Manual:Customizing Hotspot

From MikroTik Wiki

Applies to RouterOS: v3, v4, v5+

Contents

HTML customizations

Summary

You can create a completely different set of servlet pages for each HotSpot server you have, specifying the directory in "html-override-directory" property of a HotSpot server profile /ip hotspot profile. The default servlet pages are copied in the directory "hotspot" directory right after you create server profile. This directory can be accessed by connecting to the router with an FTP client. You can copy this directory and modify the pages as you like using the information from this section of the manual. Note that it is suggested to edit the files manually, as automated HTML editing tools may corrupt the pages by removing variables or other vital parts. After you are finished with content modification you need to upload this modified content to some custom directory on hotspot router and point previously mentioned property "html-override-directory" value as path to this new custom HTML directory.

Note: If "html-override-directory" value path is missing or empty then hotspot server will revert back to default HTML files.

Available Pages

Main HTML servlet pages, which are shown to user:

- **redirect.html** - redirects user to another url (for example, to login page)
- **login.html** - login page shown to a user to ask for username and password. This page may take the following parameters:
    - **username** - username
    - **password** - either plain-text password (in case of PAP authentication) or MD5 hash of chap-id variable, password and CHAP challenge (in case of CHAP authentication). This value is used as e-mail address for trial users
    - **dst** - original URL requested before the redirect. This will be opened on successfull login
    - **popup** - whether to pop-up a status window on successfull login
    - **radius<id>** - send the attribute identified with <id> in text string form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
    - **radius<id>u** - send the attribute identified with <id> in unsigned integer form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
    - **radius<id>-<vnd-id>** - send the attribute identified with <id> and vendor ID <vnd-id> in text string form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
    - **radius<id>-<vnd-id>u** - send the attribute identified with <id> and vendor ID <vnd-id> in unsigned integer form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
- **md5.js** - JavaScript for MD5 password hashing. Used together with http-chap login method

- **alogin.html** - page shown after client has logged in. It pops-up status page and redirects browser to originally requested page (before he/she was redirected to the HotSpot login page)
- **status.html** - status page, shows statistics for the client. It is also able to display advertisements automatically
- **logout.html** - logout page, shown after user is logged out. Shows final statistics about the finished session. This page may take the following additional parameters:
  - **erase-cookie** - whether to erase cookies from the HotSpot server on logout (makes impossible to log in with cookie next time from the same browser, might be useful in multiuser environments)
- **error.html** - error page, shown on fatal errors only

Some other pages are available as well, if more control is needed:

- **rlogin.html** - page, which redirects client from some other URL to the login page, if authorization of the client is required to access that URL
- **rstatus.html** - similarly to rlogin.html, only in case if the client is already logged in and the original URL is not known
- **radvert.html** - redirects client to the scheduled advertisement link
- **flogin.html** - shown instead of login.html, if some error has happened (invalid username or password, for example)
- **fstatus.html** - shown instead of redirect, if status page is requested, but client is not logged in
- **flogout.html** - shown instead of redirect, if logout page is requested, but client is not logged in

Serving Servlet Pages

The HotSpot servlet recognizes 5 different request types:

1. **request for a remote host**
   - if user is logged in and advertisement is due to be displayed, radvert.html is displayed. This page makes redirect to the scheduled advertisment page
   - if user is logged in and advertisement is not scheduled for this user, the requested page is served

- if user is not logged in, but the destination host is allowed by walled garden, then the request is also served
- if user is not logged in, and the destination host is disallowed by walled garden, rlogin.html is displayed; if rlogin.html is not found, redirect.html is used to redirect to the login page

2. **request for "/" on the HotSpot host**
    - if user is logged in, rstatus.html is displayed; if rstatus.html is not found, redirect.html is used to redirect to the status page
    - if user is not logged in, rlogin.html is displayed; if rlogin.html is not found, redirect.html is used to redirect to the login page
3. **request for "/login" page**
    - if user has successfully logged in (or is already logged in), alogin.html is displayed; if alogin.html is not found, redirect.html is used to redirect to the originally requested page or the status page (in case, original destination page was not given)
    - if user is not logged in (username was not supplied, no error message appeared), login.html is showed
    - if login procedure has failed (error message is supplied), flogin.html is displayed; if flogin.html is not found, login.html is used
    - in case of fatal errors, error.html is showed
4. **request for "/status" page**
    - if user is logged in, status.html is displayed
    - if user is not logged in, fstatus.html is displayed; if fstatus.html is not found, redirect.html is used to redirect to the login page
5. **request for '/logout' page**
    - if user is logged in, logout.html is displayed
    - if user is not logged in, flogout.html is displayed; if flogout.html is not found, redirect.html is used to redirect to the login page

Note: If it is not possible to meet a request using the pages stored on the router's FTP server, Error 404 is displayed

There are many possibilities to customize what the HotSpot authentication pages look like:

- The pages are easily modifiable. They are stored on the router's FTP server in the directory you choose for the respective HotSpot server profile.
- By changing the variables, which client sends to the HotSpot servlet, it is possible to reduce keyword count to one (username or password; for example, the client's MAC address may be used as the other value) or even to zero (License Agreement; some predefined values general for all users or client's MAC address may be used as username and password)
- Registration may occur on a different server (for example, on a server that is able to charge Credit Cards). Client's MAC address may be passed to it, so that this information need not be written in manually. After the registration, the server should change RADIUS database enabling client to log in for some amount of time.

To insert variable in some place in HTML file, the $(var_name) syntax is used, where the "var_name" is the name of the variable (without quotes). This construction may be used in any HotSpot HTML file accessed as '/', '/login', '/status' or '/logout', as well as any text or HTML (.txt, .htm or .html) file stored on the HotSpot server (with the exception of traffic counters, which are available in status page only, and **error**, **error-orig**, **chap-id**, **chap-challenge** and **popup** variables, which are available in login page only). For example, to show a link to the login page, following construction can be used:

```
<a href="$(link-login)">login</a>
```

Variables

All of the Servlet HTML pages use variables to show user specific values. Variable names appear only in the HTML source of the servlet pages - they are automatically replaced with the respective values by the HotSpot Servlet. For most variables there is an example of their possible value included in brackets. All the described variables are valid in all servlet pages, but some of them just might be empty at the time they are accesses (for example, there is no uptime before a user has logged in).

List of available variables

Common server variables:

- `hostname` - DNS name or IP address (if DNS name is not given) of the HotSpot Servlet ("hotspot.example.net")
- `identity` - RouterOS identity name ("MikroTik")
- `login-by` - authentication method used by user
- `plain-passwd` - a "yes/no" representation of whether HTTP-PAP login method is allowed ("no")
- `server-address` - HotSpot server address ("10.5.50.1:80")
- `ssl-login` - a "yes/no" representation of whether HTTPS method was used to access that servlet page ("no")
- `server-name` - HotSpot server name (set in the /ip hotspot menu, as the name property)

Links:

- `link-login` - link to login page including original URL requested ("http://10.5.50.1/login?dst=http://www.example.com/")
- `link-login-only` - link to login page, not including original URL requested ("http://10.5.50.1/login")
- `link-logout` - link to logout page ("http://10.5.50.1/logout")
- `link-status` - link to status page ("http://10.5.50.1/status")
- `link-orig` - original URL requested ("http://www.example.com/")

General client information:

- `domain` - domain name of the user ("example.com")
- `interface-name` - physical HotSpot interface name (in case of bridged interfaces, this will return the actual bridge port name)
- `ip` - IP address of the client ("10.5.50.2")
- `logged-in` - "yes" if the user is logged in, otherwise - "no" ("yes")
- `mac` - MAC address of the user ("01:23:45:67:89:AB")
- `trial` - a "yes/no" representation of whether the user has access to trial time. If users trial time has expired, the value is "no"
- `username` - the name of the user ("John")
- `host-ip` - client IP address from /ip hotspot host table

User status information:

- `idle-timeout` - idle timeout ("20m" or "" if none)
- `idle-timeout-secs` - idle timeout in seconds ("88" or "0" if there is such timeout)
- `limit-bytes-in` - byte limit for send ("1000000" or "---" if there is no limit)
- `limit-bytes-out` - byte limit for receive ("1000000" or "---" if there is no limit)
- `refresh-timeout` - status page refresh timeout ("1m30s" or "" if none)
- `refresh-timeout-secs` - status page refresh timeout in seconds ("90s" or "0" if none)
- `session-timeout` - session time left for the user ("5h" or "" if none)
- `session-timeout-secs` - session time left for the user, in seconds ("3475" or "0" if there is such timeout)
- `session-time-left` - session time left for the user ("5h" or "" if none)
- `session-time-left-secs` - session time left for the user, in seconds ("3475" or "0" if there is such timeout)
- `uptime` - current session uptime ("10h2m33s")
- `uptime-secs` - current session uptime in seconds ("125")

Traffic counters, which are available only in the status page:

- `bytes-in` - number of bytes received from the user ("15423")
- `bytes-in-nice` - user-friendly form of number of bytes received from the user ("15423")
- `bytes-out` - number of bytes sent to the user ("11352")
- `bytes-out-nice` - user-friendly form of number of bytes sent to the user ("11352")
- `packets-in` - number of packets received from the user ("251")
- `packets-out` - number of packets sent to the user ("211")
- `remain-bytes-in` - remaining bytes until limit-bytes-in will be reached ("337465" or "---" if there is no limit)
- `remain-bytes-out` - remaining bytes until limit-bytes-out will be reached ("124455" or "---" if there is no limit)

Miscellaneous variables:

- `session-id` - value of 'session-id' parameter in the last request
- `var` - value of 'var' parameter in the last request
- `error` - error message, if something failed ("invalid username or password")
- `error-orig` - original error message (without translations retrieved from errors.txt), if something failed ("invalid username or password")
- `chap-id` - value of chap ID ("\371")
- `chap-challenge` - value of chap challenge ("\357\015\330\013\021\234\145\245\303\253\142\246\133\175\37
- `popup` - whether to pop-up checkbox ("true" or "false")
- `advert-pending` - whether an advertisement is pending to be displayed ("yes" or "no")
- `http-status` - allows the setting of the http status code and message
- `http-header` - allows the setting of the http header

RADIUS-related variables:

- `radius<id>` - show the attribute identified with <id> in text string form (in case RADIUS authentication was used; "" otherwise)
- `radius<id>u` - show the attribute identified with <id> in unsigned integer form (in case RADIUS authentication was used; "0" otherwise)
- `radius<id>-<vnd-id>` - show the attribute identified with <id> and vendor ID <vnd-id> in text string form (in case RADIUS authentication was used; "" otherwise)
- `radius<id>-<vnd-id>u` - show the attribute identified with <id> and

vendor ID <vnd-id> in unsigned integer form (in case RADIUS authentication was used; "0" otherwise)

Working with variables

$(if <var_name>) statements can be used in theses pages. Following content will be included, if value of <var_name> will not be an empty string. It is an equivalent to $(if <var_name> != "") It is possible to compare on equivalence as well: $(if <var_name> == <value>) These statements have effect until $(elif <var_name>), $(else) or $(endif). In general case it looks like this:

```
some content, which will always be displayed
$(if username == john)
Hey, your username is john
$(elif username == dizzy)
Hello, Dizzy! How are you? Your administrator.
$(elif ip == 10.1.2.3)
You are sitting at that crappy computer, which is damn slow...
$(elif mac == 00:01:02:03:04:05)
This is an ethernet card, which was stolen few months ago...
$(else)
I don't know who you are, so lets live in peace.
$(endif)
other content, which will always be displayed
```

Only one of those expressions will be shown. Which one - depends on values of those variables for each client.

Redirects and custom Headers

Starting from RouterOS 5.12 there are 2 new hotspot html page variables:

- **http-status** - allows the setting of the http status code and message
- **http-header** - allows the setting of the http header message

Example:

```
        $(if http-status == 302)Hotspot login required$(endif)
        $(if http-header == "Location")$(link-redirect)$(endif)
```

Note: Although the above appears to use the conditional expression 'if' it is in fact setting the 'http-status' to '302' not testing for it. Also the same for the variable 'http-header'. Once again, even though it uses an 'if' it is in fact setting the variable to 'Location' followed by the url set from the variable 'link-redirect'.

E.g. in the case where $(link-redirect) evaluates to "http://192.168.88.1/login", then the HTTP response returned to the client will be changed to:

```
HTTP/1.0 302 Hotspot login required
<regular HTTP headers>
Location: http://192.168.88.1/login
```

http-status syntax:

```
        $(if http-status == XYZ)HTTP_STATUS_MESSAGE$(endif)
```

- *XYZ* - The status code you wish to return. Should be 3 decimal digits, first one must not be 0
- *HTTP_STATUS_MESSAGE* - any text you wish to return to the client which will follow the above status code in the HTTP reply

In any HTTP response it will be on the first line and will be as follows:

```
HTTP/1.0 XYZ HTTP_STATUS_MESSAGE
```

http-header syntax:

```
        $(if http-header == HTTP_HEADER_NAME)HTTP_HEADER_VALUE$(endif)
```

- *HTTP_HEADER_NAME* - name of the HTTP header to be sent in the response
- *HTTP_HEADER_VALUE* - value of the HTTP header with name HTTP_HEADER_NAME to be sent in the response

The HTTP response will appear as:

```
HTTP_HEADER_NAME: HTTP_HEADER_VALUE
```

All variables and conditional expressions within HTTP_HEADER_VALUE and HTTP_STATUS_MESSAGE are processed as usual.

In case multiple headers with the same name are added, then only the last one will be used (previous ones will be discarded). It allows the system to override regular HTTP headers (for example, Content-Type and Cache-Control).

Customizing Error Messages

All error messages are stored in the errors.txt file within the respective HotSpot servlet directory. You can change and translate all these messages to your native language. To do so, edit the errors.txt file. You can also use variables in the messages. All instructions are given in that file.

Multiple Versions of HotSpot Pages

Multiple HotSpot page sets for the same HotSpot server are supported. They can be chosen by user (to select language) or automatically by JavaScript (to select PDA/regular version of HTML pages).

To utilize this feature, create subdirectories in HotSpot HTML directory, and place those HTML files, which are different, in that subdirectory. For example, to translate everything in Latvian, subdirectory "lv" can be created with login.html, logout.html, status.html, alogin.html, radvert.html and errors.txt files, which are translated into Latvian. If the requested HTML page can not be found in the requested subdirectory, the corresponding HTML file from the main directory will be used. Then main login.html file would contain link to "/lv/login?dst=$(link-orig-esc)", which then displays Latvian version of login page: <a href="/lv/login?dst=$(link-orig-esc)">Latviski</a> . And Latvian version would contain link to English version: <a href="/login?dst=$(link-orig-esc)">English</a>

Another way of referencing directories is to specify 'target' variable:

```
<a href="$(link-login-only)?dst=$(link-orig-esc)&target=lv">Latviski</a>
<a href="$(link-login-only)?dst=$(link-orig-esc)&target=%2F">English</a>
```

After preferred directory has been selected (for example, "lv"), all links to local HotSpot pages will contain that path (for example, $(link-status) = "http://hotspot.mt.lv/lv/status"). So, if all HotSpot pages reference links using "$(link-xxx)" variables, then no more changes are to be made - each client will stay within the selected directory all the time.

Misc

If you want to use HTTP-CHAP authentication method it is supposed that you include the **doLogin()** function (which references to the **md5.js** which must be already loaded) before the **Submit** action of the login form. Otherwise, CHAP login will fail.

The resulting password to be sent to the HotSpot gateway in case of HTTP-CHAP method, is formed MD5-hashing the concatenation of the following: chap-id, the password of the user and chap-challenge (in the given order)

In case variables are to be used in link directly, then they must be escaped accordingly. For example, in login page, **<a href="https://login.example.com/login?mac=$(mac)&user=$(username)">link</a>** will not work as intended, if username will be "123&456=1 2". In this case instead of $(user), its escaped version must be used: $(user-esc): **<a href="https://login.server.serv/login?mac=$(mac-esc)&user=$(user-esc)">link</a>**. Now the same username will be converted to "123%26456%3D1+2", which is the valid representation of "123&456=1 2" in URL. This trick may be used with any variables, not only with $(username).

There is a boolean parameter "erase-cookie" to the logout page, which may be either "on" or "true" to delete user cookie on logout (so that the user would not be automatically logged on when he/she opens a browser next time.

Examples

With basic HTML language knowledge and the examples below it should be easy to implement the ideas described above.

- To provide predefined value as username, in login.html change:

```
<type="text" value="$(username)>
```

to this line:

```
<input type="hidden" name="username" value="hsuser">
```

(where hsuser is the username you are providing)

- To provide predefined value as password, in login.html change:

```
<input type="password">
```

to this line:

```
<input type="hidden" name="password" value="hspass">
```

(where hspass is the password you are providing)

- To send client's MAC address to a registration server in form of:

https://www.example.com/register.html?mac=XX:XX:XX:XX:XX:XX

change the Login button link in login.html to:

```
https://www.example.com/register.html?mac=$(mac)
```

(you should correct the link to point to your server)

- To show a banner after user login, in alogin.html after

$(if popup == 'true') add the following line:

```
open('http://www.example.com/your-banner-page.html', 'my-banner-name','');
```

(you should correct the link to point to the page you want to show)

- To choose different page shown after login, in login.html change:

```
<input type="hidden" name="dst" value="$(link-orig)">
```

to this line:

```
<input type="hidden" name="dst" value="http://www.example.com">
```

(you should correct the link to point to your server)

- To erase the cookie on logoff, in the page containing link to the logout (for example, in status.html) change:

```
open('$(link-logout)', 'hotspot_logout', ...
```

to this:

```
open('$(link-logout)?erase-cookie=on', 'hotspot_logout', ...
```

or alternatively add this line:

```
<input type="hidden" name="erase-cookie" value="on">
```

before this one:

```
<input type="submit" value="log off">
```

External authentication

Another example is making HotSpot to authenticate on a remote server (which may, for example, perform creditcard charging):

- Allow direct access to the external server in walled-garden (either HTTP-based, or IP-based)
- Modify login page of the HotSpot servlet to redirect to the external authentication server. The external server should modify RADIUS database as needed

Here is an example of such a login page to put on the HotSpot router (it is redirecting to https://auth.example.com/login.php, replace with the actual address of an external authentication server):

```
<html>
<title>...</title>
<body>
<form name="redirect" action="https://auth.example.com/login.php" method="post">
<input type="hidden" name="mac" value="$(mac)">
<input type="hidden" name="ip" value="$(ip)">
<input type="hidden" name="username" value="$(username)">
<input type="hidden" name="link-login" value="$(link-login)">
<input type="hidden" name="link-orig" value="$(link-orig)">
<input type="hidden" name="error" value="$(error)">
</form>
<script language="JavaScript">
<!--
        document.redirect.submit();
//-->
</script>
</body>
</html>
```

- The external server can log in a HotSpot client by redirecting it back to the original HotSpot servlet login page, specifying the correct username and password

  Here is an example of such a page (it is redirecting to https://hotspot.example.com/login, replace with the actual address of a HotSpot router; also, it is displaying www.mikrotik.com after successful login, replace with what needed):

```
<html>
<title>Hotspot login page</title>
<body>
<form name="login" action="https://hotspot.example.com/login" method="post">
<input type="text" name="username" value="demo">
<input type="password" name="password" value="none">
<input type="hidden" name="domain" value="">
<input type="hidden" name="dst" value="http://www.mikrotik.com/">
<input type="submit" name="login" value="log in">
</form>
</body>
</html>
```

- Hotspot will ask RADIUS server whether to allow the login or not. If allowed, alogin.html page will be displayed (it can be modified to do anything). If not allowed, flogin.html (or login.html) page will be displayed, which will redirect client back to the external authentication server.

Note: as shown in these examples, HTTPS protocol and POST method can be used to secure communications.

HTTP header detection

The Hotspot login pages have access to HTTP headers by using **$(http-header-name);**

For example, there exists an ability to check the user agent (or browser), and will return any other content instead of the regular login page, if so desired. This can be used to disable automatic popups in phones, for example.

For example, to output "SUCCESS" for users of a specific Firefox mobile version, instead of the login page, you can these lines on the top of the **rlogin.html** page in your hotspot directory:

```
$(if user-agent == "Mozilla/5.0 (Android; Mobile; rv:40.0) Gecko/40.0 Firefox/40.0" )
<HTML><HEAD><TITLE>Success</TITLE></HEAD><BODY>Success</BODY></HTML>
$(else)
---- regular content of rlogin.html page  ----
$(endif)
```

This will DISABLE the login popup for Android Firefox 40 users.

One click login

It is possible to create modified captive portal for quick one click login for scenarios where no user or password is required.

What you need to do is:

- Create user for this purpose. In example it is "notsosecretuser" with password "notsosecretpass"
- Assign this user to user profile that allows specific/unlimited amount of simultaneous active users.
- Copy original hotspot directory that is already generated in routers file menu on root level.

- Modify contents of this copy directory contents.
  - Only one file requires modifications for this to work, the "login.html".

Original:

```
<table width="100" style="background-color: #ffffff">
  <tr><td align="right">login</td>
      <td><input style="width: 80px" name="username" type="text" value="$(username)"/></td>
  </tr>
  <tr><td align="right">password</td>
      <td><input style="width: 80px" name="password" type="password"/></td>
  </tr>
  <tr><td> </td>
      <td><input type="submit" value="OK" /></td>
  </tr>
</table>
```

Modified:

```
<table width="100" style="background-color: #ffffff">
  <tr style="display:none;"><td align="right">login</td>
    <td><input style="width: 80px" name="username" type="text" value="notsosecretuser"/></td>
  </tr>
  <tr style="display:none;"><td align="right">password</td>
    <td><input style="width: 80px" name="password" type="password" value="notsosecretpass"/></td>
  </tr>
  <tr><td> </td>
    <td><input type="submit" value="Proceed to Internet!" /></td>
  </tr>
</table>
```

What changed:

-
  -
    - User and Password "<tr>" fields are hidden.
    - Both User and Password field values contain predefined values.
    - Changed "OK" button value(name) to something more fitting.

- Now upload this new hotspot folder back to router, preferably with different name.
- Change settings in hotspot server profile to use this new html directory.

```
/ip hotspot profile set (profile number or name) html-directory-override=(dir path/name)
```

Firewall customizations

Summary

Apart from the obvious dynamic entries in the /ip hotspot submenu itself (like hosts and active users), some additional rules are added in the firewall tables when activating a HotSpot service. Unlike RouterOS version 2.8, there are relatively few firewall rules added in the firewall as the main job is made by the one-to-one NAT algorithm.

NAT

From **/ip firewall nat print dynamic** command, you can get something like this (comments follow after each of the rules):

```
0 D chain=dstnat action=jump jump-target=hotspot hotspot=from-client
```

Putting all HotSpot-related tasks for packets from all HotSpot clients into a separate chain.

```
1 I chain=hotspot action=jump jump-target=pre-hotspot
```

Any actions that should be done before HotSpot rules apply, should be put in the pre-hotspot chain. This chain is under full administrator control and does not contain any rules set by the system, hence the invalid jump rule (as the chain does not have any rules by default).

```
2 D chain=hotspot action=redirect to-ports=64872 dst-port=53 protocol=udp
3 D chain=hotspot action=redirect to-ports=64872 dst-port=53 protocol=tcp
```

Redirect all DNS requests to the HotSpot service. The 64872 port provides DNS service for all HotSpot users. If you want HotSpot server to listen also to another port, add rules here the same way, changing dst-port property.

```
4 D chain=hotspot action=redirect to-ports=64873 hotspot=local-dst dst-port=80
    protocol=tcp
```

Redirect all HTTP login requests to the HTTP login servlet. The 64873 is HotSpot HTTP servlet port.

```
5 D chain=hotspot action=redirect to-ports=64875 hotspot=local-dst dst-port=443
    protocol=tcp
```

Redirect all HTTPS login requests to the HTTPS login servlet. The 64875 is HotSpot HTTPS servlet port.

```
6 D chain=hotspot action=jump jump-target=hs-unauth hotspot=!auth protocol=tcp
```

All other packets except DNS and login requests from unauthorized clients should pass through the hs-unauth chain.

```
7 D chain=hotspot action=jump jump-target=hs-auth hotspot=auth protocol=tcp
```

And packets from the authorized clients - through the hs-auth chain.

```
8 D ;;; www.mikrotik.com
    chain=hs-unauth action=return dst-address=66.228.113.26 dst-port=80 protocol=tcp
```

First in the **hs-unauth** chain is put everything that affects TCP protocol in the `/ip hotspot walled-garden ip` submenu (i.e., everything where either protocol is not set, or set to TCP). Here we are excluding www.mikrotik.com from being redirected to the login page.

```
9 D chain=hs-unauth action=redirect to-ports=64874 dst-port=80 protocol=tcp
```

All other HTTP requests are redirected to the Walled Garden proxy server which listens the 64874 port. If there is an allow entry in the `/ip hotspot walled-garden` menu for an HTTP request, it is being forwarded to the destination. Otherwise, the request will be automatically redirected to the HotSpot login servlet (port 64873).

```
10 D chain=hs-unauth action=redirect to-ports=64874 dst-port=3128 protocol=tcp
11 D chain=hs-unauth action=redirect to-ports=64874 dst-port=8080 protocol=tcp
```

HotSpot by default assumes that only these ports may be used for HTTP proxy requests. These two entries are used to "catch" client requests to unknown proxies (you can add more rules here for other ports). I.e., to make it possible for the clients with unknown proxy settings to work with the HotSpot system. This feature is called "Universal Proxy". If it is detected that a client is using some proxy server, the system will automatically mark that packets with the http hotspot mark to work around the unknown proxy problem, as we will see later on. Note that the port used (64874) is the same as for HTTP requests in the rule #9 (so both HTTP and HTTP proxy requests are processed by the same code).

```
12 D chain=hs-unauth action=redirect to-ports=64875 dst-port=443 protocol=tcp
```

HTTPS proxy is listening on the 64875 port.

```
13 I chain=hs-unauth action=jump jump-target=hs-smtp dst-port=25 protocol=tcp
```

Redirect for SMTP protocol may also be defined in the HotSpot configuration. In case it is, a redirect rule will be put in the hs-smtp chain. This is done so that users with unknown SMTP configuration would be able to send their mail through the service provider's (your) SMTP server instead of going to the [possibly unavailable outside their network of origin] SMTP server users have configured on their computers. The chain is empty by default, hence the invalid jump rule.

```
14 D chain=hs-auth action=redirect to-ports=64874 hotspot=http protocol=tcp
```

Providing HTTP proxy service for authorized users. Authenticated user requests may need to be subject to transparent proxying (the "Universal Proxy" technique and advertisement feature). This http mark is put automatically on the HTTP proxy requests to the servers detected by the HotSpot HTTP proxy (the one that is listening on the 64874 port) as HTTP proxy requests for unknown proxy servers. This is done so that users that have some proxy settings would use the HotSpot gateway instead of the [possibly unavailable outside their network of origin] proxy server users have configured in their computers. This mark is also applied when advertisement is due to be shown to the user, as well as on any HTTP requests done form the users whose profile is configured to transparently proxy their requests.

```
15 I chain=hs-auth action=jump jump-target=hs-smtp dst-port=25 protocol=tcp
```

Providing SMTP proxy for authorized users (the same as in rule #13).

Packet Filtering

From **/ip firewall filter print dynamic** command, you can get something like this (comments follow after each of the rules):

```
 0 D chain=forward action=jump jump-target=hs-unauth hotspot=from-client,!auth
```

Any packet that traverse the router from an unauthorized client will be sent to the **hs-unauth** chain. The hs-unauth implements the IP-based Walled Garden filter.

```
 1 D chain=forward action=jump jump-target=hs-unauth-to hotspot=to-client,!auth
```

Everything that comes to clients through the router, gets redirected to another chain, called **hs-unauth-to**. This chain should reject unauthorized requests to the clients.

```
2 D chain=input action=jump jump-target=hs-input hotspot=from-client
```

Everything that comes from clients to the router itself, gets to yet another chain, called **hs-input**.

```
3 I chain=hs-input action=jump jump-target=pre-hs-input
```

Before proceeding with [predefined] dynamic rules, the packet gets to the administratively controlled **pre-hs-input** chain, which is empty by default, hence the invalid state of the jump rule.

```
4 D chain=hs-input action=accept dst-port=64872 protocol=udp
5 D chain=hs-input action=accept dst-port=64872-64875 protocol=tcp
```

Allow client access to the local authentication and proxy services (as described earlier).

```
6 D chain=hs-input action=jump jump-target=hs-unauth hotspot=!auth
```

All other traffic from unauthorized clients to the router itself will be treated the same way as the traffic traversing the routers.

```
7 D chain=hs-unauth action=return protocol=icmp
8 D ;;; www.mikrotik.com
    chain=hs-unauth action=return dst-address=66.228.113.26 dst-port=80 protocol=tcp
```

Unlike NAT table where only TCP-protocol related Walled Garden entries were added, in the packet filter **hs-unauth** chain is added everything you have set in the **/ip hotspot walled-garden ip** menu. That is why although you have seen only one entry in the NAT table, there are two rules here.

```
 9 D chain=hs-unauth action=reject reject-with=tcp-reset protocol=tcp
10 D chain=hs-unauth action=reject reject-with=icmp-net-prohibited
```

Everything else that has not been while-listed by the Walled Garden will be rejected. Note usage of TCP Reset for rejecting TCP connections.

```
11 D chain=hs-unauth-to action=return protocol=icmp
12 D ;;; www.mikrotik.com
     chain=hs-unauth-to action=return src-address=66.228.113.26 src-port=80 protocol=tcp
```

Same action as in rules #7 and #8 is performed for the packets destined to the clients (chain **hs-unauth-to**) as well.

```
13 D chain=hs-unauth-to action=reject reject-with=icmp-host-prohibited
```

Reject all packets to the clients with ICMP reject message.

[Top | Back to Content **]**