and then click OK.

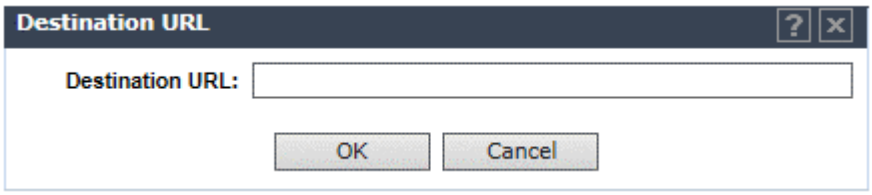


Figure 43: Adding a Destination for a Batch Location Report

6 Click Save to save the changes.

Note

You cannot enter more than five destinations for a copy of the location file. After the fifth destination has been added, the Add button is disabled.

If you no longer need one of the URLs, simply select the check box to the left of the URL in the Batch Reporting page's "Destination URL" list, and then click Delete Selected. Multiple URLs can be selected and deleted at once if desired.

Shortly after the list of destinations is saved, a location report will be generated and sent to each of the configured destinations. Reports will then be published to all listed destinations periodically, at the configured interval.

Receiving the Controller's Location Reports

As mentioned earlier, the controller will perform an HTTP PUT of the location file to each of the URLs configured on the controller. Each URL must point to a valid destination for an HTTP PUT operation.

Where the URL can point to depends on the web server and security environment in question. Some web servers can perform PUT operations directly to a directory on the server. In this case the URL configured on the controller can simply map to a directory that the web server has been configured to write to.

This is not the most secure way to handle a PUT. An alternative is to write a program that gets invoked when a file is PUT to the server. The program can decide where it wants to store the file and can verify that the file is of the type, content and size expected. Simple PHP Script to Accept an HTTP Put file on page 151 is an example of a PHP program that can receive a PUT location file and store it in the file system. The program is quite simple; it opens stdin and copies the input stream to a file that it has opened for output. The example is for a Linux file system but the same type of program works on Windows as well.

This script must be configured to run when the HTTP PUT verb is received. How this is done depends on the web server being used. On an Internet Information Server (IIS) the site must be configured to permit PUT operations; any script in the site can then receive and process a PUT.

The Apache web server must be told which script to run when a PUT verb is received. This could be done by creating a "Script" directive within the `<VirtualHost>` block that configures the specific website on the web server. Assuming the PHP file in Simple PHP Script to Accept an HTTP Put file on page 151 was called "locationcollector.php," the Apache directive would look like:

```
Script PUT /locationcollector.php
```

The files of location data can be quite large. It is also important to be sure that a file of location data is received and stored completely before the controller starts pushing the next batch location report file to the web server. If a slower web server is being used it is worthwhile keeping processing in the receiving program to a minimum. For example, the program could just receive the file and copy it to a back end file server with an appropriate unique name. A separate program could then parse the file to analyze it or to store individual records into a back end database.

Simple PHP Script to Accept an HTTP Put file

```
<?php
 /* PUT data comes in on the stdin stream */
  $putdata = fopen("php://input", "r");
/* Create a unique name for the output file */
  $filename = "./tmpLocn".time().".xml";
/* Open a file for writing. This overwrites any previously
   * copy of the file so be sure to process the previously
   * received copy before receiving this one.
   */
  $fp = fopen($filename, "w");
/* Read the data and write it to the output file
   * 1024 bytes at a time. This program limits the total number
   * of 1K blocks copied to 8192 (i.e. 8 Megabytes in total).
```

```
   * The limit depends on how big a file is expected and of course
   * on the available local storage as well.
   */
  $block_limit = 8192;
  $blocks_copied = 0;
 while ($blocks_copied < $block_limit && $data = fread($putdata, 1024)) {
    fwrite($fp, $data);
    $blocks_copied++;
  }
/* Close the streams */
  fclose($fp);
  fclose($putdata);
/* Post process the file here if desired. */
?>
```

Parsing the File of Location Reports

The location report uploaded by the controller takes the form of an XML document. The document conforms to the schema shown in DeviceLocations.xsd on page 216. This section describes the file format in more detail.

The code below is an example of a location report XML file. Note that the file clearly contains redundant information. Information like the size of the area represented by the floor plan is repeated as a convenience. This avoids the need to use complex Xpath queries when processing the file.

Example Device Locations File

```
<?xml version="1.0" encoding="UTF-8"?>
<DeviceLocations schemaVersion="1.0" fileGeneratedTime="2014-11-18T22:35:01Z"
    entries="5">
    <DeviceLocation probability="0.11" currentlyTracked="true"
        macAddress="20:B3:99:9C:5A:28" apMacAddress="20:B3:99:AE:C5:AA">
        <LocationDetails floorRefId="1"
            floorName="/World/Map99 - test2_created_108B">
            <Dimension unit="FEET_AND_INCHES" width="479'0" length="721'9" />
        </LocationDetails>
        <Coordinates unit="FEET_AND_INCHES" y="341'2" x="150'11" />
        <Details lastLocatedTime="2014-11-18T22:33:33Z" />
    </DeviceLocation>
    <DeviceLocation probability="0.11" currentlyTracked="true"
        macAddress="20:B3:99:E5:BA:A0" apMacAddress="20:B3:99:AE:C5:AA">
        <LocationDetails floorRefId="1"
            floorName="/World/Map99 - test2_created_108B">
            <Dimension unit="FEET_AND_INCHES" width="479'0" length="721'9" />
        </LocationDetails>
        <Coordinates unit="FEET_AND_INCHES" y="328'1" x="137'9" />
        <Details lastLocatedTime="2014-11-18T22:33:33Z" />
    </DeviceLocation>
    <DeviceLocation probability="0.11" currentlyTracked="true"
        macAddress="00:1F:3B:10:9B:DD" bssid="20:B3:99:E1:78:92" ssid="L203-
C25-bac-AAA-wpa2"
        apMacAddress="20:B3:99:E1:7B:B0" channelNumber="161">
        <LocationDetails floorRefId="1"
            floorName="/World/Map99 - test2_created_108B">
            <Dimension unit="FEET_AND_INCHES" width="479'0" length="721'9" />
```

```
        </LocationDetails>
        <Coordinates unit="FEET_AND_INCHES" y="249'4" x="124'8" />
        <Details lastLocatedTime="2014-11-18T22:34:10Z" />
    </DeviceLocation>
    <DeviceLocation probability="0.02" currentlyTracked="true"
        macAddress="20:B3:99:43:54:08" apMacAddress="20:B3:99:AE:C5:AA">
        <LocationDetails floorRefId="1"
            floorName="/World/Map99 - test2_created_108B">
            <Dimension unit="FEET_AND_INCHES" width="479'0" length="721'9" />
        </LocationDetails>
        <Coordinates unit="FEET_AND_INCHES" y="255'10" x="288'8" />
        <Details lastLocatedTime="2014-11-18T22:33:33Z" />
    </DeviceLocation>
    <DeviceLocation probability="0.11" currentlyTracked="true"
        macAddress="48:F8:B3:64:6F:A7" apMacAddress="20:B3:99:AE:C5:AA">
        <LocationDetails floorRefId="1"
            floorName="/World/Map99 - test2_created_108B">
            <Dimension unit="FEET_AND_INCHES" width="479'0" length="721'9" />
        </LocationDetails>
        <Coordinates unit="FEET_AND_INCHES" y="367'5" x="223'1" />
        <Details lastLocatedTime="2014-11-18T22:33:33Z" />
    </DeviceLocation>
</DeviceLocations>
```

The top level element is “DeviceLocations,” which contains a list of “DeviceLocation” elements, one per device that has been located by the controller. The DeviceLocations element has the following attributes:

- schemaVersion – The entire schema on which the device locations file is based is versioned. The version number changes each time an element is added, an element is removed or a non-backward compatible change is made to an existing element. The current schema version is ‘1.0’.
- fileGeneratedTime – This is a standard XML dateTime representing the time at which the file was generated. This can be useful if the back end collects a batch of location files and processes them at once. The time stamp is in UTC.
- Entries – This is the total number of deviceLocation entries in the deviceLocations list. The number of entries can range from zero to many tens of thousands of entries.

The deviceLocation element is the most interesting part of the file. A deviceLocation has attributes and contains a number of elements. The attributes are:

- probability – The probability that the station is in the area identified by this location report. Theoretically the probability can range between 0.0 (no chance the user is at the specified location) and 1.0 (the station is certain to be at the location in the report). The probability is only available for triangulated location estimates.
- currentlyTracked – “true” if the location estimate is based on triangulation and “false” for CellId-RSS.
- macAddress – This is the MAC address of the station to which the deviceLocation report applies. The MAC address always is reported in the format “XX:XX:XX:XX:XX:XX” where “X” is an uppercase hexadecimal digit.
- bssid – This is the BSSID to which the station is associated at the time its location is calculated. The BSSID is a MAC address belonging to one of the associated AP’s radios. This address identifies the interface / service on the AP that the station is using to access the wired network. The BSSID always is reported in the format “XX:XX:XX:XX:XX:XX” where “X” is an uppercase hexadecimal digit. The BSSID is the same attribute normally reported by wireless NAS in the “Called-Station-ID” RADIUS

attribute in an Access-Request message. The BSSID is reported only if the station is associated to an AP managed by the controller hosting the location service or that controller's availability partner.

- ssid—This is the Service Set Identifier to which the station associated. The SSID is present only if the station has associated to a managed AP.
- apMacAddress – This is the MAC address of the AP that reports the strongest RSS reading for the station at the time the location is calculated. This is the MAC address of that AP's wired interface. The AP that reports the strongest RSS reading typically is the AP to which the station has associated. However this is not always the case. Consequently, use the BSSID attribute to identify the AP to which the station is associated, not the apMacAddress attribute.
- channelNumber – This is the identifier for the channel that the station was operating on at the time its location was reported. If the station is using a bonded channel then this is the primary channel that identifies the bonded channel. In some cases the channel number may not be known in which case it will be reported as "0".

A deviceLocation contains three sub-elements:

- LocationDetails – This element describes the location / floor plan representing the floor on which the station is located. LocationDetails must be present for triangulatedLocations and may be present for CellID-RSS based locations. So long as the reporting APs are placed on floor plans the deviceLocation will contain a LocationDetails element.
- Coordinates – This element identifies the location of the station relative to the origin for location calculations. The origin for triangulated locations is the upper left hand corner of the associated floor plan. The origin for CellID-RSS based locations is the AP identified by the apMacAddress attribute.
- Details – This element contains housekeeping details that may help with interpreting the location estimate.

The LocationDetails element has two attributes and two sub-elements of its own. The two attributes are

- floorName – the hierarchical name for the location and floor plan for that location. This name is configured in OneView maps. The names also can be seen on the controller GUI in the "Location Engine Settings" advanced dialog. The levels of the hierarchical name are separated with forward slashes ('/').
- floorRefId – a numeric identifier for the location/floor plan. On a single controller the floorRefId maps one-to-one with the floorName attribute.

LocationDetails two sub-elements are:

- Dimension – the real-world dimensions represented by the associated floor plan. Dimension has three attributes:
  - uit – the units in which the dimensions of the floor plan are expressed. This can be "METER" or "FEET_AND_INCHES" depending on which was selected when the batch location publishing feature was configured.
  - width – represents the width of the area represented by the floor plan. If the unit field is set to "METER" the width is an integer. If the unit is "FEET_AND_INCHES" the width is reported in the format "x'y" where "x" is the number of feet and "y" is the number of inches. In this case the total width in inches would be calculated as (x * 12) + y.

[20] To access the advanced dialog, open the Radar menu (by clicking on "Radar" in the top menu if necessary), open the "Location Engine" submenu, click on "Location Engine Settings" and on the page that appears, click the button labeled "Advanced...".

    - length – the depth of the area represented by the floor plan. If the unit field is set to “METER” the length is expressed as an integer. If the unit is “FEET_AND_INCHES” the length is expressed in the format “x’y” where “x” is the number of feet and “y” is the number of inches. In this case the total length in inches would be calculated as (x * 12) + y.
- FloorPlan – this element has one sub-element. That sub-element has one attribute, which is a string containing the name of the image file that contains the graphical representation of the floor plan. This is the name of the file installed into OneView to represent a given location. Release 9.12 does not make use of this sub-element.

The Coordinates element has four attributes, although not all of them are used in a single deviceLocation:

- unit – the units in which the location of the station are given. This will be “FEET_AND_INCHES” or “METER” depending on the option chosen when configuring location file publishing.
- y – the distance of the most likely location of the subject station from the location represented by the top edge of the floor plan.
- x – the distance of the most likely location of the subject station from the location represented by the left edge of the floor plan.
- apDistance – the distance the station is from the AP that reported it.

The deviceLocation always has a unit attribute. It also will have an “x” and “y” attribute if the location is triangulated. Otherwise it will have an “apDistance”.

The Details element has one attribute “lastLocatedTime,” which is the last time the station’s location was recomputed. The “lastLocatedTime” is the controller’s local time. It is possible that this will be “Never” if the location engine is not tracking the station currently. It is possible that “Never” can appear for a station that was tracked if sufficient time has passed for the location engine to stop tracking it before the station becomes visible again.

A Siemens VSA Dictionary in FreeRadius Format

This section contains the RADIUS dictionary definitions for the Siemens VSAs used by ExtremeWireless. The dictionary is in FreeRADIUS format.

```
#
# dictionary.siemens
#
# Siemens VSAs
#
VENDOR Siemens 4329
BEGIN-VENDOR Siemens
ATTRIBUTE Siemens-URL-Redirection 1 string Siemens
ATTRIBUTE Siemens-AP-Name 2 string Siemens
ATTRIBUTE Siemens-AP-Serial 3 string Siemens
ATTRIBUTE Siemens-VNS-Name 4 string Siemens
ATTRIBUTE Siemens-SSID 5 string Siemens
ATTRIBUTE Siemens-BSS-MAC 6 string Siemens
ATTRIBUTE Siemens-Policy-Name 7 string Siemens
ATTRIBUTE Siemens-Topology-Name 8 string Siemens
ATTRIBUTE Siemens-Ingress-RC-Name 9 string Siemens
ATTRIBUTE Siemens-Egress-RC-Name 10 string Siemens
END-VENDOR Siemens
```

B Siemens Session Control Web Interface Reference

Overview

This chapter is intended as a quick reference guide to the messages available in the controller's session control web interface. Details are available in Facility Reference—External Captive Portal: External Authorization on page 22, Facility Reference—External Captive Portal: Internal Authorization on page 40, and Facility Reference—Unsolicited Session Control Web Interface on page 49.

Conventions used in this Chapter

The following conventions are used in this chapter:

- Braces "{...}" identify an attribute value. The name between the braces is the name of the parameter.
- Text that is not contained between angled brackets is a literal.
- An optional parameter is enclosed in square brackets "[...]".

get_vsa_xml.php

Purpose:

Retrieve details about a user's session. The session is identified by its associated token or by the user's IP address.

Request Format:

If encryption is not used:

```
http://{controller_ip}:{controller_port}/get_vsa_xml.php?token={token_id}
[&mu_ip_addr={ipAddress}]
```

or

```
https://{controller_ip}:{controller_port}/get_vsa_xml.php?token={token_id}
[&mu_ip_addr={ipAddress}]
```

If encryption is used:

```
http://{controller_ip}:{controller_port}/get_vsa_xml.php?
param={parameter_string}
```

or

```
https://{controller_ip}:{controller_port}/get_vsa_xml.php?
param={parameter_string}
```

Table 27: Request Parameters:

Parameter	Content
controller_ip	The IP address of the session control interface as configured in the WLAN Service's external captive portal configuration dialog box.
controller_port	The port number of the session control interface as configured in the WLAN Service's external captive portal configuration dialog box.
ipAddress	The IP address of the user that is the subject of the get_vsa_xml.php request.
parameter_string	Encrypted string containing the "token_id" and possibly the "ipAddress" parameter.
token_id	Controller-generated ASCII identifier for the redirected user's session.

Response Format:

An XML document with the following format:

```
<?xml version="1.0"?>
<response>
  <token>{token_id}</token>
  <ap_name>{ap_name}</ap_name>
  <ap_serial>{ap_serial_number}</ap_serial>
  <vns_name>{vns_name}</vns_name>
  <ssid>{ssid_string}</ssid>
  <mac>{user's_mac_address}</mac>
  <status>{session_control_web_interface_return_code}</status>
  <policy>{currently_assigned_role_name}</policy>
  <topology>{assigned_role_default_topology_containment_topology}</topology>
  <ingress_rc>N/A</ingress_rc>
  <egress_rc>N/A</egress_rc>
  <BSSID>{colon_separated_BSSID_to_which_station_associated;
   e.g. 00:11:22:33:44:55}</BSSID>
</response>
```

If encryption has been configured for the interface, the return value is an ASCII string that needs to be decrypted. Once decrypted it will have the same format as if it had been transmitted unencrypted.

Comments:

This message should be used only prior to the user completing authentication. After the user has authenticated, use event.php instead.

Refer to chapter Facility Reference—External Captive Portal: External Authorization on page 22 for a detailed description of how to compose and use the encrypted and unencrypted versions of this request.

approval.php

Purpose:

Indicate to the controller that a specific user has been approved. The message can set the maximum duration of the user's session and the role to assign to the authenticated user.

Request Format:

If encryption is not used:

```
http://{controller_ip}:{controller_port}/approval.php?token={token_id}
[&mu_ip_addr={ipAddress}][&opt27={max_session_duration_seconds}]
[&username={user_name}][&filter={role_name}][&vns={vns_name}]
```

or

```
https://{controller_ip}:{controller_port}/approval.php?token={token_id}
[&mu_ip_addr={ipAddress}][&opt27={max_session_duration_seconds}]
[&username={user_name}][&filter={role_name}][&vns={vns_name}]
```

If encryption is used:

```
http://{controller_ip}:{controller_port}/approval.php?param={parameter_string}
```

or

```
https://{controller_ip}:{controller_port}/approval.php?
param={parameter_string}
```

Table 28: Request Parameters:

Parameter	Content
controller_ip	The IP address of the session control interface as configured in the WLAN Service's external captive portal configuration dialog box.
controller_port	The port number of the session control interface as configured in the WLAN Service's external captive portal configuration dialog box.
filter	The name of a role defined on the controller that is to be assigned to the authenticated user.
ipAddress	The IP address of the user that is the subject of the get_vsa_xml.php request.

Table 28: Request Parameters: (continued)

Parameter	Content
opt27	The maximum duration of the user's session in seconds. If not specified the user will be assigned the default session time configured for the WLAN Service.
parameter_string	Encrypted string containing the "token_id" and possibly the "ipAddress" parameter.
token_id	Controller-generated ASCII identifier for the redirected user's session.
user_name	The user ID that the subject user authenticated with.
vns	The name of the VNS the user is accessing. Not really useful.

Response Format:

An XML document with the following format:

```
<?xml version="1.0"?>
<response>
  <token>{token_id}</token>
  <status>{session_control_web_interface_return_code}</status>
</response>
```

Comments:

Refer to Facility Reference—External Captive Portal: External Authorization on page 22 for a detailed description of how to compose and use the encrypted and unencrypted versions of this request.

auth_user_xml.php

Purpose:

Cause the controller to interact with a RADIUS server to authenticate the credentials in the auth_user_xml.php request. If the RADIUS server rejects the request, the controller terminates the subject user's session. If the RADIUS server accepts the request, the controller applies any policy settings included in the RADIUS response. The controller notifies the sender whether the action succeeded or failed.

Request Format:

If encryption is not used:

```
http://{controller_ip}:{controller_port}/auth_user_xml.php?
token={token}&username={user_name}&password={password}
[&mu_ip_addr={ipAddress}]
```

or

```
https://{controller_ip}:{controller_port}/auth_user_xml.php?
token={token}&username={user_name}&password={password}
[&mu_ip_addr={ipAddress}]
```

If encryption is used:

```
http://{controller_ip}:{controller_port}/auth_user_xml.php?
param={parameter_string}
```

or

```
https://{controller_ip}:{controller_port}/auth_user_xml.php?
param={parameter_string}
```

Table 29: Request Parameters:

Parameter	Content
controller_ip	The IP address of the session control interface as configured in the WLAN Service's external captive portal configuration dialog box.
controller_port	The port number of the session control interface as configured in the WLAN Service's external captive portal configuration dialog box.
ipAddress	The IP address of the user that is the subject of the get_vsa_xml.php request.
password	Password entered by user into the external captive portal's login form.
parameter_string	Encrypted string containing the "token_id" and possibly the "ipAddress" parameter.
token_id	Controller-generated ASCII identifier for the redirected user's session.
user_name	The user ID that the subject user authenticated with.

Response Format:

An XML document with the following format:

```
<?xml version="1.0"?>
<response>
  <token>{token_id}</token>
  <status>{session_control_web_interface_return_code}</status>
</response>
```

Comments:

Refer to Facility Reference—External Captive Portal: Internal Authorization on page 40 for a detailed description of how to compose and use the encrypted and unencrypted versions of this request.

event.php

Purpose:

Implement unsolicited/asynchronous session control. Services in this API can be called at any time, not just when the user is authenticating. These services can be used to retrieve details about a user's session, to terminate a user's session, or to assign a new role to the station without disassociating the user. The target user is identified by a MAC or IP address.

Request Format:

If encryption is not used:

```
http://192.168.18.7:54321/event.php?type={type_code}&value={value_list}
```

or

```
https://192.168.18.7:54321/event.php?type={type_code}&value={value_list}
```

If encryption is used:

```
http://192.168.18.7:54321/event.php?param={parameters}
```

or

```
https://192.168.18.7:54321/event.php?param={parameters}
```

Request Parameters:

The "{type}" parameter is a positive integer in the range of 1-10 or 12. This parameter identifies the type of event.php request being made.

The "{value_list}" is a sequence of one or more parameters. The parameter is mandatory but has different contents and different numbers of entries depending on the type of request made. If the "{value_list}" contains more than one value the values are comma-separated. If more than one value appears in the "{value_list}," they must be listed in a specific order. The order is identified in the table below.

Table 30: Event.php request Types and Value Lists

Request Type	Type Code	Value List
Disassociate the user with the given IP address.	1	{Target_IP_Address}
Disassociate the user with the given MAC address.	2	{Target_MAC_Address} The address is a sequence of hex digits. The digits representing octets are colon-separated.
Change the role, and optionally the authentication state of the user who with the given IP address.	3	{Target_IP_Address},{Role_Name}[, {Auth_State}]

Table 30: Event.php request Types and Value Lists (continued)

Request Type	Type Code	Value List
Change the role, and optionally the authentication state of the user who with the given MAC address.	4	{Target_MAC_Address},{Role_Name}[, {Auth_State}]
Place the device with the given MAC address on a blacklist.	5	{Target_MAC_Address}
Retrieve information about the session of the user with the given MAC address.	6	{Target_MAC_Address}
Change the URL to which a specific user will be redirected after a successful authentication. Optionally, change the user's authentication state (authenticated, unauthenticated). The target is identified by IP address.	7	{Target_IP_Address},{Redirection_URL}[, {Auth_State}]
Change the URL to which a specific user will be redirected after a successful authentication. Optionally, change the user's authentication state (authenticated, unauthenticated). The target is identified by MAC address.	8	{Target_MAC_Address},{Redirection_URL}[, {Auth_State}]
Change the role currently assigned to a user and the URL to which the user will be redirected after a successful authentication. Optionally change the user's authentication state (authenticated, unauthenticated). The target is identified by IP address.	9	{Target_IP_Address},{Role_Name}, {Redirection_URL}[,{Auth_State}]
Change the role currently assigned to a user and the URL to which a specific user will be redirected after a successful authentication. Optionally change the user's authentication state (authenticated, unauthenticated). The target is identified by MAC address.	10	{Target_MAC_Address},{Role_Name}, {Redirection_URL}[,{Auth_State}]
Retrieve information about an active user's session based on the user's device's MAC address.	12	{Target_IP_Address}

Note

The "{Role_Name}" argument is case sensitive. It must match exactly the name of a role defined on the controller to which the event.php request is sent.

Table 31: Value_List Arguments

Argument	Description
{Auth_State}	Authentication state to assign to the user. Set to 1 to make the user authenticated and 2 to make the user unauthenticated.
{Redirection_URL}	A valid absolute URL. If the target user is on a WLAN Service that uses captive portal authentication and if the user is not authenticated then the user will be redirected to this URL when he/she sends HTTP traffic that is denied by the user's role.
{Role_Name}	The name of a role defined on the controller.

Table 31: Value_List Arguments (continued)

Argument	Description
{Target_IP_Address}	The IP address of the user who is the subject of the request. The IP address is in standard IP address format (e.g. 123.123.123.123).
{Target_MAC_Address}	The MAC address of the user who is the subject of the request. The MAC address is a sequence of hex digits separated by colons.

The "{parameters}" attribute is created by concatenating the type and value parameters together with a comma, then passing the result through the selected encryption algorithm (AES or legacy). Refer to chapter Facility Reference—Unsolicited Session Control Web Interface on page 49 for details on how to construct the "param" string.

Response Format:

The response to an event.php request takes the form of an XML document. The document may be encrypted if the controller is configured to use it on the session control interface that received the request. After decryption the XML response document will have one of the three formats shown in the code snippets below.

Return Code Only

```
<?xml version="1.0"?>
<response>
    <status>{Status_Code}</status>
</response>
```

Target Client not on the Controller

```
<?xml version="1.0"?>
<response>
    <client>
        Not Found
    </client>
    <status>1</status>
</response>
```

Document Returned In Response to Successful Type 6 and Type 12 Requests

```
<?xml version="1.0"?>
<response>
    <client>
        <vns_id>{vns_name}</vns_id>
        <ap_serial>{AP_Serial_Number}</ap_serial>
        <ssid>{Associated_SSID}</ssid>
```

```
            <ip_addr>{Target_IP_Address}</ip_addr>
            <mac_addr>{Target_MAC_Address}</mac_addr>
            <user>{User_Name}</user>
            <client_status>{Client_Status}</client_status>
            <session_start>{DateStamp_TimeStamp}</session_start>
            <policy>{Role_Name}</policy>
            <topology>{Topology_Name}</topology>
            <ingress_rc>N/A</ingress_rc>
            <egress_rc>N/A</egress_rc>
        </client>
        <status>1</status>
    </response>
```

Most responses take the form of Return Code Only on page 164. A single return code is all that is required to indicate that the request succeeded or failed. The response to a successful request for information about the session identified by MAC or IP address takes the form of Document Returned In Response to Successful Type 6 and Type 12 Requests on page 164. The response takes the form of Target Client not on the Controller on page 164 only when the IP or MAC address that is the subject of the request cannot be found on the controller receiving the request.

Status Codes Returned from the Session Control Web Interface

The following table lists the session control web interface's return codes.

Code	Meaning
0	Unexpected error in session control script. Should not be seen in practice.
1	Success.
2	RADIUS Server returned an ACCESS-REJECT response, mostly likely because the station's credentials weren't valid.
3	Request failed.
4	The controller timed out waiting for a response from the configured RADIUS server.
5	There is a problem with the shared secret configured for use with the RADIUS server. Confirm that the shared secret is the same on the controller and on the configured RADIUS server.
6	An internal error occurred in the controller's RADIUS client while processing the request. If this problem repeats, escalate to the Extreme Networks support organization.
7	The controllers RADIUS client has reached its limit on the maximum number of concurrent logins it can process. Try to authenticate again in a few minutes.
8	The user ID or password is missing from the auth_user_xml.php request or a parameter in the event.php request is invalid.
9	Request failed.
14	Returned if the parameters to an event.php request are invalid.
15	The device/user that is the target of the request was recently redirected to a captive portal. The station is likely engaged in finishing authentication. Either try again in 5 minutes or use a controller interface to disassociate the target.
16	The token is invalid. It either does not have the correct internal format or does not identify a session currently on the controller that received the request.

Code	Meaning
18	The controller could not locate a session for the identifier in the request (MAC address, IP address or token).
19	The controller is handling the maximum number of event.php and captive portal authentication requests. Try again in a few minutes.
20	An event.php request is missing both an IP address and a MAC address and so the target can't be identified. This code is also returned if the request does not change either the redirection URL or the role assigned to the user.
21	The role name argument does not match the name of a role defined on the controller.
99	The controller could not complete the request within a reasonable amount of time. This is likely due to an issue on the controller that is temporarily slowing it down.

C PHP External Captive Portal: External Authentication

net-auth.php
login.php
common_utilities.php
crypt_aes.php
crypt_md5.php

This appendix contains a set of scripts that together implement a basic external captive portal (ECP). This type of ECP is responsible for collecting credentials from a user, authenticating them and then ExtremeWireless the access policy to apply to the user.

The scripts are minimal because they are intended to highlight the critical steps required by an external captive portal implementation.

This appendix contains five scripts:

- net-auth.php on page 167 — This is the script to which the controller redirects user's requiring authentication. This script is responsible for saving critical information required to identify the user in session control API calls. The script may call get_vsa_xml.php to obtain details about the user so that they can be displayed on the login page. net_auth.php is responsible for creating the login page or for directing the user to it.
- login.php — This is the script that accepts submitted user credentials and forwards them to the controller for processing. It receives the response from the controller and sends the user to an appropriate web page.
- common_utilities.php — Some simple utilities called by net_auth.php and login.php. They are not essential to understanding how an ECP works, but are required if the scripts above are to be executed.
- crypt_aes.php — A file containing a simple wrapper around the calls used for AES encryption. It massages the data in the same way that the controller does before encryption or decryption.
- crypt_md5.php — A file containing scripts that implement the MD5 encryption and decryption algorithms used when legacy encryption is used to secure communication through the session control web interface.

net-auth.php

```
<?php
// net-auth.php
// This is a simple implementation of a script that receives HTTP requests
that have been redirected
// by the controller. This script is responsible for collecting critical
// information from the redirection (in particular the session token)
// and for constructing the login page for the user.
```

```
// The script uses get_vsa_xml.php to get the SSID the user associated to
// so that it can be displayed on the login page.
// This script is able to interact with the session control interface
// regardless of the type of encryption that is configured for it.
// This is mainly for illustrative purposes.
// In an actual deployment the script is likely to only need to support
// one type of encryption or no encryption at all.
//
// Assumptions
// ===========
// 1. The controller is configured to include its IP address and port in the
redirection URL.
// 2. The variable sc_encyption is defined in php.ini. It can take one of
three values:
    //    0 - no encryption
    //    1 - MD5 encryption
    //    2 - AES encryption
    //    If the variable is undefined then the script assumes no encyption
is required.
// 3. If encryption is required then sc_key is defined in php.ini.
// It is the same encryption key that is defined on the controller's WLAN
Service's
// Captive Portal Configuration dialog box.
// 4. Whether the script uses HTTP or HTTPS depends on the value of
// sc_use_https which must be defined in php.ini.
//    If the value of sc_use_https is 1 then the script uses HTTPS.
// If the configuration variable has any other value or is not defined,
// then the script uses HTTP.
// In practice, an actual site is going to settle on using HTTP or HTTPS.
//    The scripts can then assume that method is being used
//    rather than looking up the method in php.ini.
// The variables, sc_encryption, sc_key and sc_use_https are user-defined
variables.
// They must be created in php.ini by the web server administrator.
    require_once("crypt_md5.php");
    require_once("crypt_aes.php");
    require_once("common_utilities.php");
// Mainline processing starts here. Utilities are defined after the mainline.
    // 1. Get the variables passed from the controller on the redirection URL.
    // The token is the most important of these variables.
    $token = isset($_REQUEST['token']) ? $_REQUEST['token'] : "";
    if(0 == strlen($token)) {
    // The controller always includes the token so this is suspicious.
    // Maybe someone tried to access this page using a handcrafted URL.
    printError("Token is mandatory but missing from request parameters.");
    exit;
    }
    // Get the URL that the user was trying to go to when he or she was
redirected here.
    $dest = isset($_REQUEST['dest']) ? convertUrlParam($_REQUEST['dest']) :
"";
    // Determine which controller interface to interact with
    if(isset($_REQUEST['hwc_ip']) && isset($_REQUEST['hwc_port'])) {
        //BM IP address and port is enabled
        $ewc_ip = trim($_REQUEST['hwc_ip']);
        $ewc_port = trim($_REQUEST['hwc_port']);
    } else {
        // The controller has not been configured as expected. It did not
```

```
        // include its address and port on the redirection URL. This is
        // easy to fix but all this program can do is report the error.
        printError("Controller must be configured to include its IP address &
port in the request.");
        exit;
    }
    if(!tokenCheck($token)) {
    printError("Error: <span style='color:red'>Failed to process the request:
incorrect token.</span>");
    exit;
    } else if(isset($ewc_port) && !is_numeric($ewc_port)) {
    printError("Error: <span style='color:red'>Failed to process the request:
incorrect EWC port.</span>");
    exit;
    }
    // Get this user's IP address. This comes from the host web server, not
    // the redirected HTTP request.
    $mu_ip_addr = isset($_SERVER['REMOTE_ADDR']) ?
          $_SERVER['REMOTE_ADDR'] : "";
    // 2. Get some details about the user from the controller.
    // This is optional. It is included here to show how to do it.
    $user_attributes = get_vsa_xml($ewc_ip, $ewc_port, $mu_ip_addr,
            $token);
    if (false === $user_attributes) {
        // get_vsa_xml already output the error message.
        exit;
    }
    // 3. Compose the login page and send it to the user.
    print compose_login_page($ewc_ip, $ewc_port, $token, $dest,
           $user_attributes);
      // 4. And exit. This script is finished executing.
    exit;
    // End of mainline
    // get_vsa_xml composes and sends a request to the redirecting
    // controller to get some details about the current user. If
    // the requests succeeds the response is returned. If the
    // request fails the procedure returns false.
    function get_vsa_xml($ewc_ip, $ewc_port, $mu_ip_addr, $token)
    {
        if (get_cfg_var('sc_use_https')) {
            if (1 == get_cfg_var('sc_use_https')) {
                $protocol = "https://";
            } else {
                $protocol = "http://";
            }
        } else {
            $protocol = "http://";
        }
        $ewc_addr = $protocol.$ewc_ip.":".$ewc_port."/get_vsa_xml.php?";
        // Determine the type of encryption to use: 1->Legacy;
            // 2->AES (configure in ecp page)
        $cp_encryption = get_cfg_var('sc_encryption')?
                  intval(get_cfg_var('sc_encryption')) : 0;
        $crypt_key = get_cfg_var('sc_key')? get_cfg_var('sc_key') : "";
        // Build the query string for the request
        switch ($cp_encryption)
        {
            case 1: /* MD5 encryption selected */
```

```
                // Crypt_md5 is a php script function from the
                        // open source community
                $query_string = "param=".bin2hex(crypt_md5("mu_ip_addr=
$mu_ip_addr,token=$token", $crypt_key));
                break;
            case 2: /* AES Encryption selected */
                $query_string =
"param=".base64_url_encode(crypt_aes("mu_ip_addr=$mu_ip_addr,token=$token",
$crypt_key));
                break;
            default: /* No encryption requested */
                $query_string = "mu_ip_addr=$mu_ip_addr&token=$token";
        }
        // Now call get_vsa_xml.php to retrieve an XML document
            // describing the identified user.
        $user_attributes_xml =
                  file_get_contents($ewc_addr.$query_string);
        if (false === $user_attributes_xml) {
            printError("Could not get the attributes for this user”.
                  " from the controller.".
            " Please try again in 5 minutes.");
            return false;
        }
        // Decrypt the reply if necessary
        if (1 == $cp_encryption) {
            // MD5 encryption selected
            $user_attributes_xml =  decrypt_md5(
                hex2bin($user_attributes_xml), $crypt_key);
        } else {
            // AES encryption selected
            if (2 == $cp_encryption) {
                $user_attributes_xml = decrypt_aes(
                    base64_url_decode($user_attributes_xml),
                                $crypt_key);
                               file_put_contents("getvsatest.txt",
strftime("%c")."\n".$user_attributes_xml."\n\n", FILE_APPEND);
            }
        }
        $user_attributes = my_xml2array($user_attributes_xml);
        if (false === $user_attributes) {
            printError("Could not parse the answer returned by the
controller.");
            return false;
        }
        return $user_attributes;
    }
    // This function generates a basic login page containing a form
      // that allows the user to submit his credentials back to this
      // server. The page displays the SSID of the service the user
      // associated to. Otherwise the page is pretty plain. Obviously
      // a real login page would have more content on it. This routine
      // highlights the critical aspects of composing the login page so
      // that when the user submits his credentials all the information
      // necessary to manage the user's session will be on the page.
    function compose_login_page($ewc_ip, $ewc_port, $token, $dest,
          $user_attributes)
    {
        $SSID = get_value_by_path($user_attributes, 'response/ssid' );
```

```
        $template = "<!DOCTYPE html>
<html>
<head>
    <meta charset=\"ISO-8859-1\">
    <title>Please Login</title>
</head>
<body>
    <form id=\"Login\" name=\"Login\" method=\"post\" action=\"login.php\">
        <table border='0' width='800' height='310' cellpadding='0'
            cellspacing='0'>
            <tr>
                <td colspan=\"3\" height=\"100\"> </td>
            </tr>
            <tr>
                <td width='260' height='1' border='0'/>
                <td width='300' height='65'>
                    Please login to use '$SSID' network.</td>
                <td width='240' rowspan='5'> </td>
            </tr>
            <tr>
                <td align=\"right\"><b>User Name  </b>
                </td>
                <td height=\"28\">
                    <input type=\"text\" autocomplete=\"off\"
                        id=\"userid\" name=\"userid\" tabindex=\"1\">
                </td>
            </tr>
            <tr>
                <td align=\"right\"><b>Password  </b>
                </td>
                <td height=\"28\"><input type=\"password\" autocomplete=\"off
\"
                    id=\"passwd\" name=\"passwd\" tabindex=\"2\">
                </td>
            </tr>
            <tr>
                <td><br>
                </td>
                <td height=\"33\" valign=\"bottom\"><input type=\"submit\"
                    style=\"width: 100px\" value=\"Login\" tabindex=\"3\">
                </td>
            </tr>
        </table>
        <input type=\"hidden\" name=\"ewc_ip\" id=\"ewc_ip\" value=\"$ewc_ip
\"/>
        <input type=\"hidden\" name=\"ewc_port\" id=\"ewc_port\" value=
\"$ewc_port\"/>
        <input type=\"hidden\" name=\"token\" id=\"token\" value=\"$token\"/>
        <input type=\"hidden\" name=\"dest\" id=\"dest\" value=\"http://$dest
\" />
    </form>
</body>
</html>";
        return $template;
    }
?>
```

login.php

```
<?php
// login.php
// This is a simple implementation of a script that receives a user's
credentials,
// authenticates the credentials, selects an access control role for
// the user, configures the controller to apply the role to the user,
// then redirects the user to his or her original destination URL.
// This script assumes that the credentials are submitted on the form
// created by the example script net-auth.php.
//
// This script is able to interact with the session control interface
// regardless of the type of encryption that is configured for it.
// This is mainly for illustrative purposes.
// In an actual deployment, the script is likely to only need to
// support one type of encryption or no encryption at all.
//
// Assumptions
// ===========
// 1. The controller is configured to include its IP address
    // and port in the redirection URL and the submitted login
    // form contains that IP address and port. This allows the
    // ECP to interact with more than 1 controller.
// 2. The variable sc_encyption is defined in php.ini.
    // It can take one of three values:
    // 0 - no encryption
    // 1 - MD5 encryption
    // 2 - AES encryption
    // If the variable is undefined then the script assumes no encyption is
required.
// 3. If encryption is required then sc_key is defined in php.ini.
    //It is the same encryption key that is defined on the controller's
    // WLAN Service's Captive Portal Configuration dialog box.
// 4. Whether the script uses HTTP or HTTPS depends on the
    // value of sc_use_https which must be defined in php.ini.
    // If the value of sc_use_https is 1 then the script uses
    // HTTPS. If the configuration variable has any other value
    // or is not defined then the script uses HTTP. In practice
    // an actual site is going to settle on using HTTP or HTTPS.
    // The scripts can then assume that method is being used
    // rather than looking up the method in php.ini.
    // The variables, sc_encryption, sc_key and sc_use_https are
    // user-defined variables. They must be created in php.ini by
    // the web server administrator.
    require_once("crypt_md5.php");
    require_once("crypt_aes.php");
    require_once("common_utilities.php");
    // The mainline begins here. The utilities are defined after
    // the mainline.
    // 1. Collect the parameters submitted on the login form.
    // Some of these attributes come from hidden fields.
    $ewc_ip = trim($_REQUEST['ewc_ip']);
    $ewc_port = trim($_REQUEST['ewc_port']);
    $redirection = trim($_REQUEST['dest']);
    $token = trim($_REQUEST['token']);
    $username = (isset($_REQUEST['userid'])) ?
           trim($_REQUEST['userid']) : "";
```

```
    $passwd = (isset($_REQUEST['passwd'])) ?
           trim($_REQUEST['passwd']) : "";
    if(!tokenCheck($token)) {
    printError("Error: <span style='color:red'>Failed to process the request:
incorrect token.</span>");
    exit;
    } else if(isset($ewc_port) && !is_numeric($ewc_port)) {
    printError("Error: <span style='color:red'>Failed to process the request:
incorrect EWC port.</span>");
    exit;
    }
    // For this example the maximum duration of any user's
    // session will be 36000 seconds. The session is terminated
    // no later than this time. After the session is terminated
    // the user can access the network but will be
    // unauthenticated again.
    $max_duration = 36000;
    // It is a good idea to send both the token & the user's IP
    // address to the controller. The user's IP address is
    // reported by the web server and is not on the submitted
    // login form.
    $mu_ip_addr = isset($_SERVER['REMOTE_ADDR']) ?
        $_SERVER['REMOTE_ADDR'] : "";
    // 2. Authenticate the user & select an appropriate role.
    // Selecting the role is optional. If one is not specified
    // for the controller the controller will apply the
    // default authenticated role of the WLAN Service the user
    // is accessing.
    $assigned_role = authenticate($username, $passwd);
    if (false === $assigned_role) {
        // Failed to authenticate the user.
        // Authenticate prints the error message for
        // the browser. We just exit here.
        exit;
    }
    // 3. Tell the controller that the user is authenticated
    // and tell it which role to apply to the user.
    if (approve($ewc_ip, $ewc_port, $token, $mu_ip_addr,
        $username, $assigned_role, $max_duration)) {
        // 4. Redirect the user to his original destination.
        // This is optional. If desired the script could compose
        // a special landing page for the user.
        header( 'Location: '.$redirection);
    }
    exit;
    // End of mainline.
    // A method that validates the user's credentials and
    // returns the role to apply to the user. In some cases
    // this routine might also return the maximum session
    // duration in seconds.
    //
    // For purposes of this example this procedure is
    // not much more than a stub. The stub can be replaced
    // by any authentication method, such as sending access
    // requests to a backend RADIUS server, or performing
    // a lookup in an application credential database.
    function authenticate($userid, $passwd) {
        if (("" == $userid) || ("" == $passwd)) {
```

```
            printError("Invalid Userid or Password. ".
            "Please press the 'Back' button and try again.");
            // If you generate another login page for the user
            // be sure to copy the ewc_ip address, ewc_port,
            // token and dest attributes from the submitted
            // login form to the login page.
            return false;
        } else {
            // Return the name of a role to be applied
            // to the station. The role must be defined on
            // the controller or it will substitute the
            // default authenticated role of the VNS the
            // user is logging into.
            // For purposes of this example assume all
            // authenticated users get the 'Guest_Access' role.
            return "Guest_Access";
        }
    }
    // A function that submits an approval.php request to the
    // controller that directed the user here. Approval.php
    // tells the controller that the station in question is
    // approved and optionally, how long the use's session
    // can be and the role to apply to the station.
    function approve($ewc_ip, $ewc_port, $token, $mu_ip_addr,
         $userid, $assigned_role, $max_duration) {
        if (get_cfg_var('sc_use_https')) {
            if (1 == get_cfg_var('sc_use_https')) {
                $protocol = "https://";
            } else {
                $protocol = "http://";
            }
        } else {
            $protocol = "http://";
        }
        $ewc_addr = $protocol.$ewc_ip.":".$ewc_port."/approval.php?";
        // Determine the type of encryption to use: 1->Legacy;
        // 2->AES (configure in ecp page)
        $cp_encryption = get_cfg_var('sc_encryption')?
                 intval(get_cfg_var('sc_encryption')) : 0;
        $crypt_key = get_cfg_var('sc_key')? get_cfg_var('sc_key') : "";
        // Build the query string for the request
        switch ($cp_encryption)
        {
            case 1: /* MD5 Encryption selected */
                // Crypt_md5 is a php script function from the open
                    // source community
                $parms=make_query_string($token, $mu_ip_addr,
                    $userid, $assigned_role, $max_duration, ",");
                $query_string = "param=".bin2hex(crypt_md5($parms,
                                $crypt_key));
                break;
            case 2: /* AES Encryption selected */
                $parms=make_query_string($token, $mu_ip_addr,
                    $userid, $assigned_role, $max_duration, ",");
                $query_string =
                                "param=".base64_url_encode(crypt_aes($parms,
                    $crypt_key));
                break;
```

```
            default: /* No encryption requested */
                $query_string = make_query_string($token,
                    $mu_ip_addr, $userid, $assigned_role,
                    $max_duration, "&");
        }
        // Now call approval.php to tell the controller the new settings
            // for the user session.
        $approval_response_xml =
                    file_get_contents($ewc_addr.$query_string);
        if (false === $approval_response_xml) {
            printError("Could not access the controller to tell it to".
                " authorize you. Please try again in 5 minutes.");
            return false;
        }
        // Decrypt the reply if necessary
        if (1 == $cp_encryption) {
            // MD5 encryption selected
            $approval_response_xml =  decrypt_md5(
                hex2bin($approval_response_xml), $crypt_key);
        } else {
            // AES encryption selected
            if (2 == $cp_encryption) {
                $approval_response_xml = decrypt_aes(
                    base64_url_decode($approval_response_xml),
                        $crypt_key);
            }
        }
        // Massage the response to make it easier to proces.
        $approval_response = my_xml2array($approval_response_xml);
        $ewc_token = get_value_by_path($approval_response,
                 'response/token');
        $ewc_status = get_value_by_path($approval_response,
                 'response/status');
        if( ($token == $ewc_token) && (1 == $ewc_status) ){
            // Success path exit.
             return true;
        } else {
            // Something went wrong. Output a suitable error message.
               if  ($token != $ewc_token) {
                   // Somehow the controller responded to the wrong
                 // request.
                   printError("Error: <span style='color:red'>Unexpected
response from access controller.
                 Please try again in 5 minutes.</span>");
              } else {
                   // Print an error message corresponding to the return
                 // code.
                   printError("Error: ".code_2_message($ewc_status,
                            'content'));
              }
              return false;
          }
    }
    function make_query_string($token, $mu_ip_addr, $userid,
                    $assigned_role, $max_duration,
                    $separator) {
        $q_str = "token=".$token;
             if (is_not_empty($mu_ip_addr)) {
```

```
            $q_str = $q_str.$separator."mu_ip_addr=".$mu_ip_addr;
        }
             if (is_not_empty($userid)) {
                   $q_str = $q_str.$separator."username=".$userid;
             }
        if (is_not_empty($assigned_role)) {
            $q_str = $q_str.$separator."filter=".$assigned_role;
        }
             if (is_not_empty($max_duration)) {
            $q_str = $q_str.$separator."opt27=".$max_duration;
             }
        return $q_str;
    }
?>
```

common_utilities.php

```
<?php
// A library of utilities that can be used by PHP scripts comprising an
external captive portal.
// A utility that translates error codes to error messages.
function code_2_message($code, $content_type)
{
    $errMsgList = array (
        0 =>
        array (
          'label' => 'Invalid',
          'content' => '<span style=\'color:red\'>Empty id / password not
allowed. Please try again.</span>',
        ),
        1 =>
        array (
          'label' => 'Success',
          'content' => 'Success',
        ),
        2 =>
        array (
          'label' => 'Access Fail',
          'content' => '<span style=\'color:red\'>Userid or password
incorrect. Please try again.</span>',
        ),
        3 =>
        array (
          'label' => 'Fail',
          'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
        ),
        4 =>
        array (
          'label' => 'Timeout',
          'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
            ),
        5 =>
        array (
```

```
          'label' => 'RADIUS shared security key fail',
          'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
        ),
        6 =>
        array (
          'label' => 'RADIUS internal error',
          'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
        ),
        7 =>
        array (
          'label' => 'Max RADIUS login fail',
          'content' => '<span style=\'color:red\'>Too many users trying to
login at the same time.Please try again later.</span>',
        ),
        8 =>
        array (
          'label' => 'Invalid Login parameters',
          'content' => '<span style=\'color:red\'>Userid or password
incorrect. Please try again.</span>',
        ),
        9 =>
        array (
          'label' => 'General failure',
          'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
        ),
        14 =>
        array (
          'label' => 'Invalid third party parameters',
          'content' => '<span style=\'color:red\'>Invalid third party
parameters.</span>',
        ),
        15 =>
        array (
          'label' => 'Authentication in progress failure',
          'content' => '<span style=\'color:red\'>Authentication is in
progress.</span>',
        ),
        17 =>
        array (
          'label' => 'Max concurrent session failure',
          'content' => '<span style=\'color:red\'>Login rejected because the
maximum number of concurrent sessions for this set of credentials has been
reached. Please try again later.</span>',
        ),
        18 =>
        array (
          'label' => 'Identified session not found',
          'content' => '<span style=\'color:red\'>Login failed because could
not find a session for the specified identifiers.</span>'
        ),
        99 =>
        array (
```

```
          'label' => 'Timeout while trying to authorize a session',
          'content' => '<span style=\'color:red\'>Login failed because
because the controller took too long to authorize the session.</span>'
        )
      );
      return (isset($errMsgList[$code])) ?
$errMsgList[$code][$content_type] :
          "Unrecognized error code: ".$code;
}
// General purpose error reporting procedure.
function printError($errorMsg) {
    header('Content-type: text/html; charset=iso-8859-1');
    print "<html>\n<head><title>Error</title></head><body>\n<p>\n$errorMsg
\n</p>\n</body>\n</html>\n";
}
// Use base64 url safe encode/decode when dealing
// with AES-encrypted strings.
// encode: '+'=>'-', '/' => '_' , '=' => '!'
function base64_url_encode($input) {
    return strtr(base64_encode($input), '+/=', '-_!');
}
// Decode: '-'=>'+', '_' => '/' , '!' => '='
function base64_url_decode($input) {
    return base64_decode(strtr($input, '-_!', '+/='));
}
// xml parsing functions
function my_xml2array($contents)
{
    $xml_values = array();
    if (! isset($contents)) {
        return false;
    }
    $parser = xml_parser_create('');
    if(!$parser) {
    return false;
    }
    xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING,
                'UTF-8');
    xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
    xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
    xml_parse_into_struct($parser, trim($contents), $xml_values);
    xml_parser_free($parser);
    if (!$xml_values) {
        return array();
    }
    $xml_array = array();
    $last_tag_ar =& $xml_array;
    $parents = array();
    $last_counter_in_tag = array(1=>0);
    foreach ($xml_values as $data)
    {
        switch($data['type'])
        {
            case 'open':
                $last_counter_in_tag[$data['level']+1] = 0;
                $new_tag = array('name' => $data['tag']);
                if(isset($data['attributes']))
                $new_tag['attributes'] = $data['attributes'];
```

```
                if(isset($data['value']) &&
                                trim($data['value']))
                $new_tag['value'] = trim($data['value']);
                $last_tag_ar[$last_counter_in_tag[
$data['level']]] = $new_tag;
                $parents[$data['level']] =& $last_tag_ar;
                $last_tag_ar =& $last_tag_ar[
                    $last_counter_in_tag[$data['level']]++];
                break;
            case 'complete':
                $new_tag = array('name' => $data['tag']);
                if(isset($data['attributes']))
                $new_tag['attributes'] = $data['attributes'];
            if(isset($data['value']) &&
                    trim($data['value']))
                $new_tag['value'] = trim($data['value']);
                $last_count = count($last_tag_ar)-1;
                $last_tag_ar[ $last_counter_in_tag[$data[
'level' ]]++ ] = $new_tag;
                break;
            case 'close':
                $last_tag_ar =& $parents[$data['level']];
                break;
            default:
                break;
        };
    }
    return $xml_array;
}
function get_value_by_path($__xml_tree, $__tag_path)
{
    $tmp_arr =& $__xml_tree;
    $tag_path = explode('/', $__tag_path);
    foreach($tag_path as $tag_name)
    {
        $res = false;
        foreach($tmp_arr as $key => $node)
        {
            if(is_int($key) && $node['name'] == $tag_name)
            {
                $tmp_arr = $node;
                $res = true;
                break;
            }
        }
        if(!$res) {
            return false;
        }
    }
    if( isset($tmp_arr['value']) ) {
        return $tmp_arr['value'];
    } else {
        return null;
    }
}
function is_not_empty($string) {
    return (isset($string) && (0 < strlen($string)));
}
```

```
//check token format
  function tokenCheck($val){
      return preg_match('/^([a-zA-Z0-9-_!]){0,24}$/', $val);
  }
  //check the mac address
  function macCheck($val){
      return preg_match("/^[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-
f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}$/", $val);
  }
  //encode the input string to avoid script attack
  function convertUrlParam($input) {
      return htmlentities($input, ENT_QUOTES);
}
?>
```

crypt_aes.php

```
<?php
// A wrapper around the calls to Openssl encryption and decryption.
define("AES_BLOCK_LEN", 16);
// Use the following PKCS#7 padding (RFC 5652) if the chosen
// encryption library does not add PKCS#7 padding
// automatically. Openssl does so crypt_aes.php does not call
// aes_cbc_pad.
function aes_cbc_pad($message)
{
    $length = strlen($message);
    $pad = AES_BLOCK_LEN - ($length % AES_BLOCK_LEN);
    return (AES_BLOCK_LEN == $pad) ? message : str_pad($message,
    $length + $pad, chr($pad));
}
// A wrapper around openssl AES encryption. The wrapper sets
// the IV to 16 binary zeroes and selects the encryption
// algorithm based on the key length. The algorithm is
// always a variant of AES in Cipher Block Chaining (CBC)
// mode.
function crypt_aes($message, $shared_key)
{
    $iv = pack('H*', "00000000000000000000000000000000");
    $method = (strlen($shared_key) >= 32) ? 'aes-256-cbc' :
              'aes-128-cbc';
    return openssl_encrypt($message, $method, $shared_key,
              true, $iv);
}
// A wrapper around openssl AES decryption. The wrapper sets
// the IV to 16 binary zeroes and selects the decryption
// algorithm based on the key length. The algorithm is
// always a variant of AES in Cipher Block Chaining (CBC)
// mode.
function decrypt_aes($message, $shared_key)
{
    $iv = pack('H*', "00000000000000000000000000000000");
    $method = (strlen($shared_key) >= 32) ? 'aes-256-cbc' :
              'aes-128-cbc';
    return openssl_decrypt($message, $method, $shared_key,
              true, $iv);
```

```
}
>
```

crypt_md5.php

```
<?php
// Copyright: hkmaly@matfyz.cz
// http://adela.karlin.mff.cuni.cz/~hkmaly/php/
// Extremenetworks.com: Added minor modifications to
// to deal with input that has a length that
// is not a multiple of 16.
function hex2bin($data)
    {
        $len = strlen($data);
        return pack("H" . $len, $data);
    }
    function bytexor($a,$b,$l)
    {
        $c="";
        for($i=0;$i<$l;$i++) { $c.=$a{$i}^$b{$i}; }
        return($c);
    }
    function binmd5($val)
    {
        return(pack("H*",md5($val)));
    }
    function decrypt_md5($msg,$heslo)
    {
        $key=$heslo;$sifra="";
        $key1=binmd5($key);
        $msglen=strlen($msg);
        while($msglen >= 16) {
            $m=substr($msg,0,16);
            $msg=substr($msg,16);
            $msglen -= 16;
            $sifra.=$m=bytexor($m,$key1,16);
            $key1=binmd5($key.$key1.$m);
        }
    if ($msglen > 0) {
        $sifra.=bytexor($msg,$key1,$msglen);
    }
        return($sifra);
    }
    function crypt_md5($msg,$heslo)
    {
        $key=$heslo;$sifra="";
        $key1=binmd5($key);
        $msglen=strlen($msg);
        while($msglen >= 16) {
            $m=substr($msg,0,16);
            $msg=substr($msg,16);
            $msglen -= 16;
            $sifra.=bytexor($m,$key1,16);
            $key1=binmd5($key.$key1.$m);
        }
    if ($msglen > 0) {
        $sifra.=bytexor($msg,$key1,$msglen);
```

```
        }
            return($sifra);
        }
    ?>
```

D PHP External Captive Portal: Internal Authentication

net-auth.php
login.php
common_utilities.php
crypt_aes.php
crypt_md5.php

This appendix contains a set of scripts that together implement a basic external captive portal. The captive portal in this case collects credentials from a user and forwards them to the controller for authentication. The controller in turn forwards them to a configured RADIUS server and applies to the target user the role returned by the RADIUS server in the ACCESS-ACCEPT response. The controller responds to the external captive portal with an HTTP response indicating whether the authentication was successful.

This appendix contains five scripts:

- net_auth.php

  This is the script to which the controller redirects user's requiring authentication. This script is responsible for saving critical information required to identify the user in session control API calls. The script may call get_vsa_xml.php to obtain details about the user so that they can be displayed on the login page. net_auth.php is responsible for creating the login page or for directing the user to it.
- login.php

  This is the script that accepts submitted user credentials, verifies them against an authentication database and uses the session control web interface to tell the controller the policy to apply to the user.
- common_utilities.php

  Some simple utilities called by net_auth.php and login.php. They are not essential to understanding how an ECP works, but are required if the scripts above are to be executed.
- crypt_aes.php

  A file containing a simple wrapper around the calls used for AES encryption. It massages the data in the same way that the controller does before encryption or decryption.
- crypt_md5.php

  A file containing scripts that implement the MD5 encryption and decryption algorithms used when legacy encryption is used to secure communication through the session control web interface.

net-auth.php

```
<?php
    // net-auth.php
    // This is a simple implementation of a script that
    // receives HTTP requests that have been redirected
    // by the controller.
    // This script is responsible for collecting critical
    // information from the redirection (in particular the
    // session token) and for constructing the login page
    // for the user. The script uses get_vsa_xml.php to
    // get the SSID the user associated to so that it can
    // be displayed on the login page.
    // This script is able to interact with the session
    // control interface regardless of the type of encryption
    // that is configured for it. This is mainly for
    // illustrative purposes. In an actual deployment the script
    // is likely to only need to support one type of encryption
    // or no encryption at all.
    //
    // Assumptions
    // ===========
    // 1. The controller is configured to include its IP address
    //    and port in the redirection URL.
    // 2. The variable sc_encyption is defined in php.ini. It can
    //    take one of three values:
    //    0 - no encryption
    //    1 - MD5 encryption
    //    2 - AES encryption
    //    If the variable is undefined then the script assumes
    //    no encryption is required.
    // 3. If encryption is required then sc_key is defined in
    //    php.ini. It is the same encryption key that is defined
    //    on the controller's WLAN Service's Captive Portal
    //    Configuration dialog box.
    // 4. Whether the script uses HTTP or HTTPS depends on the
    //    value of sc_use_https which must be defined in
    //          php.ini.
    //    If the value of sc_use_https is 1 then the script uses
    //    HTTPS. If the configuration variable has any other
    //          value or is not defined then the script uses HTTP. In
    //           practice an actual site is going to settle on
using                              HTTP or HTTPS.
    //    The scripts can then assume that method is being used
    //    rather than looking up the method in php.ini.
    // The variables, sc_encryption, sc_key and sc_use_https are
    // user-defined variables. They must be created in php.ini by
    // the web server administrator.
    require_once("crypt_md5.php");
    require_once("crypt_aes.php");
    require_once("common_utilities.php");
    // Mainline processing starts here. Utilities are defined
      // after the mainline.
    // 1. Get the variables passed from the controller on the
      // redirection URL.
    // The token is the most important of these variables.
    $token = isset($_REQUEST['token']) ? $_REQUEST['token'] : "";
    if(0 == strlen($token)) {
```

```
        // The controller always includes the token so this is
// suspicious.
        // Maybe someone tried to access this page using a
            // handcrafted URL.
        printError("Token is mandatory but missing from request parameters.");
        exit;
    }
    // Get the URL that the user was trying to go to when he or she
    // was redirected here.
    $dest = isset($_REQUEST['dest']) ? $_REQUEST['dest'] : "";
    // Determine which controller interface to interact with
    if(isset($_REQUEST['hwc_ip']) && isset($_REQUEST['hwc_port'])) {
        //BM IP address and port is enabled
        $ewc_ip = trim($_REQUEST['hwc_ip']);
        $ewc_port = trim($_REQUEST['hwc_port']);
    } else {
        // The controller has not been configured as expected. It did not
        // include its address and port on the redirection URL. This is
        // easy to fix but all this program can do is report the error.
        printError("Controller must be configured to include its IP address &
port in the request.");
        exit;
    }
    // Get this user's IP address. This comes from the host web server, not
    // the redirected HTTP request.
    $mu_ip_addr = isset($_SERVER['REMOTE_ADDR']) ? $_SERVER['REMOTE_ADDR'] :
"";
    // 2. Get some details about the user from the controller.
    // This is optional. It is included here to show how to do it.
    $user_attributes = get_vsa_xml($ewc_ip, $ewc_port, $mu_ip_addr,
            $token);
    if (false === $user_attributes) {
        // get_vsa_xml already output the error message.
        exit;
    }
    // 3. Compose the login page and send it to the user.
    print compose_login_page($ewc_ip, $ewc_port, $token, $dest,
           $user_attributes);
      // 4. And exit. This script is finished executing.
    exit;
    // End of mainline
    // get_vsa_xml composes and sends a request to the redirecting
    // controller to get some details about the current user. If
    // the requests succeeds the response is returned. If the
    // request fails the procedure returns false.
    function get_vsa_xml($ewc_ip, $ewc_port, $mu_ip_addr, $token)
    {
        if (get_cfg_var('sc_use_https')) {
            if (1 == get_cfg_var('sc_use_https')) {
                $protocol = "https://";
            } else {
                $protocol = "http://";
            }
        } else {
            $protocol = "http://";
        }
        $ewc_addr = $protocol.$ewc_ip.":".$ewc_port."/get_vsa_xml.php?";
        // Determine the type of encryption to use: 1->Legacy;
```

```
            // 2->AES (configure in ecp page)
        $cp_encryption = get_cfg_var('sc_encryption')?
                  intval(get_cfg_var('sc_encryption')) : 0;
        $crypt_key = get_cfg_var('sc_key')? get_cfg_var('sc_key') : "";
        // Build the query string for the request
        switch ($cp_encryption)
        {
            case 1: /* MD5 encryption selected */
                // Crypt_md5 is a php script function from the
                        // open source community
                $query_string = "param=".bin2hex(crypt_md5("mu_ip_addr=
$mu_ip_addr,token=$token", $crypt_key));
                break;
            case 2: /* AES Encryption selected */
                $query_string =
"param=".base64_url_encode(crypt_aes("mu_ip_addr=$mu_ip_addr,token=$token",
$crypt_key));
                break;
            default: /* No encryption requested */
                $query_string = "mu_ip_addr=$mu_ip_addr&token=$token";
        }
        // Now call get_vsa_xml.php to retrieve an XML document
            // describing the identified user.
        $user_attributes_xml =
                  file_get_contents($ewc_addr.$query_string);
        if (false === $user_attributes_xml) {
            printError("Could not get the attributes for this user”.
                  " from the controller.".
            " Please try again in 5 minutes.");
            return false;
        }
        // Decrypt the reply if necessary
        if (1 == $cp_encryption) {
            // MD5 encryption selected
            $user_attributes_xml =  decrypt_md5(
                hex2bin($user_attributes_xml), $crypt_key);
        } else {
            // AES encryption selected
            if (2 == $cp_encryption) {
                $user_attributes_xml = decrypt_aes(
                    base64_url_decode($user_attributes_xml),
                                $crypt_key);
                            file_put_contents("getvsatest.txt",
strftime("%c")."\n".$user_attributes_xml."\n\n", FILE_APPEND);
            }
        }
        $user_attributes = my_xml2array($user_attributes_xml);
        if (false === $user_attributes) {
            printError("Could not parse the answer returned by the
controller.");
            return false;
        }
        return $user_attributes;
    }
    // This function generates a basic login page containing a
      // form that allows the user to submit his credentials
      // back to this server. The page displays the SSID of
      // the service the user associated to. Otherwise the
```

```
      // page is pretty plain. Obviously a real login page
            // would have more content on it. This routine
            // highlights the critical aspects of composing the
            // login page so that when the user submits his
            // credentials all the information necessary to manage
            // the user's session will be on the page.
    function compose_login_page($ewc_ip, $ewc_port, $token, $dest,
          $user_attributes)
    {
        $SSID = get_value_by_path($user_attributes, 'response/ssid' );
        $template = "<!DOCTYPE html>
<html>
<head>
    <meta charset=\"ISO-8859-1\">
    <title>Please Login</title>
</head>
<body>
    <form id=\"Login\" name=\"Login\" method=\"post\" action=\"login.php\">
        <table border='0' width='800' height='310' cellpadding='0'
            cellspacing='0'>
            <tr>
                <td colspan=\"3\" height=\"100\"> </td>
            </tr>
            <tr>
                <td width='260' height='1' border='0'/>
                <td width='300' height='65'>
                    Please login to use '$SSID' network.</td>
                <td width='240' rowspan='5'> </td>
            </tr>
            <tr>
                <td align=\"right\"><b>User Name  </b>
                </td>
                <td height=\"28\">
                    <input type=\"text\" autocomplete=\"off\"
                        id=\"userid\" name=\"userid\" tabindex=\"1\">
                </td>
            </tr>
            <tr>
                <td align=\"right\"><b>Password  </b>
                </td>
                <td height=\"28\"><input type=\"password\" autocomplete=\"off
\"
                    id=\"passwd\" name=\"passwd\" tabindex=\"2\">
                </td>
            </tr>
            <tr>
                <td><br>
                </td>
                <td height=\"33\" valign=\"bottom\"><input type=\"submit\"
                    style=\"width: 100px\" value=\"Login\" tabindex=\"3\">
                </td>
            </tr>
        </table>
        <input type=\"hidden\" name=\"ewc_ip\" id=\"ewc_ip\" value=\"$ewc_ip
\"/>
        <input type=\"hidden\" name=\"ewc_port\" id=\"ewc_port\" value=
\"$ewc_port\"/>
        <input type=\"hidden\" name=\"token\" id=\"token\" value=\"$token\"/>
```

```
        <input type=\"hidden\" name=\"dest\" id=\"dest\" value=\"http://$dest
\" />
    </form>
</body>
</html>";
        return $template;
    }
?>
```

login.php

```
<?php
    // login.php - A Sample PHP script to authenticate &
    // authorize a user as part of an external captive
    // portal with internal authentication.
    //
    // This is a simple implementation of a script that
    // receives a user's credentials and forwards them
    // to a controller's web session control interface
    // The controller is responsible for authenticating
    // and authorizing the user, and for applying the
    // access control role selected for the user. The
    // controller notifies this script when the authentication
    // has completed. It tells this script whether the
    // authentication was successful.
    //
    // This script assumes that the credentials are
    // submitted on the form created by the example script
    // net-auth.php.
    //
    // This script is able to interact with the session
    // control interface regardless of the type of encryption
    // that is configured for it. This is for illustrative
    // purposes. In an actual deployment the script is likely
    // to only need to support one type of encryption or no
    // encryption at all.
    //
    // Assumptions
    // ===========
    // 1. The controller is configured to include its IP address
    //    and port in the redirection URL and the submitted
    //           login form contains that IP address and port. This
    //           allows the ECP to interact with more than 1
    //           controller.
    // 2. The variable sc_encryption is defined in php.ini. It
    //           can take one of three values:
    //    0 - no encryption
    //    1 - MD5 encryption
    //    2 - AES encryption
    //    If the variable is undefined then the script assumes
    //    no encryption is required.
    // 3. If encryption is required then sc_key is defined in
    //    php.ini. It is the same encryption key that is defined
    //    on the controller's WLAN Service's Captive Portal
    //    Configuration dialog box.
    // 4. Whether the script uses HTTP or HTTPS depends on the
    //    value of sc_use_https which must be defined
```

```
in                                                  php.ini.
    //    If the value of sc_use_https is 1 then the script uses
    //    HTTPS. If the configuration variable has any other
    //           value or is not defined then the script uses HTTP.
In     //          practice an actual site is going to settle on
using                              HTTP or HTTPS.
    //    The scripts can then assume that method is being used
    //    rather than looking up the method in php.ini.
    // 5. The WLAN Service that redirected to the captive portal
    //           is configured with at least 1 RADIUS server
for                                              authentication.
    //
    // The variables, sc_encryption, sc_key and sc_use_https are
    // user-defined variables. They must be created in php.ini by
    // the web server administrator.
    require_once("crypt_md5.php");
    require_once("crypt_aes.php");
    require_once("common_utilities.php");
    // The mainline begins here. The utilities are defined after
    // the mainline.
    // 1. Collect the parameters required to send the auth_user_xml.php
    //    request. Some of these attributes come from hidden fields on
    //    on the submitted form.
    $ewc_ip = trim($_REQUEST['ewc_ip']);
    $ewc_port = trim($_REQUEST['ewc_port']);
    $redirection = trim($_REQUEST['dest']);
    $token = trim($_REQUEST['token']);
    $username = (isset($_REQUEST['userid'])) ?
          trim($_REQUEST['userid']) : "";
    $passwd = (isset($_REQUEST['passwd'])) ?
          trim($_REQUEST['passwd']) : "";
    $mu_ip_addr = isset($_SERVER['REMOTE_ADDR']) ?
          $_SERVER['REMOTE_ADDR'] : "";
    if(!tokenCheck($token)) {
    printError("Error: <span style='color:red'>Failed to process the request:
incorrect token.</span>");
    exit;
    } else if(isset($ewc_port) && !is_numeric($ewc_port)) {
    printError("Error: <span style='color:red'>Failed to process the request:
incorrect EWC port.</span>");
    exit;
    }
    // 2. Send the user's credentials to the controller in the
    //    auth_user_xml.php request.
    $successful_authentication = auth_user($ewc_ip, $ewc_port,
       $username, $passwd, $token, $mu_ip_addr);
    // 3. If the user authenticated successfully redirect him or
    // her to the URL they were originally trying to get to. If
    // the user failed authentication auth_user composes & sends
    // an error message. So if the authentication fails,
just                    // exit.
    if ($successful_authentication) {
        header( 'Location: '.$redirection);
    }
    exit;
    // End of mainline.
    // A procedure that composes the auth_user_xml.php requests,
    // sends the request and wait for the reply from the controller.
```

```
function auth_user($ewc_ip, $ewc_port, $username, $passwd,
    $token, $mu_ip_addr) {
    if (get_cfg_var('sc_use_https')) {
        if (1 == get_cfg_var('sc_use_https')) {
            $protocol = "https://";
        } else {
            $protocol = "http://";
        }
    } else {
        $protocol = "http://";
    }
    $ewc_addr =
            $protocol.$ewc_ip.":".$ewc_port."/auth_user_xml.php?";
    // Determine the type of encryption to use: 1->Legacy; 2->AES
        // (configure in WLAN Service ECP configuration page). A real
        // ECP would only need to implement one of these.
    $cp_encryption = get_cfg_var('sc_encryption')?
             intval(get_cfg_var('sc_encryption')) : 0;
    $crypt_key = get_cfg_var('sc_key')? get_cfg_var('sc_key') : "";
    // Build the query string for the request
    switch ($cp_encryption)
    {
        case 1: /* MD5 Encryption selected */
            // Crypt_md5 is a php script function from the open
                    // source community
            $parms=make_query_string($token, $mu_ip_addr,
                $username, $passwd, ",");
            $query_string = "param=".bin2hex(crypt_md5($parms,
                        $crypt_key));
            break;
        case 2: /* AES Encryption selected */
            $parms=make_query_string($token, $mu_ip_addr,
                $username, $passwd, ",");
            $query_string = "param=".base64_url_encode(
                        crypt_aes($parms, $crypt_key));
            break;
        default: /* No encryption requested */
            $query_string = make_query_string($token,
                $mu_ip_addr, $username, addslashes($passwd),
                          "&");
    }
    // Now call approval.php to tell the controller the new settings
    // for the user session.
    $auth_user_response_xml =
            file_get_contents($ewc_addr.$query_string);
    if (false === $auth_user_response_xml) {
        printError("Could not access the controller to tell it”.
                  " to authorize you.".
            " Please try again in 5 minutes.");
        return false;
    }
    // Decrypt the reply if necessary
    if (1 == $cp_encryption) {
        // MD5 encryption selected
        $auth_user_response_xml =  decrypt_md5(
            hex2bin($auth_user_response_xml), $crypt_key);
    } else {
        // AES encryption selected
```

```
            if (2 == $cp_encryption) {
                $auth_user_response_xml = decrypt_aes(
                    base64_url_decode($auth_user_response_xml),
                                $crypt_key);
            }
        }
        // Massage the response to make it easier to process.
        $auth_user_response = my_xml2array($auth_user_response_xml);
        $ewc_token = get_value_by_path($auth_user_response,
                'response/token');
        $ewc_status = get_value_by_path($auth_user_response,
                'response/status');
        if( ($token == $ewc_token) && (1 == $ewc_status) ){
            // Success path exit.
             return true;
        } else {
            // Something went wrong. Output a suitable error message.
               if ($token != $ewc_token) {
                   // Somehow the controller responded to the wrong
// request.
                  printError("Error: <span".
                            " style='color:red'>Unexpected".
                            " response from access controller. ".
                            " Please try again in 5 minutes.</span>");
              } else {
                  // Print an error message corresponding to the
                        // return code.
                  printError("Error: ".code_2_message($ewc_status,
                            'content'));
              }
              return false;
          }
    }
    function make_query_string($token, $mu_ip_addr, $userid,
        $passwd, $separator)
    {
        $q_str = "token=".$token;
        $q_str .= $separator."username=".$userid;
        $q_str .= $separator."password=".$passwd;
             if (is_not_empty($mu_ip_addr)) {
            $q_str .= $separator."mu_ip_addr=".$mu_ip_addr;
        }
        return $q_str;
    }
?>
```

common_utilities.php

```
<?php
    // A library of utilities that can be used by PHP scripts
    // comprising an external captive portal.
    // A utility that translates error codes to error messages.
    function code_2_message($code, $content_type)
    {
        $errMsgList = array (
            0 =>
            array (
```

```
              'label' => 'Invalid',
              'content' => '<span style=\'color:red\'>Empty id / password not
allowed. Please try again.</span>',
            ),
            1 =>
            array (
              'label' => 'Success',
              'content' => 'Success',
            ),
            2 =>
            array (
              'label' => 'Access Fail',
              'content' => '<span style=\'color:red\'>Userid or password
incorrect. Please try again.</span>',
            ),
            3 =>
            array (
              'label' => 'Fail',
              'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
            ),
            4 =>
            array (
              'label' => 'Timeout',
              'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
            ),
            5 =>
            array (
              'label' => 'RADIUS shared security key fail',
              'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
            ),
            6 =>
            array (
              'label' => 'RADIUS internal error',
              'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
            ),
            7 =>
            array (
              'label' => 'Max RADIUS login fail',
              'content' => '<span style=\'color:red\'>Too many users trying
to login at the same time.Please try again later.</span>',
            ),
            8 =>
            array (
              'label' => 'Invalid Login parameters',
              'content' => '<span style=\'color:red\'>Userid or password
incorrect. Please try again.</span>',
            ),
            9 =>
            array (
              'label' => 'General failure',
```

```
              'content' => '<span style=\'color:red\'>A problem has occurred
while trying to validate your userid & password.<br>Please contact your
system administrator.</span>',
            ),
            14 =>
            array (
              'label' => 'Invalid third party parameters',
              'content' => '<span style=\'color:red\'>Invalid third party
parameters.</span>',
            ),
            15 =>
            array (
              'label' => 'Authentication in progress failure',
              'content' => '<span style=\'color:red\'>Authentication is in
progress.</span>',
            ),
            17 =>
            array (
              'label' => 'Max concurrent session failure',
              'content' => '<span style=\'color:red\'>Login rejected because
the maximum number of concurrent sessions for this set of credentials has
been reached. Please try again later.</span>',
            ),
            18 =>
            array (
              'label' => 'Identified session not found',
              'content' => '<span style=\'color:red\'>Login failed because
could not find a session for the specified identifiers.</span>'
            ),
            99 =>
            array (
              'label' => 'Timeout while trying to authorize a session',
              'content' => '<span style=\'color:red\'>Login failed because
because the controller took too long to authorize the session.</span>'
            )
          );
          return (isset($errMsgList[$code])) ?
$errMsgList[$code][$content_type] :
              "Unrecognized error code: ".$code;
    }
    // General purpose error reporting procedure.
    function printError($errorMsg) {
        header('Content-type: text/html; charset=iso-8859-1');
        print "<html>\n<head><title>Error</title></head><body>\n<p>\n$errorMsg
\n</p>\n</body>\n</html>\n";
    }
    // Use base64 url safe encode/decode when dealing
    // with AES-encrypted strings.
    // encode: '+'=>'-', '/' => '_' , '=' => '!'
    function base64_url_encode($input) {
        return strtr(base64_encode($input), '+/=', '-_!');
    }
    // Decode: '-'=>'+', '_' => '/' , '!' => '='
    function base64_url_decode($input) {
        return base64_decode(strtr($input, '-_!', '+/='));
    }
    // xml parsing functions
    function my_xml2array($contents)
```

```
    {
        $xml_values = array();
        if (! isset($contents)) {
            return false;
        }
        $parser = xml_parser_create('');
        if(!$parser) {
            return false;
        }
        xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING,
                  'UTF-8');
        xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
        xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
        xml_parse_into_struct($parser, trim($contents), $xml_values);
        xml_parser_free($parser);
        if (!$xml_values) {
            return array();
        }
        $xml_array = array();
        $last_tag_ar =& $xml_array;
        $parents = array();
        $last_counter_in_tag = array(1=>0);
        foreach ($xml_values as $data)
        {
            switch($data['type'])
            {
                case 'open':
                    $last_counter_in_tag[$data['level']+1] = 0;
                    $new_tag = array('name' => $data['tag']);
                    if(isset($data['attributes']))
                    $new_tag['attributes'] = $data['attributes'];
                    if(isset($data['value']) &&
                                trim($data['value']))
                    $new_tag['value'] = trim($data['value']);
                    $last_tag_ar[$last_counter_in_tag[
$data['level']]] = $new_tag;
                    $parents[$data['level']] =& $last_tag_ar;
                    $last_tag_ar =& $last_tag_ar[
                        $last_counter_in_tag[$data['level']]++];
                    break;
                case 'complete':
                    $new_tag = array('name' => $data['tag']);
                    if(isset($data['attributes']))
                    $new_tag['attributes'] = $data['attributes'];
                    if(isset($data['value']) &&
                        trim($data['value']))
                    $new_tag['value'] = trim($data['value']);
                    $last_count = count($last_tag_ar)-1;
                        $last_tag_ar[ $last_counter_in_tag[$data[
'level' ]]++ ] = $new_tag;
                    break;
                case 'close':
                    $last_tag_ar =& $parents[$data['level']];
                    break;
                default:
                    break;
            };
        }
```

```
        return $xml_array;
    }
    function get_value_by_path($__xml_tree, $__tag_path)
    {
        $tmp_arr =& $__xml_tree;
        $tag_path = explode('/', $__tag_path);
        foreach($tag_path as $tag_name)
        {
            $res = false;
            foreach($tmp_arr as $key => $node)
            {
                if(is_int($key) && $node['name'] == $tag_name)
                {
                    $tmp_arr = $node;
                    $res = true;
                    break;
                }
            }
            if(!$res) {
                return false;
            }
        }
        if( isset($tmp_arr['value']) ) {
            return $tmp_arr['value'];
        } else {
            return null;
        }
    }
    function is_not_empty($string) {
        return (isset($string) && (0 < strlen($string)));
    }
    //check token format
    function tokenCheck($val){
        return preg_match('/^([a-zA-Z0-9-_!]){0,24}$/', $val);
    }
    //check the mac address
    function macCheck($val){
        return preg_match("/^[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-
Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}$/", $val);
    }
    //encode the input string to avoid script attack
    function convertUrlParam($input) {
        return htmlentities($input, ENT_QUOTES);
    }
?>
```

crypt_aes.php

```
<?php
    // A wrapper around the calls to Openssl encryption and decryption.
      // PHP must be compiled with Openssl support enabled. Openssl must
      // be installed on the web server's host. On Windows the ExtensionList
      // section of php.ini must contain the line "extension=php_openssl.dll"
      // Version 5.3.19, which is installed by version 4.5 of the Microsoft
      // Web Platform Installer is preconfigured to enable Openssl support.
      // For other platforms please consult the documentation on Openssl at
      // www.php.net.
```

```
    define("AES_BLOCK_LEN", 16);
    // Use the following PKCS#7 padding (RFC 5652) if the chosen
    // encryption library does not add PKCS#7 padding
    // automatically. Openssl does so crypt_aes.php does not call
    // aes_cbc_pad.
    function aes_cbc_pad($message)
    {
        $length = strlen($message);
        $pad = AES_BLOCK_LEN - ($length % AES_BLOCK_LEN);
        return (AES_BLOCK_LEN == $pad) ? message : str_pad($message,
        $length + $pad, chr($pad));
    }
    // A wrapper around openssl AES encryption. The wrapper sets the
    // IV to 16 binary zeroes and selects the encryption algorithm based
    // on the key length. The algorithm is always a variant of AES in
    // Cipher Block Chaining (CBC) mode.
    function crypt_aes($message, $shared_key)
    {
        $iv = pack('H*', "00000000000000000000000000000000");
        $method = (strlen($shared_key) >= 32) ? 'aes-256-cbc' :
                'aes-128-cbc';
        return openssl_encrypt($message, $method, $shared_key,
                true, $iv);
    }
    // A wrapper around openssl AES decryption. The wrapper sets the
    // IV to 16 binary zeroes and selects the decryption algorithm based
    // on the key length. The algorithm is always a variant of AES in
    // Cipher Block Chaining (CBC) mode.
    function decrypt_aes($message, $shared_key)
    {
        $iv = pack('H*', "00000000000000000000000000000000");
        $method = (strlen($shared_key) >= 32) ? 'aes-256-cbc' :
                'aes-128-cbc';
        return openssl_decrypt($message, $method, $shared_key,
                true, $iv);
    }
?>
```

crypt_md5.php

```
// Copyright: hkmaly@matfyz.cz
    // http://adela.karlin.mff.cuni.cz/~hkmaly/php/
    // Extremenetworks.com: Added minor modifications to
    // to deal with input that has a length that
    // is not a multiple of 16.
    function hex2bin($data)
    {
        $len = strlen($data);
        return pack("H" . $len, $data);
    }
    function bytexor($a,$b,$l)
    {
        $c="";
        for($i=0;$i<$l;$i++) { $c.=$a{$i}^$b{$i}; }
        return($c);
    }
    function binmd5($val)
```

```
    {
        return(pack("H*",md5($val)));
    }
    function decrypt_md5($msg,$heslo)
    {
        $key=$heslo;$sifra="";
        $key1=binmd5($key);
        $msglen=strlen($msg);
        while($msglen >= 16) {
            $m=substr($msg,0,16);
            $msg=substr($msg,16);
            $msglen -= 16;
            $sifra.=$m=bytexor($m,$key1,16);
            $key1=binmd5($key.$key1.$m);
        }
    if ($msglen > 0) {
        $sifra.=bytexor($msg,$key1,$msglen);
    }
        return($sifra);
    }
    function crypt_md5($msg,$heslo)
    {
        $key=$heslo;$sifra="";
        $key1=binmd5($key);
        $msglen=strlen($msg);
        while($msglen >= 16) {
            $m=substr($msg,0,16);
            $msg=substr($msg,16);
            $msglen -= 16;
            $sifra.=bytexor($m,$key1,16);
            $key1=binmd5($key.$key1.$m);
        }
    if ($msglen > 0) {
        $sifra.=bytexor($msg,$key1,$msglen);
    }
        return($sifra);
    }
?>
```

E PHP External Captive Portal, Controller's Firewall Friendly API

net-auth.php
login.php
common_utilities.php
crypt_aws_s4.php
ffecp-config.php

This appendix contains five files that serve as an example of how to build an External Captive Portal that makes use of the controller's Firewall-Friendly External Captive Portal Interface. The files presented are:

- net-auth.php

  Receives redirected requests from browsers trying to access web sites, verifies that the redirect was sent from the controller and if so, displays a suitable login page.
- login.php

  This script gets invoked as a consequence of a browser submitting the login form created by net-auth.php. The script authenticates the station in whatever way it likes. If the station is authorized the script selects a maximum session duration and an access control role for the station. It then redirects the station's browser back to a web server on the controller, using a URI that contains the access control role, the maximum session duration, other data required by the controller, and a signature.
- crypt_aws_s4.php

  This file contains the code that verifies the signatures on received URLs and that signs the URLs that redirect the station back to the controller.
- common_utilities.php

  Utilities used by various ECP scripts
- ffecp-config.php

  Contains the main statically configured parameters that the application uses to verify signed URLs and to create signed URLs.

net-auth.php

```
<?php
  // net-auth.php
  // This is a simple implementation of a script that
  // receives HTTP requests that have been redirected
  // by a controller configured with "Firewall-Friendly
  // External Captive Portal" support enabled.
  // This script is responsible for collecting critical
  // information from the redirection, such as the
```

```
// session token and for constructing the login page
// for the user. The script reads the VNS attribute
// from the redirected request so that the script can
// display it on the login page.
//
// The script expects the controller to sign the
// its redirection response. The script validates the
// signature. If the signature is valid it displays
// the login page. Otherwise it displays an error page.
//
// Assumptions
// ===========
// 1. The controller is configured to include its IP address
//    and port in the redirection URL.
// 2. The controller is configured to sign its redirection
//    responses using the Amazon S3 version 4 signature
//    algorithm (as of May 2014).
// 3. The controller is configured to include the VNS in its
// redirection response.
// 4. This application assumes that the identity & shared key
// key pairs it is allowed to use are stored in an associative
// array. It also assumes that some configuration options such
// as the 'service' and 'region' are stored in another associative
// array. Real applications might retrieve this information from
// databases or configuration files.

require_once("ffecp_config.php");
require_once("crypt_aws_s4.php");
require_once("common_utilities.php");

// Mainline processing starts here. Utilities are defined after
// the mainline.
// 1. Verify that the request has all necessary fields
// and a valid signature.
$rc = SimpleAws::verifyAwsUrlSignature(getURL($_SERVER),
 $awsKeyPairs);
if (SimpleAws::AWS4_ERROR_NONE != $rc) {
    printError(SimpleAws::getAwsError($rc));
    exit;
}
// Determine which controller interface to interact with
if(isset($_REQUEST['hwc_ip']) && isset($_REQUEST['hwc_port'])) {
  //BM IP address and port is enabled
  $ewc_ip = trim($_REQUEST['hwc_ip']);
  $ewc_port = trim($_REQUEST['hwc_port']);
} else {
  // The controller has not been configured as expected. It did not
  // include its address and port on the redirection URL. This is
  // easy to fix but all this program can do is report the error.
  printError("Controller must be configured to include its IP " .
  "address & port in the request.");
  exit;
}
// Collect the data required by the login page and
// subsequent authentication.
$dest = isset($_REQUEST['dest']) ? $_REQUEST['dest'] : "";
$bssid = isset($_REQUEST['bssid']) ? $_REQUEST['bssid'] : "";
$wlan = isset($_REQUEST['wlan']) ? $_REQUEST['wlan'] : "";
```

```
  $vns = isset($_REQUEST['vns']) ? $_REQUEST['vns'] : "";
  $mu_mac = isset($_REQUEST['mac']) ? $_REQUEST['mac'] : "";
  $ap_name = isset($_REQUEST['ap']) ? $_REQUEST['ap'] : "";
  $token = isset($_REQUEST['token']) ? $_REQUEST['token'] : "";
  if(!tokenCheck($token)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect token.</span>");
      exit;
  } else if(isset($ewc_port) && !is_numeric($ewc_port)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect EWC port.</span>");
      exit;
  }else if($mu_mac && !macCheck($mu_mac)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect client MAC address.</span>");
      exit;
  } else if(!empty($wlan) && !is_numeric($wlan)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect WLAN.</span>");
      exit;
  }
   //escape the parameters
  $dest =convertUrlParam($dest);
  $bssid = convertUrlParam($bssid);
  $vns = convertUrlParam($vns);
  $ap_name = convertUrlParam($ap_name);
  // 3. Compose the login page and send it to the user. The page
  // is used to store session data, but this could have been
  // stored in the user session variable or in cookies.
  print compose_login_page($ewc_ip, $ewc_port, $token, $dest,
           $wlan, $vns, $bssid, $mu_mac, $ap_name);
    // 4. And exit. This script is finished executing.
  exit;
  // End of mainline
  // A function that reconstructs the URL that the
  // station was trying to Get, from the variables
  // generated by the PHP runtime.
  function getURL($data) {
    $ssl = (!empty($data['HTTPS']) && $data['HTTPS'] == 'on') ? true:false;
    $protocol = $ssl ? "https" : "http";
    $port = $data['SERVER_PORT'];
    $port = ((!$ssl && $port=='80') || ($ssl && $port=='443')) ? '' :
   ':'.$port;
    $host = isset($data['HTTP_HOST']) ? $data['HTTP_HOST'] :
   $data['SERVER_NAME'] . $port;
    return $protocol . '://' . $host . $data['REQUEST_URI'];
  }
  // This function generates a basic login page containing a form
  // that allows the user to submit his credentials back to this
  // server. The page displays the name of the VNS (service) the user
  // associated to. Otherwise the page is pretty plain. Obviously
  // a real login page would have more content on it. This routine
  // highlights the critical aspects of composing the login page so
  // that when the user submits his credentials all the information
  // necessary to manage the user's session will be on the page.
  function compose_login_page($ewc_ip, $ewc_port, $token, $dest,
          $wlan, $vns, $bssid, $mu_mac, $ap_name)
   {
```

```
    $template = "<!DOCTYPE html>
<html>
<head>
  <meta charset=\"ISO-8859-1\">
  <title>Please Login</title>
</head>
<body>
  <form id=\"Login\" name=\"Login\" method=\"post\" action=\"login.php\">
    <table border='0' width='800' height='310' cellpadding='0'
      cellspacing='0'>
      <tr>
        <td colspan=\"3\" height=\"100\"> </td>
      </tr>
      <tr>
        <td width='260' height='1' border='0'/>
        <td width='300' height='65'>
          Please login to use '$vns' network.</td>
        <td width='240' rowspan='5'> </td>
      </tr>
      <tr>
        <td align=\"right\"><b>User Name  </b>
        </td>
        <td height=\"28\">
          <input type=\"text\" autocomplete=\"off\"
            id=\"userid\" name=\"userid\"
tabindex=\"1\">
        </td>
      </tr>
      <tr>
        <td align=\"right\"><b>Password  </b>
        </td>
        <td height=\"28\"><input type=\"password\"
autocomplete=\"off\"
          id=\"passwd\" name=\"passwd\" tabindex=\"2\">
        </td>
      </tr>
      <tr>
        <td><br>
        </td>
        <td height=\"33\" valign=\"bottom\"><input
type=\"submit\"
          style=\"width: 100px\" value=\"Login\"
tabindex=\"3\">
        </td>
      </tr>
    </table>
    <input type=\"hidden\" name=\"ewc_ip\" id=\"ewc_ip\"
value=\"$ewc_ip\"/>
    <input type=\"hidden\" name=\"ewc_port\" id=\"ewc_port\"
value=\"$ewc_port\"/>
    <input type=\"hidden\" name=\"token\" id=\"token\"
value=\"$token\"/>
    <input type=\"hidden\" name=\"dest\" id=\"dest\"
value=\"http://$dest\" />
    <input type=\"hidden\" name=\"wlan\" id=\"wlan\"
value=\"$wlan\" />
    <input type=\"hidden\" name=\"mu_mac\" id=\"mu_mac\"
value=\"$mu_mac\" />
```

```
    <input type=\"hidden\" name=\"bssid\" id=\"bssid\"
 value=\"$bssid\" />
    <input type=\"hidden\" name=\"ap\" id=\"ap\"
 value=\"$ap_name\" />
  </form>
 </body>
 </html>";
    return $template;
  }
 ?>
```

login.php

```
<?php
  // login.php
  // This is a simple implementation of a script that
  // receives a user's credentials, authenticates the
  // credentials, selects an access control role for
  // the user, then redirects the user back to the
  // controller using a signed URL containing the selected
  // access control role.
  // This script assumes that the credentials are
  // submitted on the form created by the example script
  // net-auth.php.
  //
  //
  // Assumptions
  // ===========
  // 1. The controller is configured to include its IP address
  //    and port in the redirection URL and the submitted login
  //    form contains that IP address and port. This allows the
  //    ECP to interact with more than 1 controller.
  // 2. Whether the script uses HTTP or HTTPS in its redirection
  //    response depends on the value of use_https
  //    which must be defined in php.ini.
  //    If the value of use_https is 1 then the script uses
  //    HTTPS. If the configuration variable has any other value
  //    or is not defined then the script uses HTTP. In practice
  //    an actual site is going to settle on using HTTP or HTTPS.
  //    The scripts can then assume that method is being used
  //    rather than looking up the method in php.ini.
  // The use_https is a user-
  // defined variables. They must be created in php.ini by the
  // web server administrator.
  require_once("ffecp_config.php");
  require_once("crypt_aws_s4.php");
  require_once("common_utilities.php");
  // Some local constants
  const EWC_HTTP_REQ = "http://";
  const EWC_HTTPS_REQ = "https://";
  const EWC_REDIRECT_TARGET = "/ext_approval.php?";
  // The mainline begins here. The utilities are defined after the
  // mainline.
  // 1. Collect the parameters submitted on the login form.
  //    Some of these attributes come from hidden fields.
  $ewc_ip = trim($_REQUEST['ewc_ip']);
  $ewc_port = trim($_REQUEST['ewc_port']);
```

```
  $dest = trim($_REQUEST['dest']);
  $token = trim($_REQUEST['token']);
  $username = (isset($_REQUEST['userid'])) ?
    trim($_REQUEST['userid']) : "";
  $passwd = (isset($_REQUEST['passwd'])) ?
    trim($_REQUEST['passwd']) : "";
  $wlan = isset($_REQUEST['wlan']) ?
    trim($_REQUEST['wlan']) : "";
  if(!tokenCheck($token)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect token.</span>");
      exit;
  } else if(isset($ewc_port) && !is_numeric($ewc_port)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect EWC port.</span>");
      exit;
  } else if(!empty($wlan) && !is_numeric($wlan)) {
      printError("Error: <span style='color:red'>Failed to process the
request: incorrect WLAN.</span>");
      exit;
  }
  // For this example the maximum duration of any user's
  // session will be 36000 seconds. The session is terminated
  // no later than this time. After the session is terminated
  // the user can access the network but will be unauthenticated
  // again.
  $max_duration = 36000;
  // 2. Authenticate the user & select an appropriate role.
  //    Selecting the role is optional. If one is not specified
  //    for the controller the controller will apply the default
  //    authenticated role of the WLAN Service the user is
  //    accessing.
  $assigned_role = authenticate($username, $passwd);
  if (false === $assigned_role) {
      // Failed to authenticate the user.
      // Authenticate prints the error message for
      // the browser. We just exit here.
      exit;
  }
  // 3. Tell the controller that the user is authenticated
  //    and tell it which role to apply to the user.
  //    3.a Build the URL that needs to be signed.
  $pUrl = makeUnsignedUrl($ewc_ip, $ewc_port, isHttps(), $token,
          $username, $wlan, $assigned_role, $dest,
          $max_duration);
  // 3.b Sign the URL. Otherwise the role and session
  //     duration options will be ignored by the controller.
  $redirection = SimpleAws::createPresignedUrl(
    $pUrl, 'BigAuthInc', $awsKeyPairs['BigAuthInc'],
      $awsConfig['region'], $awsConfig['service'],
      $awsConfig['expires']);
  if (null == $redirection) {
      // Quietly exit. createPresignedUrl has already
      // reported an error to the browser.
    exit;
  }
  header( 'Location: '.$redirection);
  exit;
```

```
// End of mainline.
// A method that validates the user's credentials and
// returns the role to apply to the user. In some cases
// this routine might also return the maximum session
// duration in seconds.
//
// For purposes of this example this procedure is
// not much more than a stub. The stub can be replaced
// by any authentication method, such as sending access
// requests to a backend RADIUS server, or performing
// a lookup in an application credential database.
function authenticate($userid, $passwd) {
    if (("" == $userid) || ("" == $passwd)) {
        printError("Invalid Userid or Password. ".
                "Please press the 'Back' button and try again.");
        // If you generate another login page for the user
        // be sure to copy the ewc_ip address, ewc_port,
        // token and dest attributes from the submitted
        // login form to the login page.
        return false;
    } else {
        // Return the name of a role to be applied
        // to the station. The role must be defined on
        // the controller or it will substitute the
        // default authenticated role of the VNS the
        // user is logging into.
        // For purposes of this example assume all
        // authenticated users get the 'Guest_Access' role.
        return "Guest_Access";
    }
}
/**
 * A function that decides whether to send
 * to use HTTP or HTTPS in the redirect to
 * the controller. This example just uses
 * a php.ini user configuration variable
 * to decide.
 */
function isHttps() {
    if (get_cfg_var('use_https')) {
        if (1 == get_cfg_var('use_https')) {
            $useHttps = true;
        } else {
            $useHttps = false;
        }
    } else {
        $useHttps = false;
    }
    return $useHttps;
}
/**
 * A method that assembles an unsigned URL out of the
 * the input from the user's succesful login
 * @param string  $ewc_ip     IP or FQDN of controller
 * @param int     $ewc_port   Port on controller to receive redirection
 * @param bool    $useHttps   Whether the redirect uses HTTP or HTTPS
 * @param string  $token      Identifier for the station's session
 * @param string  $username   The name the station's user logged in with
```

```
     * @param int     $wlanid     Identifier for the WLAN the station is using
     * @param string  $assigned_role Name of the access control role to assign
     * @param string  $dest       The URL the station was trying to get to
     * @param int     $max_duration The maximum length of the station's session.
     */
    function makeUnsignedUrl($ewc_ip, $ewc_port, $useHttps, $token,
            $username, $wlanid, $assigned_role, $dest,
            $max_duration) {
        $redirectUrl = ($useHttps ? EWC_HTTPS_REQ : EWC_HTTP_REQ)
            .$ewc_ip;
        if ((80 != $ewc_port) && (443 != $ewc_port)) {
            $redirectUrl .= ":".$ewc_port;
        }
        $redirectUrl .= EWC_REDIRECT_TARGET
            .'token='. rawurlencode($token)
            .'&wlan='.rawurlencode($wlanid)
            .'&username='.rawurlencode($username)
            .(is_not_empty($dest) ?'&dest='.rawurlencode($dest):'')
            .(is_not_empty($assigned_role) ? '&role='.
                    rawurlencode($assigned_role):'')
            .(is_not_empty($max_duration) ?'&opt27='.$max_duration:'');
        return $redirectUrl;
    }
    ?>
```

common_utilities.php

```
<?php
  // A library of utilities that can be used by PHP scripts
  // comprising an external captive portal.
  // A utility that translates error codes to error messages.
  function code_2_message($code, $content_type)
  {
    $errMsgList = array (
      0 =>
      array (
        'label' => 'Invalid',
        'content' => '<span style=\'color:red\'>Empty id /
password not allowed. Please try again.</span>'
      ),
      1 =>
      array (
        'label' => 'Success',
        'content' => 'Success',
      ),
      2 =>
      array (
        'label' => 'Access Fail',
        'content' => '<span style=\'color:red\'>Userid or
password incorrect. Please try again.</span>',
      ),
      3 =>
      array (
        'label' => 'Fail',
        'content' => '<span style=\'color:red\'>A problem has
occurred while trying to validate your userid & password.<br>Please contact
your system administrator.</span>',
```

```
      ),
      4 =>
      array (
        'label' => 'Timeout',
        'content' => '<span style=\'color:red\'>A problem has
occurred while trying to validate your userid & password.<br>Please contact
your system administrator.</span>',
      ),
      5 =>
      array (
        'label' => 'RADIUS shared security key fail',
        'content' => '<span style=\'color:red\'>A problem has
occurred while trying to validate your userid & password.<br>Please contact
your system administrator.</span>',
      ),
      6 =>
      array (
        'label' => 'RADIUS internal error',
        'content' => '<span style=\'color:red\'>A problem has
occurred while trying to validate your userid & password.<br>Please contact
your system administrator.</span>',
      ),
      7 =>
      array (
        'label' => 'Max RADIUS login fail',
        'content' => '<span style=\'color:red\'>Too many users
trying to login at the same time.Please try again later.</span>',
      ),
      8 =>
      array (
        'label' => 'Invalid Login parameters',
        'content' => '<span style=\'color:red\'>Userid or
password incorrect. Please try again.</span>',
      ),
      9 =>
      array (
        'label' => 'General failure',
        'content' => '<span style=\'color:red\'>A problem has
occurred while trying to validate your userid & password.<br>Please contact
your system administrator.</span>',
      ),
      14 =>
      array (
        'label' => 'Invalid third party parameters',
        'content' => '<span style=\'color:red\'>Invalid third
party parameters.</span>',
      ),
      15 =>
      array (
        'label' => 'Authentication in progress failure',
        'content' => '<span style=\'color:red\'>Authentication is
in progress.</span>',
      ),
      17 =>
      array (
        'label' => 'Max concurrent session failure',
        'content' => '<span style=\'color:red\'>Login rejected
because the maximum number of concurrent sessions for this set of credentials
```

```
has been reached. Please try again later.</span>',
      ),
      18 =>
      array (
        'label' => 'Identified session not found',
        'content' => '<span style=\'color:red\'>Login failed
because could not find a session for the specified identifiers.</span>'
      ),
      99 =>
      array (
        'label' => 'Timeout while trying to authorize a session',
        'content' => '<span style=\'color:red\'>Login failed
because because the controller took too long to authorize the
session.</span>'
      )
      );
      return (isset($errMsgList[$code])) ?
        $errMsgList[$code][$content_type] :
        "Unrecognized error code: ".$code;
  }
  // General purpose error reporting procedure.
  function printError($errorMsg) {
    header('Content-type: text/html; charset=iso-8859-1');
    print
"<html>\n<head><title>Error</title></head><body>\n<p>\n$errorMsg\n</p>\n</bod
y>\n</html>\n";
  }
  // Use base64 url safe encode/decode when dealing
  // with AES-encrypted strings.
  // encode: '+'=>'-', '/' => '_' , '=' => '!'
  function base64_url_encode($input) {
    return strtr(base64_encode($input), '+/=', '-_!');
  }
  // Decode: '-'=>'+', '_' => '/' , '!' => '='
  function base64_url_decode($input) {
    return base64_decode(strtr($input, '-_!', '+/='));
  }
  // xml parsing functions
  function my_xml2array($contents)
  {
    $xml_values = array();
    if (! isset($contents)) {
      return false;
    }
    $parser = xml_parser_create('');
    if(!$parser) {
      return false;
    }
    xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING,
                  'UTF-8');
    xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
    xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
    xml_parse_into_struct($parser, trim($contents), $xml_values);
    xml_parser_free($parser);
    if (!$xml_values) {
      return array();
    }
    $xml_array = array();
```

```
    $last_tag_ar =& $xml_array;
    $parents = array();
    $last_counter_in_tag = array(1=>0);
    foreach ($xml_values as $data)
    {
      switch($data['type'])
      {
        case 'open':
          $last_counter_in_tag[$data['level']+1] = 0;
          $new_tag = array('name' => $data['tag']);
          if(isset($data['attributes']))
          $new_tag['attributes'] = $data['attributes'];
          if(isset($data['value']) && trim($data['value']))
          $new_tag['value'] = trim($data['value']);
          $last_tag_ar[$last_counter_in_tag[
$data['level']]] = $new_tag;
          $parents[$data['level']] =& $last_tag_ar;
          $last_tag_ar =& $last_tag_ar[
            $last_counter_in_tag[$data['level']]++];
          break;
        case 'complete':
          $new_tag = array('name' => $data['tag']);
          if(isset($data['attributes']))
          $new_tag['attributes'] = $data['attributes'];
          if(isset($data['value']) &&
            trim($data['value']))
          $new_tag['value'] = trim($data['value']);
          $last_count = count($last_tag_ar)-1;
            $last_tag_ar[ $last_counter_in_tag[$data[
'level' ]]++ ] = $new_tag;
          break;
        case 'close':
          $last_tag_ar =& $parents[$data['level']];
          break;
        default:
          break;
      };
    }
    return $xml_array;
  }
  function get_value_by_path($__xml_tree, $__tag_path)
  {
    $tmp_arr =& $__xml_tree;
    $tag_path = explode('/', $__tag_path);
    foreach($tag_path as $tag_name)
    {
      $res = false;
      foreach($tmp_arr as $key => $node)
      {
        if(is_int($key) && $node['name'] == $tag_name)
        {
          $tmp_arr = $node;
          $res = true;
          break;
        }
      }
      if(!$res) {
        return false;
```

```
      }
    }
    if( isset($tmp_arr['value']) ) {
      return $tmp_arr['value'];
    } else {
      return null;
    }
  }
  function is_not_empty($string) {
    return (isset($string) && (0 < strlen($string)));
  }
  //check token format
    function tokenCheck($val){
        return preg_match('/^([a-zA-Z0-9-_!]){0,24}$/', $val);
    }
    //check the mac address
    function macCheck($val){
        return preg_match("/^[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-
Fa-f0-9]{2}:[A-Fa-f0-9]{2}:[A-Fa-f0-9]{2}$/", $val);
    }
    //encode the input string to avoid script attack
    function convertUrlParam($input) {
        return htmlentities($input, ENT_QUOTES);
  }
?>
```

crypt_aws_s4.php

```
<?php
class SimpleAws {
    const     AWS4_ERROR_NONE=0;
    const     AWS4_ERROR_NULL_INPUT=1;
    const     AWS4_ERROR_INPUT_BUFFER_TOO_SMALL=2;
    const     AWS4_ERROR_INVALID_PROTOCOL=3;
    const     AWS4_ERROR_INPUT_URL_TOO_BIG=4;
    const     AWS4_ERROR_INPUT_ID_TOO_BIG=5;
    const     AWS4_ERROR_INPUT_KEY_TOO_BIG=6;
    const     AWS4_ERROR_INVALID_REGION=7;
    const     AWS4_ERROR_INVALID_SIGNATURE=8;
    const     AWS4_ERROR_MISSING_QUERY=9;
    const     AWS4_ERROR_MISSING_QUERY_DATE=10;
    const    AWS4_ERROR_MISSING_QUERY_SIGNED_HEADERS=11;
    const    AWS4_ERROR_MISSING_QUERY_EXPIRES=12;
    const    AWS4_ERROR_MISSING_QUERY_SIGNATURE=13;
    const    AWS4_ERROR_MISSING_QUERY_CREDENTIAL=14;
    const    AWS4_ERROR_MISSING_QUERY_ALGORITHM=15;
    const    AWS4_ERROR_MISSING_QUERY_PARAMS=16;
    const    AWS4_ERROR_MISSING_CRED_PARAMS=17;
   const     AWS4_ERROR_STALE_REQUEST=2001;
   const     AWS4_ERROR_UNKNOWN_IDENTITY=2002;
    const     AWS4_EXTREME_REQUEST="aws4_request";
    const     AWS4_MAX_URL_SIZE= 512;
    const     AWS4_HTTP_REQ = "http://";
    const     AWS4_HTTPS_REQ= "https://";
    const     AWS4_MANDATORY_CRED_PARAMS = 4;
    /**
     * Method to verify the AWS signature based on given full URL address.
```

```
         *
     * @param string $pUrl
     * @param array $awsKeyPairs identity, shared secret key pairs
     * @return AWS error code
     */
    public static function verifyAwsUrlSignature($pUrl,
        $awsKeyPairs) {
          // Perform basic validation
        if($pUrl==NULL) {
            return self::AWS4_ERROR_NULL_INPUT;
        }
        if (2*self::AWS4_MAX_URL_SIZE < strlen($pUrl)) {
            return self::AWS4_ERROR_INPUT_URL_TOO_BIG;
        }
        if(stripos($pUrl, self::AWS4_HTTP_REQ)!=0 || stripos($pUrl,
self::AWS4_HTTPS_REQ)!=0) {
            return self::AWS4_ERROR_INVALID_PROTOCOL;
        }
        $urlParams = parse_url($pUrl);
        if (!isset($urlParams['query'])) {
            return self::AWS4_ERROR_MISSING_QUERY;
        }
        $queryParams = explode("&", $urlParams['query']);
        foreach($queryParams AS $el) {
            $arr = explode("=", $el);
            $q[$arr[0]] = $arr[1];
        }
        $valResult = self::validateQueryParms($q);
        if (self::AWS4_ERROR_NONE != $valResult) {
            return $valResult;
        }
        // Done with the basic validations.
        $date = $q['X-Amz-Date'];
        $sign = $q['X-Amz-Signature'];
        $credentVal = rawurldecode($q['X-Amz-Credential']);
        ksort($q);
        // Remove the signature from the list of parameters over
        // which the signature will be recomputed.
        unset($q['X-Amz-Signature']);
        $credentAttrs = explode("/", $credentVal);
        $pKey = $credentAttrs[0];
        if (self::AWS4_MAX_URL_SIZE < strlen($pKey)) {
            return self::AWS4_ERROR_INPUT_KEY_TOO_BIG;
        }
        if(self::AWS4_MANDATORY_CRED_PARAMS > count($credentAttrs)) {
            return self::AWS4_ERROR_MISSING_CRED_PARAMS;
        }
       if (!isset($awsKeyPairs[$pKey])) {
            return self::AWS4_ERROR_UNKNOWN_IDENTITY;
       }
        $scope = $credentAttrs[1]."/".$credentAttrs[2]."/"
         .$credentAttrs[3]."/".$credentAttrs[4];
      $port = $urlParams['port'];
       $host = strtolower($urlParams['host']);
        if($port && (($urlParams['scheme']=='https' && $port !=
         443)||($urlParams['scheme']=='http' && $port != 80))) {
            $host .= ':'.$port;
        }
```

```
    $canonical_request = self::getCanonicalFFECPContent($q,
        $host, $urlParams['path']);
    $stringToSign = "AWS4-HMAC-SHA256\n{$date}\n{$scope}\n" .
      hash('sha256', $canonical_request);
    $signingKey = self::getSigningKey($credentAttrs[1], $credentAttrs[2],
        $credentAttrs[3], $awsKeyPairs[$pKey]);
    $mySign = hash_hmac('sha256', $stringToSign, $signingKey);
    if (strcmp($mySign,$sign)){
        return self::AWS4_ERROR_INVALID_SIGNATURE;
    }
    return self::AWS4_ERROR_NONE;
}
/**
 * Method to verify that the query parameters contain the elements
 * required in the response to the controller and the ones required to
 * sign the request.
 * @param array $qParams: an associative array in which the key of an
 * entry is the name of a query parameter and the corresponding value
 * is the value of that parameter.
 * @return an AWS_ERROR code.
 */
 private static function validateQueryParms($qParams) {
     if (is_null($qParams)) {
         return self::AWS4_ERROR_MISSING_QUERY;
     }
     if ((!isset($qParams['wlan'])) or (!isset($qParams['token']))
        or (!isset($qParams['dest']))) {
        return self::AWS4_ERROR_MISSING_QUERY_PARAMS;
     }
     if (!isset($qParams['X-Amz-Signature'])) {
         return self::AWS4_ERROR_MISSING_QUERY_SIGNATURE;
     }
     if(!isset($qParams['X-Amz-Algorithm'])) {
         return self::AWS4_ERROR_MISSING_QUERY_ALGORITHM;
     }
     if (!isset($qParams['X-Amz-Credential'])) {
         return self::AWS4_ERROR_MISSING_QUERY_CREDENTIAL;
     }
     if (!isset($qParams['X-Amz-Date'])) {
         return self::AWS4_ERROR_MISSING_QUERY_DATE;
     }
     if (!isset($qParams['X-Amz-Expires'])) {
         return self::AWS4_ERROR_MISSING_QUERY_EXPIRES;
     }
     if (!isset($qParams['X-Amz-SignedHeaders'])) {
         return self::AWS4_ERROR_MISSING_QUERY_SIGNED_HEADERS;
     }
     // The date & expires parameters exist in the request.
     // Verify that the request is not stale or replayed.
     $redirectedAt = DateTime::createFromFormat('Ymd?Gis?',
        $qParams['X-Amz-Date'], new DateTimeZone("UTC"));
     $expires = $qParams['X-Amz-Expires'];
     $now = date_create();
     $delta = $now->getTimestamp() - $redirectedAt->getTimestamp();
     // The following gives some lattitude for clocks not being synched
     if (($delta < -10) or ($delta > $expires)) {
         print("<br>");
         print(date("Y-m-d H:i:sZ", $now->getTimestamp()));
```

```
                print("<br>");
                   print("Redirected at: ");
                print(date("Y-m-d H:i:sZ", $redirectedAt->getTimestamp()));
                print("<br>");
                   print($now->getTimeZone()->getName());
                print("<br>");
            print($redirectedAt->getTimeZone()->getName());
                   print("<br>");
                print($expires);
                print("<br>");
                print($delta);
               return self::AWS4_ERROR_STALE_REQUEST;
             }
             return self::AWS4_ERROR_NONE;
         }
        /**
         * Method to generate the AWS signed URL address
         * @param string $pUrl: the URL that need to be appened with AWS
    parameters
         * @param string $identity: the AWS identity
         * @param string $sharedSecret: the secret shared with the controller
         * @param string $region:  the region component of the scope
         * @param string $service: the service component of the scope
         * @param int $expires: number of seconds till presigned URL is untrusted.
         * @return URL string with AWS parameters
         */
        public static function createPresignedUrl(
            $pUrl, $identity, $sharedSecret, $region,
            $service, $expires) {
            $urlParams = parse_url($pUrl);
            $httpDate = gmdate('Ymd\THis\Z', time());
            $scopeDate = substr($httpDate, 0, 8);
            $scope = "{$scopeDate}/".$region."/".
    $service."/".self::AWS4_EXTREME_REQUEST;
            $credential = $identity . '/' . $scope;
            $duration = $expires;
            //set the aws parameters
            $awsParams = array(
                'X-Amz-Date'=>$httpDate,
                'X-Amz-Algorithm'=> 'AWS4-HMAC-SHA256',
                'X-Amz-Credential'=> $credential,
                'X-Amz-SignedHeaders' =>'host',
                'X-Amz-Expires'=> $duration
            );
            parse_str($urlParams['query'],$q);
            $q = array_merge($q, $awsParams);
            ksort($q);
          $port = $urlParams['port'];
           $host = strtolower($urlParams['host']);
            if($port && (($urlParams['scheme']=='https' && $port !=
             443)||($urlParams['scheme']=='http' && $port != 80))) {
                $host .= ':'.$port;
            }
            $canonical_request = self::getCanonicalFFECPContent($q,
             $host, $urlParams['path'], true);
            $stringToSign = "AWS4-HMAC-SHA256\n{$httpDate}\n{$scope}\n" .
                hash('sha256', $canonical_request);
            $signingKey = self::getSigningKey(
```

```
                $scopeDate,
                $region,
                $service,
                $sharedSecret
        );
        $q['X-Amz-Signature'] = hash_hmac('sha256', $stringToSign,
                $signingKey);
        $p = substr($pUrl, 0, strpos($pUrl,'?'));
        $queryParams = array();
        foreach($q AS $k=>$v) {
            $queryParams[] = "$k=".rawurlencode($v);
        }
        $p .= '?'.implode('&', $queryParams);
        return $p;
    }
    /**
     * Method to generate the AWS signing key
     * @param string $shortDate: short date format (20140611)
     * @param string $region: Region name (us-east-1)
     * @param string $service: Service name (s3)
     * @param string $secretKey Secret Access Key
     * @return string
     */
    protected static function getSigningKey($shortDate, $region, $service,
$secretKey) {
        $dateKey = hash_hmac('sha256', $shortDate, 'AWS4' . $secretKey, true);
        $regionKey = hash_hmac('sha256', $region, $dateKey, true);
        $serviceKey = hash_hmac('sha256', $service, $regionKey, true);
        return  hash_hmac('sha256', self::AWS4_EXTREME_REQUEST, $serviceKey,
true);
    }
    /**
     * Create the canonical context for the AWS service
     * @param array $queryHash the query parameter hash
     * @param string $host host name or ip address for the target service
     * @param string $path the service address for the target service
     * @param boolean $encode determine if the query parameter need to be
encoded or not.
     * @return string the canonical content for the request
     */
    protected static function getCanonicalFFECPContent($queryHash, $host,
$path, $encode=false) {
        $queryParams = array();
        foreach($queryHash AS $k=>$v) {
            if($encode) {$v = rawurlencode($v);}
        $queryParams[] = "$k=$v";
         }
         $canonical_request = "GET\n"
            .$path."\n"
            .implode('&',$queryParams)."\n"
            .'host:'.$host
            ."\n\nhost\nUNSIGNED-PAYLOAD";
         return $canonical_request;
    }
    /**
     * Create user readable error message
     * @param integer $eid error code after verifying the AWS URL
     * @return string the error message
```

```
         */
        public static function getAwsError($eid) {
            $forAws = " for Amazon Web Service request.";
            SWITCH ($eid) {
                case self::AWS4_ERROR_NULL_INPUT:
                    $res = "Empty input".$forAws;
                break;
                case self::AWS4_ERROR_INPUT_BUFFER_TOO_SMALL:
                    $res = "Input buffer is too small".$forAws;
                break;
                case self::AWS4_ERROR_INVALID_PROTOCOL:
                    $res = "Invalid protocol".$forAws;
                break;
                case self::AWS4_ERROR_INPUT_URL_TOO_BIG:
                    $res = "Input url is too big".$forAws;
                break;
                case self::AWS4_ERROR_INPUT_ID_TOO_BIG:
                    $res = "Input ID is too big".$forAws;
                break;
                case self::AWS4_ERROR_INVALID_REGION:
                    $res = "Invalid region".$forAws;
                break;
                case self::AWS4_ERROR_INVALID_SIGNATURE:
                    $res = "Invalid signature".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY:
                    $res = "Missing all query parameters".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_DATE:
                    $res = "Missing query date".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_SIGNED_HEADERS:
                    $res = "Missing query signed headers".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_EXPIRES:
                    $res = "Missing query expires".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_SIGNATURE:
                    $res = "Missing query signature".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_CREDENTIAL:
                    $res = "Missing query credential".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_ALGORITHM:
                    $res = "Missing query algorithm".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_QUERY_PARAMS:
                    $res = "Missing query parameter".$forAws;
                break;
                case self::AWS4_ERROR_MISSING_CRED_PARAMS:
                    $res = "Missing credential parameters".$forAws;
                break;
                case self::AWS4_ERROR_STALE_REQUEST:
                    $res = "Invalid request date".$forAws;
                break;
                case self::AWS4_ERROR_UNKNOWN_IDENTITY:
                    $res = "Unrecognized identity or identity without a shared
secret.";
```

```
            break;
            default:
                $res = "Successfully validated".$forAws;
            break;
        }
        return $res;
    }
    /**
     * Return the AWS validation error message
     * @param string $pUrl
     * @return string the error message
     */
    public function getUrlValidationResult($pUrl) {
        $eid = self::verifyAwsUrlSignature($pUrl);
        return self::getAwsError($eid);
     }
  }
?>
```

ffecp-config.php

```
<?php
  // This file contains PHP associative arrays holding the relatively
  // static configuration for this ECP application. A real application
  // might read the data in from an XML or '.ini' file.
  // An associative array of identity => shared secret pairs.
  // This example only uses the first one. Any printable ASCII
  // alphanumeric string can be use for the identity and shared
  // secret so long as both the ECP and the controller use the
  // same pair.
  $awsKeyPairs = array(
      'BigAuthInc'=>'secretferqrer123456667',
      'testingidentity1'=>'secretferqrer123456668',
      'testingidentity2'=>'secretferqrer123456669'
  );
  // Aws Signature-related Configuration
  // Region and service are used to build the scope.
  // Expires is the maximum amount of time the signed URL
  // should be trusted.
  $awsConfig = array(
      'region' => 'world',
      'signature'=> 'v4',
      'service'=>'ecp',
      'expires'=>60
  );
?>
```

N

F DeviceLocations.xsd

This appendix contains the XML schema that is the basis for the published files of locations.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="DeviceLocations">
    <xs:complexType>
    <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded"
    name="DeviceLocation">
        <xs:complexType>
            <xs:all>
            <xs:element minOccurs="0" maxOccurs="1"
            name="LocationDetails">
        <xs:complexType>
            <xs:all>
                <xs:element name="Dimension">
                    <xs:complexType>
                       <xs:attribute default="METER" name="unit"
                       type="xs:string" />
                       <xs:attribute name="width" type="xs:integer" />
                       <xs:attribute name="length" type="xs:integer" />
                    </xs:complexType>
                </xs:element>
                <xs:element minOccurs="0" maxOccurs="1" name="FloorPlan">
                    <xs:complexType>
                       <xs:attribute name="imageFileName" type="xs:string" />
                    </xs:complexType>
                </xs:element>
            </xs:all>
                <xs:attribute name="floorRefId" type="xs:decimal" />
                <xs:attribute name="floorName" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:element name="Coordinates">
        <xs:complexType>
            <xs:attribute default="METER" name="unit" type="xs:string" />
            <xs:attribute name="y" type="xs:decimal" />
            <xs:attribute name="x" type="xs:decimal" />
            <xs:attribute name="apDistance" type="xs:decimal" />
        </xs:complexType>
    </xs:element>
    <xs:element name="Details">
        <xs:complexType>
            <xs:attribute name="lastLocatedTime" type="xs:string" />
        </xs:complexType>
    </xs:element>
    </xs:all>
            <xs:attribute name="probability" type="xs:decimal" />
            <xs:attribute name="currentlyTracked" type="xs:boolean" />
            <xs:attribute name="macAddress" type="xs:string" />
            <xs:attribute name="bssid" type="xs:string" />
            <xs:attribute name="ssid" type="xs:string" />
            <xs:attribute name="apMacAddress" type="xs:string" />
```

```
            <xs:attribute name="channelNumber" type="xs:integer" />
        /xs:complexType>
    /xs:element>
</xs:sequence>
            <xs:attribute default="1.0" name="schemaVersion"
            type="xs:string" />
            <xs:attribute name="fileGeneratedTime" type="xs:dateTime" />
            <xs:attribute name="entries" type="xs:decimal" />
        </xs:complexType>
    </xs:element>
</xs:schema>
```