Configuring Hotspots with Secure Hotspot
Table of Contents

Copyright Notice and Proprietary Information ................................................................. 3
Intended Audience ................................................................................................................. 4
Overview ................................................................................................................................. 5
  What Is Covered Here ........................................................................................................... 5
  Requirements for this Document ......................................................................................... 5
Introduction & Key Concepts ................................................................................................. 6
  What Is It? .............................................................................................................................. 6
  When to Use Secure Hotspot .............................................................................................. 8
  Key Concepts and Terminology ......................................................................................... 8
Installation Overview ............................................................................................................. 10
  Process Steps ....................................................................................................................... 10
ZoneDirector Setup ................................................................................................................ 11
  Overview .............................................................................................................................. 11
  Configuration ....................................................................................................................... 11
    Set Northbound Interface Password ................................................................................ 11
    Define Authentication Server .......................................................................................... 12
    Configure Hotspot Service ............................................................................................... 13
    Configure Open/Provisioning WLAN ............................................................................ 14
    Configure Secure WLAN ................................................................................................. 15
Captive Portal Setup ............................................................................................................... 17
  Overview .............................................................................................................................. 17
  Apache Installation ............................................................................................................. 17
    Install Apache .................................................................................................................. 17
    Verify Apache Installation ............................................................................................... 18
    Configure Apache .......................................................................................................... 19
Captive Portal Web Logic ...................................................................................................... 22
  Hotspot Flow ....................................................................................................................... 22
  Step 1: Client Association ................................................................................................. 25
  Step 2: DNS Query (HTTP) .............................................................................................. 25
  Step 3: Client Redirection ................................................................................................. 25
    Different Client Behaviors with Captive Portals ......................................................... 25
  Step 4: Captive Portal Logic .............................................................................................. 26
    Basic login form .............................................................................................................. 27
    Parameter parsing ............................................................................................................ 28
    Communicating with the ZoneDirector (all usage) ....................................................... 30
Copyright Notice and Proprietary Information

Copyright 2013 Ruckus Wireless, Inc. All rights reserved.

No part of this documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Ruckus Wireless, Inc. (“Ruckus”), or as expressly provided by under license from Ruckus.

Destination Control Statement

Technical data contained in this publication may be subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader’s responsibility to determine the applicable regulations and to comply with them.

Disclaimer

THIS DOCUMENTATION AND ALL INFORMATION CONTAINED HEREIN (“MATERIAL”) IS PROVIDED FOR GENERAL INFORMATION PURPOSES ONLY. RUCKUS AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THE MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, OR THAT THE MATERIAL IS ERROR-FREE, ACCURATE OR RELIABLE. RUCKUS RESERVES THE RIGHT TO MAKE CHANGES OR UPDATES TO THE MATERIAL AT ANY TIME.

Limitation of Liability

IN NO EVENT SHALL RUCKUS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES, OR DAMAGES FOR LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY YOU OR ANY THIRD PARTY, WHETHER IN AN ACTION IN CONTRACT OR TORT, ARISING FROM YOUR ACCESS TO, OR USE OF, THE MATERIAL.

Trademarks

Ruckus Wireless is a trademark of Ruckus Wireless, Inc. in the United States and other countries. All other product or company names may be trademarks of their respective owners.
Intended Audience

This document addresses configuration and testing for a Secure Hotspot using a Ruckus ZoneDirector™ Smart WLAN controller. The goal of this document is a successful implementation of this feature with Ruckus Wireless equipment.

This document provides step-by-step procedures for configuration and testing. Some knowledge of hotspots, Wi-Fi design and 802.11 principles is recommended. A detailed and technical discussion of HTML, JavaScript, python and XML is included here. Knowledge of these is highly recommended.

Sample code is referred to throughout this document. It may be downloaded from http://www.ruckuswireless.com/technology/SecureHotspot.
Overview
This document describes how to configure and test the Secure Hotspot functionality and features. The document is broken into the following main categories:

- Secure Hotspot introduction
- Installation overview
- FreeRADIUS setup
- Wi-Fi configuration
- Client device configuration
- Testing

What Is Covered Here
The usage cases in this document focus on configuring Secure Hotspot in a lab environment. This tech note describes the basic process as well as information that can be used to configure devices in a lab environment. Readers with their own RADIUS and web servers may skip some of these sections.

Requirements for this Document
In order to successfully follow the steps in this document, the following equipment (at a minimum) is required and assumed:

- Ruckus ZoneFlex™ access points and ZoneDirector Smart WLAN controller
- Wireless client device
- Web server for captive portal functions
Introduction & Key Concepts

What Is It?
Secure Hotspot leverages the power of an open hotspot network with Ruckus Wireless’ Dynamic Pre-Shared Key and Zero-IT technology. Dynamic pre-Shared Key (PSK) is a patented technology developed to provide robust and secure wireless access while eliminating the necessity for manual configuration of end devices and the secure management of encryption keys.

Instead of manually configuring each individual laptop with an encryption key and the requisite wireless SSID, Dynamic PSK automates and centralizes this process. Once enabled for the entire system, a new user simply connects to the Ethernet LAN and authenticates via a captive portal hosted on the Ruckus ZoneDirector. Mobile devices like the Apple® iPhone® can also be authenticated through a captive portal over wireless. This information is checked against any standard back-end authentication (AAA) server such as Active Directory, RADIUS, LDAP or an internal user database on the ZoneDirector.

Upon successful authentication, the ZoneDirector generates a unique encryption key for each user. The lifetime of the key can be configured to align with appropriate policies. A temporary applet with the unique user key and other wireless configuration information is then pushed to the client. This applet automatically configures the user’s device without any human intervention. The user then detaches from the LAN and connects to the wireless network. Once associated, the Dynamic PSK is bound to the specific user and the end device being used.

Secure Hotspot offers two modes of operation:
• Restricted (authentication) mode
• Unrestricted mode

Restricted mode is similar to how many hotspots operate today. The user must enter a username and password into a login form on a captive portal before gaining access as an authorized user.

Unrestricted mode takes a different approach. Instead of requiring a username and password that must already exist in an authentication database, the user may either enter some simple identification (such as an email address) or could simply be presented with the terms of use before authorization.

In each case, the user will receive a D-PSK that will provision their device and connect it to the secure network.

The following diagrams illustrates this process:

1. Client associates with provisioning hotspot SSID
2. User opens a browser
3. Browser is redirected to the captive portal
4. User enters login credentials on the captive portal login form
5. Captive portal sends credentials to the ZoneDirector
6. ZoneDirector validates credentials with authentication server
7. If authentication is successful, the user is issued a D-PSK
8. The client may download a Zero-IT script from the ZoneDirector or the captive portal to autoconfigure the secure network
9. Authorized client moves to the secure, encrypted network

Figure 2 - Secure Hotspot workflow for a restricted user
When to Use Secure Hotspot

Secure Hotspot is not intended for use in very large networks and care and planning should be used to ensure it functions correctly. In particular, the number of D-PSKs that can be configured on a ZoneDirector is bound by the size and model of the controller. The following table provides a guide:

<table>
<thead>
<tr>
<th>Model</th>
<th>Maximum Supported DPSK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZD1100</td>
<td>1,000</td>
</tr>
<tr>
<td>ZD3000</td>
<td>5,000</td>
</tr>
<tr>
<td>ZD5000</td>
<td>5,000</td>
</tr>
</tbody>
</table>

Key Concepts and Terminology

Some important concepts and terminology used in this document include:
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>A server that provides authentication, authorization and accounting services</td>
</tr>
<tr>
<td>D-PSK</td>
<td>Dynamic Pre-Shared Key</td>
</tr>
<tr>
<td>Open network</td>
<td>The initial provisioning SSID where a user authenticates and receives a D-PSK</td>
</tr>
<tr>
<td>Secure network</td>
<td>The encrypted network that a user’s D-PSK is activated to use</td>
</tr>
<tr>
<td>Zero-IT</td>
<td>A provisioning applet that can be downloaded and automatically configure the end device to connect to the secure network</td>
</tr>
</tbody>
</table>
Installation Overview

Process Steps
This document is organized around the following procedure:

1. Configure ZoneDirector controller
2. Configure captive portal web server
3. Create a test account
4. Test client connectivity
5. Troubleshooting

Information and resources can be found in the Appendices at the end of this document.

This document contains specific example files and code. These are offered as examples to illustrate concepts and should not be used outside of a lab environment.
ZoneDirector Setup

Overview
The ZoneDirector is the central Wi-Fi management point of the network. It manages the APs and the WLANs. It is also the point of communication with the client (initial browser redirection) and the captive portal for authentication.

This example uses the following example design:

- Open SSID = open-dpsk
- Secure SSID = secure-dpsk

The following steps are required to configure the ZoneDirector:

- Define authentication server* (restricted mode only)
- Configure hotspot service
- Configure open and secure WLANs

* The authentication server can be any of the types supported by the ZoneDirector. These are the internal database of the ZoneDirector, RADIUS, Active Directory and LDAP. For simplicity's sake, the internal database is used in this document, but any of the support server types will work.

Configuration
Before beginning these steps, make sure the ZoneDirector is configured with an IP address and is on a subnet that has connectivity to the AP and the captive portal server.

Set Northbound Interface Password
The ZoneDirector must be configured with a password. This allows a captive portal to authenticate itself to the ZoneDirector.

1. Log into the ZoneDirector Web UI
2. Go to Configuration
3. At the bottom of the screen, click the plus sign (+) next to Network Management
4. Select the checkbox next to Northbound Portal Interface
5. Enter a password
6. Click Apply

Define Authentication Server

Authentication of users requires a back-end system of some kind before defined on the ZoneDirector. The instructions below are only required if you are not using the ZoneDirector’s internal database for authentication.

1. Log into the ZoneDirector Web UI
2. Go to Configuration -> AAA Servers
3. Click the Create New link under the Authentication and Accounting Servers
4. Enter a name for the server and specify its type (RADIUS, AD, etc.)
5. Enter the IP address for the server
6. Fill out the rest of the information as needed – this will be different for each type
7. Click the OK button to save

**Configure Hotspot Service**

A hotspot service profile defines how an authenticated user is redirected to the captive portal as well as some other additional restrictions and information.

1. Go to Configuration->Hotspot Services
2. Click the Create New link
3. Give the profile a name
4. Enter the page on the captive portal to redirect new/authenticated clients
5. Select the authentication server from the drop-down box
Additional configuration is also available such as wireless client isolation, restricted subnets, etc. For more information on these, please refer to the Ruckus Wireless ZoneDirector User Guide.

**Configure Open/Provisioning WLAN**

The first SSID created is an open WLAN intended for initial provisioning of devices. It should have firewall policies that restrict the user from access any network services not required before login. It must however have connectivity to the captive portal and, potentially, the ZoneDirector.

1. Go to Configuration->WLANs
2. Click the Create New link
3. Enter the SSID name
4. Select Hotspot (WISP) as the WLAN type
5. Select Open for the Authentication and Encryption Options
6. Select the hotspot service created previously from the drop-down box
• Click the OK button to save

**Configure Secure WLAN**

The second SSID created is a secure, D-PSK-enabled WLAN intended for use by authorized devices. This is a PSK network with D-PSK and Zero-IT enabled.

1. Go to Configuration->WLANs
2. Click the Create New link
3. Enter the SSID name
4. Select Standard Usage as the WLAN type
5. Select Open for the Authentication Option
6. Select an Encryption Option – WPA2/AES is highly recommended
7. Enter a passphrase for the SSID – this should be considered the master PSK and not given out
8. Select the box labeled Enable Zero-IT Activation
9. Select the box labeled Dynamic PSK and specify the length of the passphrase
10. Click the OK button to save

Additional options are available for both the hotspot SSID and the open SSID. These include features such as client isolation, ACLs, VLAN assignment, etc. For more information, please refer to the Ruckus Wireless ZoneDirector User Guide.
Captive Portal Setup

Overview
When a user connects to the hotspot WLAN and opens a browser, they are automatically redirected to a captive portal. The captive portal hosts the logic required to ask for the user login and/or allow on-line setup and payment.

This section describes how to create the basic code required for a web server to handle captive portal requests. The procedure is broken down into the following steps:

- Install Apache web server
- Create basic code and install

Readers with their own web server may skip these steps. Unless using Apache, the process for configuring other web servers may be different from the process described below.

Apache Installation
Apache is an open source initiative that offers both pre-built binaries and full source code. It is a full-fledged web server and supports a wide variety of authentication methods.

Installation steps:

1. Install Apache
2. Verify installation
3. Configure Apache
4. Test

Install Apache
Apache installation is consists of two main steps: installing the software and editing configuration files. The Apache software consists of the following Linux packages:

- httpd
- httpd-tools
- apr-util
- apr
- python 2.7
- mod_wsgi
- mod_ssl*
- openssl*

* For HTTPS support

The examples in this document were developed with Apache 2.4.3 on a Fedora 17 server.
Install Apache: Step-by-Step

1. Launch the software manager. From the menu bar, select Applications-&gt;System Tools-&gt;Add/Remote Software
2. Enter “apache” in the search box to find Apache-related packages

3. Select the recommended packages listed previously
4. Click the Apply button

The next step configures Apache to run automatically as a server:

```
[root@webserver raddb]# systemctl enable httpd.service
ln -s '/lib/systemd/system/httpd.service'
'/etc/systemd/system/multi-user.target.wants/httpd.service'
[root@webserver ruckus]# systemctl start httpd.service
```

Verify Apache Installation

Test Apache was installed correctly by bringing up the test screen. To do this, open a browser on the web server and point to `http://localhost`. The Apache welcome page should appear.
Configure Apache

In addition to the software, Apache will also require edits of the following files:

- `/etc/httpd/conf/httpd.conf`

It’s a good idea to make backups of any files related to any existing services. If anything goes wrong with the configuration, you can always go back to the original file and start again. To backup these files open a Terminal window as root and enter the following commands:

```
[ruckus@webserver ruckus]$ sudo bash
[sudo] password for ruckus:
[root@hotspot ruckus]# cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.orig
```

It is also recommended to backup these files after the configuration is complete as well.

Depending on the deployment environment, the following options may need configuration in the `httpd.conf` file:

**Location for HTML files:**

```
DocumentRoot "/var/www/html"
```

**CGI Configuration**

CGI is how a webserver interacts with executable scripts and files. Because this example uses python scripts the server must be configured to use them. Find the line in `httpd.conf` that starts with the following:

---

2 These locations are based on a Fedora installation – different distributions may install in different locations. For simplicity, only the default Fedora location is used in this document
<Directory "/var/www/cgi-bin”>
And make sure the entire entry looks like this:

<Directory "/var/www/cgi-bin”>
  AllowOverride None
  Options +ExecCGI
  AddHandler cgi-script .cgi .py
  Order allow,deny
  Allow from all
  Require all granted
</Directory>

Next, add the python file extension (.py) to the AddHandler command. Find the line that looks like this:

#AddHandler cgi-script .cgi

and change it to:

AddHandler cgi-script .cgi .py

There is one last place where CGI executables need to be defined. Find this line:

Options Indexes FollowSymLinks

and change it to:

Options Indexes FollowSymLinks ExecCGI

Restart Apache after making these changes.

[root@webserver conf]# systemctl restart httpd.service

If the client downloads the Zero-IT script from the web server instead of directly from the ZoneDirector, the correct MIME types must be set. Edit the file /etc/mime.types and make sure the supported platforms are included:

application/x-apple-aspen-config  ipa mobileconfig
application/octet-stream  bin lha lzh exe class so
dll img iso
application/vnd.android.package-archive  apk

Note that the .exe extension is one of several mapped to the octet-stream type.
Once all of the changes have been made, restart the web server.
Captive Portal Web Logic

Before a web server can participate in Secure Hotspot, some code must be written to handle the logic of acquiring the user credentials and sending them to the ZoneDirector for verification. This code can be as elaborate or as simple as needed. However there is certain basic information and procedures that must happen for this to work correctly.

This example uses four main files:

- **login-restricted.html** – the login page where unauthenticated clients are redirected to authenticate
- **login-unrestricted.html** – the login page where unauthenticated clients are redirected for unrestricted configuration (no password required)
- **hotspot.js** – JavaScript functions for hotspot login
- **hotspot-restricted.py** – Python script that handles authentication with the ZoneDirector
- **hotspot-unrestricted.py** – Python script that handles creating an unrestricted user
- **hspotcommon.py** – Python functions library for shared functions
- **xmlcommon.py** – Python XML functions library

These files may be downloaded from http://www.ruckuswireless.com/technology/securehotspot. Some ancillary files such, as CSS style sheets used in these files are not included here. A complete tar file with all of the files is available for download.

Hotspot Flow

Before writing hotspot code for the captive port, it is useful to understand the workflow. The basic workflow for a restricted user is shown below:
The workflow to match this is as follows:

1. Client associates with provisioning hotspot SSID
2. User opens a browser
3. Browser is redirected to the captive portal
4. User enters login credentials on the captive portal login form
5. Captive portal sends credentials to the ZoneDirector
6. ZoneDirector validates credentials with authentication server
7. If authentication is successful, the user is issued a O-PSK
8. The client may download a Zero-IT script from the ZoneDirector or the captive portal to autoconfigure the secure network
9. Authorized client moves to the secure, encrypted network
Figure 4 - Detailed Secure Hotspot workflow for a restricted user

The workflow for an unrestricted user follows the same process except an authentication server is not used.
Step 1: Client Association
The first step is straightforward; the client comes to a hotspot area and connects to the SSID. At this point, no other action occurs unless the user opens a browser and attempts to connect to a web page. This network is open to facilitate easy connection but is restricted with very limited network access. Typically a client will be able to get DNS, DHCP and the captive portal and possibly the ZoneDirector if direct client download of the Zero-IT script is permitted.

Step 2: DNS Query (HTTP)
Web-based authentication relies on the gateway (AP) redirecting an initial HTTP query to the portal URL instead. It does this by listening for a DNS query from the browser. Redirection cannot occur until an HTTP DNS query occurs. In order to get redirected, the client must:

- Have a valid IP address
- Have a valid DNS server configured
- Perform an HTTP-based DNS query (either manually or automatically)
- HTTP request must be for an unencrypted web site
- Have access to the AP and the web server (firewall must allow HTTP/S access)

Step 3: Client Redirection
From the Wi-Fi client’s point of view, the following action happens: it requests a particular URL and receives an HTTP/1.0 302 Moved Temporarily message directing it to the captive portal URL. The URL the client receives is the redirect URL configured on the ZoneDirector hotspot profile. It also may also include additional information such as the location, etc. For more information on the parameters available for use, please refer to the Ruckus Wireless ZoneDirector User Guide.

Different Client Behaviors with Captive Portals
A client is typically redirected only after the user opens a browser and attempts to go to a non-encrypted web URL. Some clients however do no necessary require this step. In particular, most Apple products automatically attempt to connect to an Apple server when the network is brought up. In a packet capture, the client is seen attempting access to www.apple.com/library/test/success.html. This is done even if the browser is not open and is the reason why these clients ask for a login immediately unlike other clients that require the user open a browser manually.

HTTP/1.0 302 Moved Temporarily
Location:

Any web site that uses encryption (the URL starts with https://) will not be redirected. This is because the AP cannot decrypt the traffic to see the session information.
If the hotspot operator wants to make iOS devices behave the same way as other types of computers, www.apple.com can be added to the walled garden settings. Allowing clients access to their default URL will prevent triggering the previous behavior.

In this example, the following parameters are passed to the captive portal web server as part of this redirect:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sip</td>
<td>IP address of the ZoneDirector</td>
</tr>
<tr>
<td>mac</td>
<td>AP MAC</td>
</tr>
<tr>
<td>client_mac</td>
<td>Client MAC</td>
</tr>
<tr>
<td>uip</td>
<td>Client IP address</td>
</tr>
<tr>
<td>lid</td>
<td>Location ID information (configured on ZoneDirector)</td>
</tr>
<tr>
<td>url</td>
<td>The URL originally requested by the client</td>
</tr>
<tr>
<td>ssid</td>
<td>Open hotspot SSID name</td>
</tr>
<tr>
<td>loc</td>
<td>Location name (configured on ZoneDirector)</td>
</tr>
<tr>
<td>vlan</td>
<td>VLAN ID of the client</td>
</tr>
</tbody>
</table>

**Step 4: Captive Portal Logic**

Once the client has been redirected to the web server, the login page is loaded onto the client. The captive portal is responsible for acquiring the login credentials that will be sent to the ZoneDirector for verification against an authentication server.

An example redirect URL looks like this:

The following is snippet from an example login page for the web server. It loads JavaScript to handle the login logic and a form that asks for the user to enter their credentials. A full copy of this file is available in Appendix A: Web Logic.

Note that although the ZoneDirector originally sent the name of the open provisioning SSID in the redirect URL, from here on references to the WLAN are for the secure SSID.

**Basic login form**

The login form prompts the user to enter credentials in the form and then calls the JavaScript hotspot_login_form() to parse the redirect URL and post the information.

Example is taken from hotspot-restricted.html:

```html
<script type="text/javascript">hotspot_login_form();</script>

Username:<input type="text" name="username" size="20" maxlength="128">
Password:<input type="password" name="password" size="20" maxlength="128">
<input type="submit" name="button" value="I Agree">
```
Parameter parsing

The HTTP redirect that the ZoneDirector gives the client contains a query string with a number of parameters. Some parsing is required to extract these values for use.

Example taken from hotspot.js:
function hotspot_login_form() {
    document.write('<form name="hotspot_login_form"
        method="post"');
    document.write('action="../cgi-bin/hotspot-restricted.py"');
    document.write('onsubmit="return hotspot_login_form_check(this.form"');

    // Hidden form fields
    document.write('<input type="hidden" name="ap_mac" value="' +
                    get_param('mac') + '">
    document.write('<input type="hidden" name="zip" value="' +
                    get_param('sip') + '">
    document.write('<input type="hidden" name="client_mac" value="' +
                    get_param('client_mac') + '">
    document.write('<input type="hidden" name="uiip" value="' +
                    get_param('uiip') + '">
    document.write('<input type="hidden" name="requested_url" value="' +
                    get_param('url') + '">
    document.write('<input type="hidden" name="log_url" value="' +
                    window.location.href + '">
    document.write('<input type="hidden" name="ssid" value="' +
                    get_param('ssid') + '">
    document.write('<input type="hidden" name="vlan" value="' +
                    get_param('vlan') + '");

}

function get_param(name) {
    if (location.href.indexOf("?") >= 0) {
        var query=location.href.split("?"[1];
        var params=query.split("&");
        for (var i=0; i < params.length; i++) {
            value_pair=params[i].split("=");
            if (value_pair[0] == name)
                return unescape(value_pair[1]);
        }
    }
    return "";
}

There are two key actions performed here: the user is prompted to enter their credentials
(username and password) and the redirect URL is parsed into its component variables by
get_param(). This information is required for the rest of the process. Note that all of the
parameters extracted from the redirect URL are sent as hidden fields. This information is
then posted to the python script: hotspot-restricted.py.

document.write('action="../cgi-bin/hotspot-restricted.py"');
If unrestricted mode is used, there is no password for the user – they are simply prompted to enter a name, email address, etc. In this case, the code is the same except it doesn’t ask or check for a password in the form. See Appendix A: Web Logic for an example.

**Communicating with the ZoneDirector (all usage)**

So far, all the web server needs to do is acquire the user name and password for the client and parse some information supplied by the ZoneDirector in the redirect URL. The next step is sending this information to the ZoneDirector using the northbound XML interface.

Note that in order to communicate with the ZoneDirector, the captive portal must authenticate itself with the northbound interface password.

Example taken from hotspot-restricted.py:

All of the fields passed to this script by the JavaScript function are placed into local variables for use.

```python
# ZoneDirector IP address
tmp_sip = form["zip"].value
if len(tmp_sip) == 0:
    return
sip = tmp_sip.strip()
...
```

The server must also be configured with the URLs required to interface with the ZoneDirector. There are two. The northbound URL is used for all authentication messages:

```
https://zonedirector-address/admin/_portalintf.jsp
```

There is also a specific URL used for Zero-IT provisioning:

```
https://zonedirector-address/ user/user_extern_prov.jsp
```

```
# nbi_url is the URL for the northbound interface of the ZD. If using a different HTTPS port, this must be changed below
nbi_url = ''
nbi_url += "https://" + zd_ip + ":443/admin/_portalintf.jsp"

# prov_url is the URL to provision a Zero-IT script
prov_url = "https://" + zd_ip + "/user/user_extern_prov.jsp"
```

**Authenticate client with username and password (restricted use)**

To authenticate a restricted user (requires user name and password), the captive portal must send an XML message containing the northbound password and URL, client IP address, client MAC, user name and password.
XML message:

Request Method: POST
Request URL: /admin/_portalintf.jsp
Post Data:

<rupkus>
    <req-password>myPassword</req-password>
    <version>1.0</version>
    <command cmd="user-authenticate">
        ipaddr="172.16.17.101" macaddr="00:22:FB:18:8B:26"
        name="test" password="test"/>
</rupkus>

Example taken from xmlcommon.py:

```python
def createXmlAuthReq(nbi_password, client_ip, client_mac, username, password):
    xml_list = ['<rupkus><req-password>']
    xml_list.append(nbi_password)
    xml_list.append('</req-password>]
    xml_list.append('<version>1.0')
    xml_list.append('</version>]
    xml_list.append('<command cmd= "user-authenticate")
    xml_list.append(' ipaddr="')
    xml_list.append(client_ip)
    xml_list.append('"
    xml_list.append(' macaddr="')
    xml_list.append(client_mac)
    xml_list.append('"
    xml_list.append(' name="')
    xml_list.append(username),
    xml_list.append('"
    xml_list.append(' password="')
    xml_list.append(password)
    xml_list.append('"
    xml_list.append(' password="')
    xml_list.append(password)
    xml_list.append('"/> </rupkus>"
    xml_request = ".join(xml_list)
    xmlTrimedString = xml_request

    return xml_request
```

The resulting XML string is sent to the ZoneDirector.

```python
def sendXmlString(xml, nbi_url):
    buf = cStringIO.StringIO()
    c=pycurl.Curl()
    c.setopt(c.FAILONERROR, True)
    c.setopt(c.HEADER, ['Accept: text/xml', 'Accept-Charset: UTF-8'])
```
c.setopt(pycurl.SSL_VERIFYHOST, False)
c.setopt(c.URL, nbi_url)
c.setopt(c.POSTFIELDS, xml)
c.setopt(c.WRITEFUNCTION, buf.write)
try:
c.setopt(c.URL, nbi_url)
c.setopt(c.POSTFIELDS, xml)
c.setopt(c.VERBOSE,True)
c.perform()
except:
cgi.print_exception()
response = buf.getvalue()
buf.close
return response

The ZoneDirector will return a response code indicating if the authentication was successful or if there was an error. If the authentication was successful, the script can request the ZoneDirector generate a D-PSK for the user.

**Authenticate client without account (unrestricted)**

Only a user name of some kind is required for unrestricted users. There is no authentication against a server; the name is just something to associate with the D-PSK for the client. If desired, the user doesn’t need to enter anything, the captive portal could auto-generate a name based on some unique information such as the client MAC. In this case, all the user would see is the terms and conditions page.

**XML message:**

Request Method:POST
Request URL:/admin/_portalintf.jsp
Post Data:

```xml
<jn
<ruckus>
<req-password>myPassword</req-password>
<version>1.0</version>
<command cmd="unrestricted" ipaddr="172.18.110.221" macaddr="c4:17:fe:03:0d:1b" name="frank"/>
</ruckus>
```

The information sent to the ZoneDirector is exactly the same as the restricted user example except there is no password.

Example taken from xmlcommon.py:

```python
def createXmlUnrestrictedUserReq(nbi_password, client_ip, client_mac, username):
```
xml_list = ['<ruckus><req-password>']
xml_list.append(nbi_password)
xml_list.append('</req-password>

xml_list.append('<version>1.0</version>

xml_list.append('<command cmd= "unrestricted" ipaddr="')
xml_list.append(client_ip)
xml_list.append('" macaddr="')
xml_list.append(client_mac)
xml_list.append('" name="')
xml_list.append(username)
xml_list.append('"/>

xml_list.append('</ruckus>')

xml_request = ''.join(xml_list)

return xml_request

If this request succeeds, the client is automatically granted access.

**Request a D-PSK from the ZoneDirector**

An authenticated client (or unrestricted device) needs a D-PSK in order to connect to the secure, encrypted WLAN. Note that in the following example code, the same function is used to create a D-PSK for both restricted and unrestricted users. A variable called "restricted" is passed to indicate the type of user.

**XML message:**

Request Method: POST
Request URL: /admin/_portalintf.jsp
Post Data:

```xml
geruss</ruckus>
    <req-password>myPassword</req-password>
    <version>1.0</version>
    <command cmd="generate-dpsk">
        <dpsk expiration="HOURS" key-length="LEN" user="USER"
            mac="MAC_ADDR" wlan="SSID" vlan-id="VLAN_ID"/>
    </command>
</ruckus>
```

Expiration is in hours. The allowed values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Time until expiration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unlimited (no expiration)</td>
</tr>
<tr>
<td>24</td>
<td>One day</td>
</tr>
<tr>
<td>Time</td>
<td>Duration</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>168</td>
<td>One week</td>
</tr>
<tr>
<td>336</td>
<td>Two weeks</td>
</tr>
<tr>
<td>720</td>
<td>One month</td>
</tr>
<tr>
<td>1440</td>
<td>Two months</td>
</tr>
<tr>
<td>2160</td>
<td>Three months</td>
</tr>
<tr>
<td>4380</td>
<td>Six months</td>
</tr>
<tr>
<td>8760</td>
<td>One year</td>
</tr>
<tr>
<td>17520</td>
<td>Two years</td>
</tr>
</tbody>
</table>

The key length is the number of characters in the generated DPSK. The length must be at least 8 characters and no more than 62.

The VLAN ID must be between 1 and 4094.

Example from xmlcommon.py:

```python
def createXmlDpskReq(nbi_password, client_ip, client_mac, username, password, expiration, key_length, wlan, vlanId, restricted):
    xml_list = ['<ruckus><req-password>']
    xml_list.append(nbi_password)
    xml_list.append('</req-password>')</n_list.append('<version>1.0')
    xml_list.append('</version>')</n_list.append('<command cmd= "generate-dpsk">')
    xml_list.append('<dpsk expiration="')
    xml_list.append(expiration)
    xml_list.append('" key-length="')
    xml_list.append(key_length)
    xml_list.append('" user="')
    xml_list.append(username)
    xml_list.append('"')
    xml_list.append(' if restricted:
        xml_list.append(' password=""
        xml_list.append(password)
        xml_list.append(' if restricted:
        xml_list.append(' mac=""
        xml_list.append(client_mac)
        xml_list.append(' if restricted:
        xml_list.append(' vlan-id=""
        xml_list.append(vlanId)
```

© 2013 Ruckus Wireless, Inc. Secure Hotspot v1.9 34
Retrieve D-PSK from the ZoneDirector

Once the key is created, the captive portal can download the key information from the ZoneDirector.

XML message:

Request Method: POST
Request URL: /admin/_portalintf.jsp
Post Data:

```
<ruckus>
  <req-password>myPassword</req-password>
  <version>1.0</version>
  <command cmd="get-dpsk">
    <dpsk mac="MAC_ADDR" wlan="SSID"/>
  </command>
</ruckus>
```

Note that the information provided by this call is also returned in the response to a generate-dpsk request.

Example from xmlcommon.py:

```python
def createXmlFetchDpsk(nbi_password, client_mac, wlan):
  xml_list = ['<ruckus><req-password>']
  xml_list.append(nbi_password)
  xml_list.append('</req-password>')</ruckus>
xml_list.append('<version>1.0</version>')
xml_list.append('<command cmd="get-dpsk">')
xml_list.append('<dpsk mac="" wlan=""/>')
xml_list.append('</command>')
xml_list.append('</ruckus>')
xml_request = ''.join(xml_list)
```
return xml_request

Note that the expiration returned by the ZoneDirector here is not the number of hours but the actual date and time the key will expire.

At this point the user is authorized to use the secure SSID and can be given the D-PSK information (passphrase and expiration date) just retrieved.

![Figure 5 - Authorized client on ZoneDirector](image)

The user may also download the Zero-IT script to automatically configure their device and move them to the new secure WLAN. There are two ways to do this: allow the client to download the script directly from the ZoneDirector or have the captive portal download the file and then offer it to the client.
Download Zero-IT to the Client

If a client downloads the script directly from the ZoneDirector, it must have connectivity to the controller; otherwise this will fail. It’s important to understand the client will not be prompted to authenticate itself to the ZoneDirector. Instead, the captive portal must create a download URL that the client can use which the ZoneDirector will trust because it trusts the captive portal.

Access is granted until 2013/03/05 09:54:56

Click [link] to download the auto-configuration tool.
This URL takes the form of:

https://192.168.252.253/user/userExtern_prov.jsp?mac=f0:b4:79:18:20:7f&wlan=secure-dpsk&key=782e99e46173ac5a4b18328cdac7087d181aba4f

Note that the base URL is different from what has been used in other examples. It uses the following construction:

https://zonedirector-ip/user/userExtern_prov.jsp = base provisioning URL
mac = client MAC
wlan = secure SSID name
key = SHA1 hash

A new component of this URL is the key parameter. Since the client has never authenticated itself directly to the ZoneDirector, some method must be used to let the ZoneDirector know it can trust the client. The key is derived as follows:

client_string = client_mac&wlan&northboundPassword
key = SHA1(client_string)

The example code from hspotcommon.py is:

```python
def getProvLink(nbi_password, prov_url, client_mac, wlan):
    # Create the client string to hash
    client_string = ''
    client_string += client_mac + '&' + wlan + '&' + nbi_password

    # Create a SHA1 hash using the client information
    m = hashlib.sha1()
    m.update(client_string)
    key = m.hexdigest()
    key = urllib.unquote(key)

    req_url = ''
    req_url += prov_url + '?'
    req_url += 'mac=' + client_mac + '&' + wlan + '&' + key

    return req_url
```

This function returns the provisioning URL, which a client may use to directly download the Zero-IT script from the ZoneDirector. It takes the form of a simple HTML post. The link can be displayed to the user to click or could be started automatically.

Example from hspotcommon.py:
Click here to download the auto-configuration tool.

NOTE: If HTTPS is used, most clients will expect a valid SSL certificate from the ZoneDirector. ZoneDirectors use a self-signed certificate by default that is not trusted. To avoid problems, a commercial certificate issued by a known root or intermediary CA is highly recommended. A certificate issued by a private CA may be used, however clients must have this certificate installed and trusted beforehand. For more information on building a private Certificate Authority, please see the Ruckus technical notes: “Creating Private PKIs with XCA” and “Creating Private PKIs with Windows Server.”
Download Zero-IT to the Server

Most deployments will likely prefer to download the Zero-IT script to the captive portal and then offer it to the client. This has the advantage of keeping the ZoneDirector more secure by limiting access to it.

The web server downloads the script via the following XML call:

XML message:

Request Method: POST
Request URL: /admin/_portalintf.jsp
Post Data:

```xml
<ruckus>
  <req-password>myPassword</req-password>
  <version>1.0</version>
  <command cmd="get-prov-file" ipaddr="" macaddr="0f:a1:40:33:c0:00" username="test" platform="win61"
    version="" user-agent="Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)="/wlansvc name="open-dpsk" expiration="168" key-length="20" vlan-id="1" />
</command>
</ruckus>
```

This message is similar to others already used in this document. However it does require the client operating system (platform) and the browser user agent. Acceptable values for platform are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>All iOS devices</td>
<td>ipa, mobileconfig</td>
</tr>
<tr>
<td>android</td>
<td>All android devices</td>
<td>apk</td>
</tr>
<tr>
<td>macosx</td>
<td>Macintosh OS</td>
<td>app, mobileconfig</td>
</tr>
<tr>
<td>win62</td>
<td>Windows 8/Server 2012</td>
<td>exe</td>
</tr>
<tr>
<td>win61</td>
<td>Windows 7/Server 2008 R2</td>
<td>exe</td>
</tr>
<tr>
<td>win60</td>
<td>Windows Vista/Server 2008</td>
<td>exe</td>
</tr>
</tbody>
</table>
When the file is retrieved from the ZoneDirector, the web server must make sure the file is written with the correct file extension. Otherwise both the server and the client may not recognize the binary correctly.

The example code from xmlcommon.py is:

```python
# Client user agent
agent = os.environ["HTTP_USER_AGENT"]

# Length of the agent string cannot be greater than 128 characters
if len(agent) > 128:
    s = agent[0:128]
    agent = s

# Find the first matching OS in platformList[]
platform = ''
for x in platformList:
    if agent.find(x.agent) != -1:
        # Found the OS
        platform = x.name
        ext = x.ext

# Version - only needed for Android
version = ''
if platform == "android":
    version = '1.0'

# Some of these values may not be safe for use in HTTP and
# need to be encoded to remove unsafe characters
tmp_username = urllib.quote_plus(username)
username = tmp_username
```

<table>
<thead>
<tr>
<th>win52</th>
<th>Windows XP (64-bit)/Server 2003</th>
<th>exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>win51</td>
<td>Windows XP (32-bit)</td>
<td>exe</td>
</tr>
<tr>
<td>winm6s</td>
<td>Windows Mobile 6 Standard</td>
<td>exe</td>
</tr>
<tr>
<td>winm6p</td>
<td>Windows Mobile 6 Professional</td>
<td>exe</td>
</tr>
<tr>
<td>winm5s</td>
<td>Windows Mobile 5 Standard</td>
<td>exe</td>
</tr>
<tr>
<td>winm5p</td>
<td>Windows Mobile 5 Professional</td>
<td>exe</td>
</tr>
</tbody>
</table>
tmp_wlan = urllib.quote_plus(wlan)
wlan = tmp_wlan

tmp_expiration = urllib.quote_plus(expiration)
expiration = tmp_expiration

xml_list = ['<ruckus><req-password>',
xml_list.append(nbi_password)
xml_list.append('</req-password>'),
xml_list.append('<version>1.0'),
xml_list.append('</version>'),
xml_list.append('<command cmd="get-prov-file"'),
xml_list.append(' ipaddr="'),
xml_list.append(client_ip)
xml_list.append(' macaddr="'),
xml_list.append(client_mac)
xml_list.append(' username="'),
xml_list.append(username)
xml_list.append(' platform="'),
xml_list.append(platform)
xml_list.append(' user-agent="'),
xml_list.append(agent)
xml_list.append(' version="'),
xml_list.append(version)
xml_list.append('>
xml_list.append('<wlansvc name="'),
xml_list.append(wlan)
xml_list.append(' expiration="'),
xml_list.append(expiration)
xml_list.append(' key_length="'),
xml_list.append(key_length)
xml_list.append(' vlan-id="'),
xml_list.append(vlan)
xml_list.append(' />
xml_list.append('</command>'),
xml_list.append('</ruckus>'),
xml_list.append('>
xml_list.append(' ').join(xml_list)

return xml_request, ext
This function returns the XML string containing the download request and the file extension that should be used for the file.

# The response can be a binary stream or an XML file with a return code
# File download location is hard-coded here to /downloads
tmpdt = strftime('%Y%m%d-%H%M%S', gmtime())
file_loc = '/downloads/prov-%s.%s' % (tmpdt, ext)
file = 'prov-%s.%s' % (tmpdt, ext)

The web server is responsible for distributing the file to the client either upon request (through a link) or directly after the client is authorized.
Troubleshooting Secure Hotspot

Overview
This provides a list of common problems that might occur when Secure Hotspot is not functioning correctly. These can be roughly categorized as:

- Network connectivity
- DNS
- Redirection
- UAM Logic
- NAS authentication
- User Authentication

This is not an exhaustive list, but represents good places to start. Following these steps in order starts with the most basic problem and works upward from there in OSI layers as well as overall complexity.

Network Connectivity
Symptoms
Client is able to connect to the WLAN, but is not redirected to the captive portal login page.

What to Check
Problems with network connectivity typically occur because of one or more of the following:

- Client does not have an IP address
- Client cannot reach the captive portal

DNS
Symptoms
Client is able to connect to the WLAN, has the correct network connectivity, but is not redirected to the captive portal login page.

What to Check
Before a client is sent to a captive portal page, the AP must see a DNS query from the client’s web browser. This can only happen if the client is configured with a valid, reachable DNS server.
Redirection

Symptoms
Client is able to connect to the WLAN, has the correct network connectivity, but is not redirected to the captive portal login page.

What to Check
SSL
If the client traffic is encrypted, the AP will be unable to understand the HTTP request and respond to it. The initial HTTP request from the client must always be unencrypted.

Server Response
The client must able to communicate with the web server. Problems usually occur because the redirection URL is misconfigured on the AP or the client cannot reach the server.

Client Authentication Failure

Symptoms
Client enters credentials on the captive portal but can't get further access, i.e. get redirected to the login page again.

What to Check
Failure here tends to be a problem with how the POST is handled to the ZoneDirector. If the URL is not constructed correctly the ZoneDirector not authorize the client. This could be because:

• ZoneDirector northbound interface URL is incorrect
• Northbound interface is incorrect
• User name and password are not correct or are not included in the URL

Another problem may be if an self-signed or untrusted certificate is installed on the ZoneDirector. If the client considers the certificate invalid, the POST to the ZoneDirector may fail. The recommended solution is to install a commercial certificate from a known, trusted certificate authority.

Authentication Server Communications

Symptoms
Client is able to connect to the WLAN, has an IP address, sees the login page but does not get further. A refresh of the web page should show the login page again (client is not authorized).
What to Check
Before an authentication request can be processed, the ZoneDirector must be able to authenticate itself to the authentication server. This password is referred to as a shared secret and takes the form of an ASCII string. Common problems are:

- RADIUS server is not configured to recognize the NAS client
- The shared secret on the RADIUS server does not match what was configured on the ZoneDirector
- The ZoneDirector does not have connectivity to the authentication server

If this type of error occurs, a message is typically logged on the authentication server and is the first place to determine if this is occurring.

User Authentication
Symptoms
Client enters credentials on the captive portal but can't get further access, i.e. get redirected to the login page again.

What to Check
This is usually a simple error in the credentials entered by the user. A log on the authentication server should reveal if the user credentials were accepted or denied.

XML Communications Failure/Invalid Messages
Symptoms
The captive portal does not get a success status from the ZoneDirector.

What to Check
There are many reasons why an XML call can fail. For more information on specific error codes and their meanings, please see Appendix B: XML APIs.

If you do not have a tool that reveals the actual XML messages between the web server and the ZoneDirector, more information (including the exact XML messages) is available using a network capture tool such as Wireshark.

Using Wireshark to Capture XML Messages
In its default configuration, Wireshark is not able to show XML messages. This is because the traffic between the web server and the ZoneDirector is encrypted over SSL. In order to view the messages, Wireshark must be configured to decrypt this traffic. Unless a hub or mirrored port is used, Wireshark should be run on the web server.

The following steps configure Wireshark to decrypt SSL traffic to the ZoneDirector:
1. Log into the ZoneDirector Web UI and go to Configure->Certificate
2. Click the Advanced Options link
3. Click the Backup Private Key button
4. Copy this file to the machine running Wireshark
5. Open Wireshark
6. Go to Edit->Preferences
7. Click on the plus (+) button next to Protocols
8. Click on SSL in the list
9. Click Edit next to the RSA keys list
10. Click New
11. Enter the IP address of the ZoneDirector, port number (default is 443) and protocol (HTTP)
12. Click Key File and select the file containing the ZoneDirector's backed up private key
13. Password should be blank unless a new certificate (not the default) has been installed
14. Click OK

Once the key is installed, Wireshark will be able to view all encrypted data between the web server and the ZoneDirector. Decryption takes effect immediately, including any currently captured traffic on the screen.

The packet sending XML data from the captive portal to the ZoneDirector should look something like this:

POST /admin/_portalintf.jsp HTTP/1.1 (application/x-www-form-urlencoded)

Scroll down to the "Line-based text data: application/x-www-form-urlencoded" and open it. The XML message will be displayed.

<rukus><req-password>testme123</req-password><version>1.0</version><command cmd= "unrestricted" ipaddr="10.3.7.13" macaddr="4c:b1:99:35:5f:d7" name="Me"/></rukus>

Responses from the ZoneDirector can be viewed in the same fashion.

Using Debug Scripts to Test a Specific XML API Command

A command line utility called curl may be used to send XML commands directly to the ZoneDirector. This allows the individual XML commands to be tested outside of the web server environment. It is useful to make sure commands are getting parsed correctly.

Example scripts and directions on how to use them are available on the Ruckus SecureHotspot evaluation web site.
Appendix A: Web Logic

The files used in this document use HTML, JavaScript and Python code to enable a web server as a Secure Hotspot captive portal. The following are requirements for editing and executing this code:

- Configure web server for Javascript, Python and CGI
- Extract .html, .js, CSS and image files into the web server htdocs directory or equivalent
- Files must be readable/executable by the web server daemon account
- Extract .py scripts into cgi-bin directory or equivalent
- Edit first line of Python scripts to point to the local pathname of the Python binary
- Scripts must be readable/executable by the web server daemon account
- Please review the README.txt file included with the sample code for further instructions

Client web browsers must support JavaScript.

The example files may be downloaded from
Appendix B: XML APIs

This section outlines the available XML functions and their response codes.

Authentication Request
The procedure for authenticating or deleting a user is shown in the diagram below:

This is an example authentication request sent from the captive portal to the ZoneDirector.

Request Method: POST
Request URL: /admin/_portalintf.jsp
Post Data:

```
<ruckus>
    <req-password>myPassword</req-password>
    <version>1.0</version>
    <command cmd="user-authenticate" ipaddr="172.16.17.101" macaddr="00:22:FB:18:8B:26"
```
name="test" password="test"/>
</ruckus>

The ipaddr and macaddr are the IP address and the MAC address of the end user. Possible responses from ZD for this authentication requests are:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK - success</td>
</tr>
<tr>
<td>201</td>
<td>Login succeeded</td>
</tr>
<tr>
<td>202</td>
<td>Authentication pending</td>
</tr>
<tr>
<td>101</td>
<td>Client not authorized or is already authenticated</td>
</tr>
<tr>
<td>300</td>
<td>Client not found - failed lookup on IP address and MAC</td>
</tr>
<tr>
<td>302</td>
<td>Bad request – XML is not in the correct format</td>
</tr>
<tr>
<td>303</td>
<td>Version not support – version mismatch</td>
</tr>
<tr>
<td>304</td>
<td>Command not supported</td>
</tr>
<tr>
<td>400</td>
<td>Internal server error (ZoneDirector error processing request)</td>
</tr>
<tr>
<td>401</td>
<td>RADIUS server error – RADIUS connection refused or timed out</td>
</tr>
</tbody>
</table>

**Deletion Request**

After a user session is authorized, an external web server can terminate the user session via an XML request to the ZoneDirector. The user will be disassociated from the network and re-association and re-authentication is required. The workflow is shown in the previous diagram.

Request Method: POST

Request URL: /admin/_portalintf.jsp

Post Data:

<ruckus>
The ipaddr and macaddr are the IP address and the MAC address of the end user. Possible responses from ZD for this authentication requests are:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK - success</td>
</tr>
<tr>
<td>300</td>
<td>Client not found (failed lookup on IP address and MAC)</td>
</tr>
<tr>
<td>302</td>
<td>Bad request – XML is not in the correct format</td>
</tr>
<tr>
<td>303</td>
<td>Version not support – version mismatch</td>
</tr>
<tr>
<td>400</td>
<td>Internal server error (ZoneDirector error processing request)</td>
</tr>
</tbody>
</table>

**Generate a D-PSK**

The procedure for generating a D-PSK is shown in the diagram below:
Figure 8 - Generate or Retrieve a D-PSK

Request Method: POST

Request URL: /admin/_portalintf.jsp

Post Data:

```xml
<ruckus>
    <req-password>myPassword</req-password>
    <version>1.0</version>
    <command cmd="generate-dpsk"><dpsk expiration="24" key-length="20" user="test" mac="11:22:33:44:33:22" vlan-id="2" wlan="frank-dpsk" />
</command>
</ruckus>
```

If the request succeeds, the ZoneDirector will reply with the D-PSK information.

```xml
<ruckus>
    <version>1.0</version>
    <response-code>0</response-code>
    <message>OK</message>
</result>
```

© 2013 Ruckus Wireless, Inc.  Secure Hotspot v1.9  52
Possible responses from ZD for this authentication requests are:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK - success</td>
</tr>
<tr>
<td>100</td>
<td>Failed – the maximum number of D-PSKs has been reached</td>
</tr>
<tr>
<td>302</td>
<td>Bad request – XML is not in the correct format</td>
</tr>
<tr>
<td>303</td>
<td>Version not support – version mismatch</td>
</tr>
<tr>
<td>400</td>
<td>Internal server error (ZoneDirector error processing request)</td>
</tr>
</tbody>
</table>

Retrieve a D-PSK
The procedure for retrieving a user’s D-PSK is shown in the previous diagram.

Request Method: POST
Request URL: /admin/_portalintf.jsp
Post Data:

```xml
<ruckus>
  <req-password>myPassword</req-password>
  <version>1.0</version>
</command>
</ruckus>
```

If successful, the ZoneDirector will respond with the status of the request.

```xml
<ruckus>
  <version>1.0</version>
  <response-code>0</response-code>
  <message>OK</message>
</result>
</ruckus>
```
Possible responses from ZD for this authentication requests are:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>Failed – entry not found</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>Bad request – XML is not in the correct format</td>
<td></td>
</tr>
<tr>
<td>303</td>
<td>Version not support – version mismatch</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>Internal server error (ZoneDirector error processing request)</td>
<td></td>
</tr>
</tbody>
</table>

**Download Zero-IT**

The client can download the Zero-IT script directly from the ZoneDirector or the captive portal may download the script and offer it to the client.
Check if the post URL above is correct

Request Method: POST
Request URL: /user/client_auth_prov.jsp
Post Data:


Grant Unrestricted Client Access
The procedure for granting a client access without requiring authentication is shown in the diagram below:
Request Method: POST
Request URL: /admin/_portalintf.jsp

Post Data:

```
<ruckus>
<req-password>myPassword</req-password>
<version>1.0</version>
<command cmd="unrestricted" ipaddr="172.18.110.221" macaddr="c4:17:fe:03:0d:1b" name="frank"/>
</ruckus>
```

Possible responses from ZD for this authentication requests are:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
</tr>
<tr>
<td>300</td>
<td>Not found – unable to lookup the client by IP or MAC address</td>
</tr>
<tr>
<td>302</td>
<td>Bad request – XML is not in the correct format</td>
</tr>
<tr>
<td>Status Code</td>
<td>Error Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>303</td>
<td>Version not support – version mismatch</td>
</tr>
<tr>
<td>400</td>
<td>Internal server error (ZoneDirector error processing request)</td>
</tr>
</tbody>
</table>
Appendix C: Apache Configuration

This is the httpd.conf file used in the examples for this document.

# This is the main Apache HTTP server configuration file. It contains the configuration directives that give the server its instructions. See <URL:http://httpd.apache.org/docs/2.4/> for detailed information. In particular, see <URL:http://httpd.apache.org/docs/2.4/mod/directives.html> for a discussion of each configuration directive.

# Do NOT simply read the instructions in here without understanding what they do. They're here only as hints or reminders. If you are unsure consult the online docs. You have been warned.

# Configuration and logfile names: If the filenames you specify for many of the server's control files begin with "/" (or "drive:" for Win32), the server will use that explicit path. If the filenames do *not* begin with "/", the value of ServerRoot is prepended -- so '/log/access_log' with ServerRoot set to '/www' will be interpreted by the server as '/www/log/access_log', where as '/log/access_log' will be interpreted as '/log/access_log'.

# ServerRoot: The top of the directory tree under which the server's configuration, error, and log files are kept.
# Do not add a slash at the end of the directory path. If you point ServerRoot at a non-local disk, be sure to specify a local disk on the Mutex directive, if file-based mutexes are used. If you wish to share the same ServerRoot for multiple httpd daemons, you will need to change at least PidFile.
ServerRoot "/etc/httpd"

# Listen: Allows you to bind Apache to specific IP addresses and/or 
# ports, instead of the default. See also the <VirtualHost> 
# directive.
#
# Change this to Listen on specific IP addresses as shown below to 
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80

# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built 
as a DSO you
# have to place corresponding `LoadModule' lines at this location 
so the
# directives contained in it are actually available _before_ they 
are used.
# Statically compiled modules (those listed by `httpd -l’) do not 
# need
# to be loaded here.
#
# Example:
# LoadModule foo_module modules/mod_foo.so
#
Include conf.modules.d/*.conf

# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.
#
# User/Group: The name (or #number) of the user/group to run httpd as.
# It is usually good practice to create a dedicated user and group for
# running httpd, as with most system services.
#
User apache
Group apache

# 'Main' server configuration
#
# The directives in this section set up the values used by the 
# 'main' 
# server, which responds to any requests that aren't handled by a 
# <VirtualHost> definition. These values also provide defaults for
# any <VirtualHost> containers you may define later in the file.
# All of these directives may appear inside <VirtualHost> containers,
# in which case these default settings will be overridden for the
# virtual host being defined.
#
#
# ServerAdmin: Your address, where problems with the server should be
e-mailed. This address appears on some server-generated pages, such
as error documents. e.g. admin@your-domain.com
# ServerAdmin root@localhost
#
# ServerName gives the name and port that the server uses to identify itself.
# This can often be determined automatically, but we recommend you specify
# it explicitly to prevent problems during startup.
# # If your host doesn't have a registered DNS name, enter its IP address here.
# #ServerName www.example.com:80
#
# Deny access to the entirety of your server's filesystem. You must
# explicitly permit access to web content directories in other
# <Directory> blocks below.
# # <Directory />
#    AllowOverride none
#    Require all denied
# </Directory>
#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
#
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this
directory, but
# symbolic links and aliases may be used to point to other locations.
# DocumentRoot "/var/www/html"

#
# Relax access to content within /var/www.
#
<Directory "/var/www">
   AllowOverride None
   # Allow open access:
   Require all granted
</Directory>

# Further relax access to the default document root:
<Directory "/var/www/html">
   # Possible values for the Options directive are "None", "All",
   # or any combination of:
   #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch
   ExecCGI MultiViews
   # Note that "MultiViews" must be named *explicitly* ---
   "Options All"
   # doesn't give it to you.
   #
   # The Options directive is both complicated and important.
   Please see
   # http://httpd.apache.org/docs/2.4/mod/core.html#options
   # for more information.
   #
   Options Indexes FollowSymLinks ExecCGI

   #
   # AllowOverride controls what directives may be placed in
   .htaccess files.
   # It can be "All", "None", or any combination of the keywords:
   # Options FileInfo AuthConfig Limit
   #
   AllowOverride None

   #
   # Controls who can get stuff from this server.
   #
   Require all granted
   Allow from all
   Satisfy any
</Directory>

#
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
#
# The following lines prevent .htaccess and .htpasswd files from being viewed by Web clients.
#
<Files ".ht*">
    Require all denied
</Files>
#
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be logged here. If you *do* define an error log file for a <VirtualHost>
# container, that host's errors will be logged there and not here.
# ErrorLog "logs/error_log"
#
# LogLevel: Control the number of messages logged to the error log. Possible values include: debug, info, notice, warn, error, crit, alert, emerg.
# LogLevel warn

<IfModule log_config_module>
#
# The following directives define some format nicknames for use with a CustomLog directive (see below).
#
    LogFormat "%h %l %u %t "%r" %>s %b "\%{Referer}i\" \"\%{User-Agent}i\" " combined
    LogFormat "%h %l %u %t "%r" %>s %b" common

<IfModule logio_module>
# You need to enable mod_logio.c to use %I and %O
    LogFormat "%h %l %u %t "%r" %>s %b "\%{Referer}i\" \"\%{User-Agent}i\" %I %O" combinedio
</IfModule>
#
# The location and format of the access logfile (Common Logfile Format).
# If you do not define any access logfiles within a
<VirtualHost>
# container, they will be logged here. Contrariwise, if you
*do*
# define per-<VirtualHost> access logfiles, transactions will
be
# logged therein and *not* in this file.
#
#CustomLog "logs/access_log" common
#
# If you prefer a logfile with access, agent, and referer
information
# (Combined Logfile Format) you can use the following
directive.
#
CustomLog "logs/access_log" combined
</IfModule>

<IfModule alias_module>
#
# Redirect: Allows you to tell clients about documents that
used to
# exist in your server's namespace, but do not anymore. The
client
# will make a new request for the document at its new location.
# Example:
# Redirect permanent /foo http://www.example.com/bar
#
# Alias: Maps web paths into filesystem paths and is used to
# access content that does not live under the DocumentRoot.
# Example:
# Alias /webpath /full/filesystem/path
#
# If you include a trailing / on /webpath then the server will
# require it to be present in the URL. You will also likely
# need to provide a <Directory> section to allow access to
# the filesystem path.
#
# ScriptAlias: This controls which directories contain server
scripts.
# ScriptAliases are essentially the same as Aliases, except
that
# documents in the target directory are treated as applications
and
# run by the server when requested rather than as documents
sent to the
# client. The same rules about trailing "/" apply to
ScriptAlias
# directives as to Alias.
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"

</IfModule>

<Directory "/var/www/cgi-bin">
  AllowOverride None
  Options +ExecCGI
  AddHandler cgi-script .cgi .py
  Order allow,deny
  Allow from all
  Require all granted
  Satisfy any
</Directory>

<IfModule mime_module>
  #
  # TypesConfig points to the file containing the list of
  # filename extension to MIME-type.
  #
  # TypesConfig /etc/mime.types
  #
  # AddType allows you to add to or override the MIME
  # configuration
  # file specified in TypesConfig for specific file types.
  #
  # AddType application/x-gzip .tgz
  #
  # AddEncoding allows you to have certain browsers uncompress
  # information on the fly. Note: Not all browsers support this.
  #
  # AddEncoding x-compress .Z
  # AddEncoding x-gzip .gz .tgz
  #
  # If the AddEncoding directives above are commented-out, then
  you
  # probably should define those extensions to indicate media
types:
  #
  AddType application/x-compress .Z
  AddType application/x-gzip .gz .tgz

  #
  # AddHandler allows you to map certain file extensions to
  "handlers":
# actions unrelated to filetype. These can be either built into the server
# or added with the Action directive (see below)
# To use CGI scripts outside of ScriptAliased directories:
# (You will also need to add “ExecCGI” to the “Options”
directive.)
# AddHandler cgi-script .cgi .py

# For type maps (negotiated resources):
#AddHandler type-map var

#
# Filters allow you to process content before it is sent to the client.
# To parse .shtml files for server-side includes (SSI):
# (You will also need to add “Includes” to the “Options”
directive.)
# AddType text/html .shtml
AddOutputFilter INCLUDES .shtml

</IfModule>

# Specify a default charset for all content served; this enables
# interpretation of all content as UTF-8 by default. To use the
# default browser choice (ISO-8859-1), or to allow the META tags
# in HTML content to override this choice, comment out this
# directive:
# AddDefaultCharset UTF-8

#
# The mod_mime_magic module allows the server to use various hints
from the
# contents of the file itself to determine its type. The
MIMEMagicFile
# directive tells the module where the hint definitions are
located.
# MIMEMagicFile conf/magic

#
# Customizable error responses come in three flavors:
# 1) plain text 2) local redirects 3) external redirects
# Some examples:
#ErrorDocument 500 "The server made a boo boo."
#ErrorDocument 404 /missing.html
#ErrorDocument 404 "/cgi-bin/missing_handler.pl"
#ErrorDocument 402 http://www.example.com/subscription_info.html
#

# EnableMMAP and EnableSendfile: On systems that support it, # memory-mapping or the sendfile syscall may be used to deliver # files. This usually improves server performance, but must # be turned off when serving from networked-mounted # filesystems or if support for these functions is otherwise # broken on your system. # Defaults if commented: EnableMMAP On, EnableSendfile Off #

#EnableMMAP off
EnableSendfile on

# Supplemental configuration
#
# Load config files in the "/etc/httpd/conf.d" directory, if any.
IncludeOptional conf.d/*.conf