opennds Documentation

Release 9.0.0

the opennds contributors

May 12, 2021

Contents

openNDS is a high performance, small footprint Captive Portal, offering by default a simple splash page restricted Internet connection, yet incorporates an API that allows the creation of sophisticated authentication applications.

It is a fork of the NoDogSplash project that in turn was derived originally from the codebase of the Wifi Guard Dog project.

openNDS is released under the GNU General Public License.

- openNDS: https://github.com/openNDS/openNDS
- Original Homepage *down*: http://kokoro.ucsd.edu/nodogsplash
- Archive: https://web.archive.org/web/20140210131130/http://kokoro.ucsd.edu/nodogsplash
- Wifidog: http://dev.wifidog.org/
- NoDogSplash: https://github.com/nodogsplash/nodogsplash
- GNU GPL: http://www.gnu.org/copyleft/gpl.html

The following describes what openNDS does, how to get it and run it, and how to customize its behavior for your application.

Contents:

CHAPTER 1

Overview

openNDS (NDS) is a high performance, small footprint Captive Portal, offering by default a simple splash page restricted Internet connection, yet incorporates an API that allows the creation of sophisticated authentication applications.

1.1 Captive Portal Detection (CPD)

All modern mobile devices, most desktop operating systems and most browsers now have a CPD process that automatically issues a port 80 request on connection to a network. NDS detects this and serves a special "**splash**" web page to the connecting client device.

1.2 Provide simple and immediate public Internet access

NDS provides two selectable ready to run methods.

- **Click to Continue**. A simple "Click to Continue" dynamic splash page sequence requiring a client user to accept Terms of Service before continuing to access the Internet (*default*). Basic client device information is recorded in a log file.
- **username/email-address login**. A simple dynamic splash page sequence that requires the client user to enter their username and email-address before accepting the Terms of Service. A welcome page and landing page that can carry an information or advertising payload are served to the client in sequence. Client user and client device information is recorded in the log file. (*This mode is selected in the configuration file*)

  Both these modes are generated using default ThemeSpec script files. These ThemeSpec script files contain easy to edit html blocks, allowing basic content changes to be made very simply.

  Modifying the content seen by users is a simple matter of editing the html blocks within the script file.

Additional more advanced ThemeSpec files are included and can also be enabled from the config file. These additional files make use of the custom_parameters, custom_images and custom_files config options. Form input fields and text comments can be added, and images and content blocks can be downloaded on demand from remote servers.

1.3 Write Your Own Captive Portal.

NDS can be used as the "Engine" behind the most sophisticated Captive Portal systems using the tools provided.

- **Forward Authentication Service (FAS)**. FAS provides pre-authentication user validation in the form of a set of dynamic web pages, typically served by a web service independent of NDS, located remotely on the Internet, on the local area network or on the NDS router.
- **PreAuth**. A special case of FAS that runs locally on the NDS router with dynamic html served by NDS itself. This requires none of the overheads of a full FAS implementation and is ideal for NDS routers with limited RAM and Flash memory.
- **BinAuth**. A method of running a post authentication script or extension program.

CHAPTER 2

What's New? - ChangeLog

openNDS (9.0.0)

- This version introduces major new functionality, some changes and fixes
- Add - post-request - add upstream payload [bluewavenet]
- Add - post-request - base64 encode payload [bluewavenet]
- Add - authmon add more status checking and default to view mode for upstream processing [bluewavenet]
- Add - authmon add housekeeping call, limit concurrent authentications, send auth-ack [bluewavenet]
- Add - fas-aes-https add housekeeping call, add auth-ack support, add "try again" button [bluewavenet]
- Add - "$" character added to htmlentityencode [bluewavenet]
- Add - Theme support - theme_click-to-continue [bluewavenet]
- Add - Themespec, custom variables and custom images options to OpenWrt config [bluewavenet]
- Add - Support for ThemeSpecPath, FasCustomParametersList, FasCustomVariablesList, FasCustomImagesList [bluewavenet]
- Add - Example theme - click-to-continue-custom-placeholders [bluewavenet]
- Add - Increase Buffer sizes to support custom parameters [bluewavenet]
- Add - themespec_path argument [bluewavenet]
- Add - Increase buffers for custom vars and images [bluewavenet]
- Add - Increase command buffer for custom vars and images [bluewavenet]
- Add - Increase HTMLMAXSIZE [bluewavenet]
- Add - Use MAX_BUF for fasparam, fasvar and fasimage [bluewavenet]
- Add - support for ThemeSpec files and placeholders [bluewavenet]
- Add - Theme Click to Continue with Custom Placeholders [bluewavenet]

- Add - make custom field a required entry [bluewavenet]
- Add - bash/ash check and simplify image download config [bluewavenet]
- Add - example custom images and text placeholders to click-to-continue-custom [bluewavenet]
- Add - theme_user-email-login-custom-placeholders [bluewavenet]
- Add - Status page for login failure [bluewavenet]
- Add - fas_custom_files_list and update Makefiles [bluewavenet]
- Add - Autoconfiguration of ndsctl socket file to use tmpfs mountpoint [bluewavenet]
- Add - example custom images and custom html [bluewavenet]
- Add - Set default gateway interface br-lan [bluewavenet]
- Add - libopennds, set wget timeout [bluewavenet]
- Add - allow disabling of gatewayfqdn [bluewavenet]
- Add - packet rate limiting for upload/download rate quotas [bluewavenet]
- Add - get custom resources from Github branch [bluewavenet]
- Add - functions start_mhd() and stop_mhd() [bluewavenet]
- Add - MHD Watchdog - restart MHD if required [bluewavenet]
- Add - Pause and retry popen on failure [bluewavenet]
- Add - function get_key_from_config() [bluewavenet]
- Remove - deprecated traffic control code [bluewavenet]
- Remove - deprecated binauth scripts [bluewavenet]
- Remove - deprecated legacy splash page support [bluewavenet]
- Remove - deprecated ndsctl clients [bluewavenet]
- Remove - outdated PreAuth scripts [bluewavenet]
- Refactor - Move hid to head of query string [bluewavenet]
- Refactor - Move libopennds to libs
- Fix - ndsctl auth crashed opennds if session duration argument was null [bluewavenet]
- Fix - fas-aes-https - correctly set path for authlist for most server types [bluewavenet]
- Fix - suppress BinAuth syslog notice message [bluewavenet]
- Fix - setting gw_fqdn in hosts file if gw_ip is changed [bluewavenet]
- Fix - add missing comma before trusted list in ndsctl json [bluewavenet] [gueux]
- Fix - Improve Shell detection [bluewavenet]
- Fix - Improve b64decode performance [bluewavenet]
- Fix - ndsctl -s option [bluewavenet] [gueux]
- Fix - Adjust config defaults to good real world values [bluewavenet]
- Fix - don't override ndsparamlist in ThemeSpec files [bluewavenet]
- Fix - Check ndsctl lock to prevent calling from Binauth [bluewavenet]
- Fix - Clean up syslog messages at info level (2) [bluewavenet]

- Fix - Debian changelog format to allow package building [bluewavenet]
- Fix - numerous compiler errors and BASH compatibility issues [bluewavenet]
- Fix - ndsctl auth, ensure if session timeout = 0 then use global value [bluewavenet]
- Fix - setting of gatewayport, caused by typo in conf.c [bluewavenet] [Ethan-Yami]
- Fix - remove unused credential info from log [bluewavenet]
- Deprecate - the legacy opennds.conf file [bluewavenet]

—Rob White <dot@blue-wave.net> Thu, 2 May 2021 17:32:43 +0000

openNDS (8.1.1)

- Fix - remove legacy code where option preauthenticated_users containing the keyword "block" would cause openNDS to fail to start [bluewavenet]

—Rob White <dot@blue-wave.net> Thu, 21 Feb 2021 16:33:34 +0000

openNDS (8.1.0)

- This version introduces some new functionality and some fixes/enhancements
- Fix - Add default values for gatewayfqdn. If not set in config could result in crash on conection of first client [bluewavenet]
- Add - Authenticated users are now granted access to the router by entry in "list authenticated_users" [bluewavenet]
- Fix - option preauth was being ignored [bluewavenet]
- Add - query string validity check and entity encode "$" character. Generate error 511 if query string is corrupted [bluewavenet]
- Add - a "Try Again" button to the login.sh script, to be displayed if the client token has expired before login. [bluewavenet]

—Rob White <dot@blue-wave.net> Thu, 18 Feb 2021 17:03:23 +0000

openNDS (8.0.0)

- This version introduces major new functionality and some major changes
- Rationalisation of support for multiple Linux distributions [bluewavenet]
- Refactor login.sh script introducing base64 encoding and hashed token (hid) support [bluewavenet]
- Refactor fas-hid script introducing base64 encoding and simplifying customisation of the script [bluewavenet]
- Refactor binauth_log.sh and log BinAuth custom data as url encoded [bluewavenet]
- Refactor fas-aes, simplifying customisation of the script [bluewavenet]
- Refactor fas-aes-https, simplifying customisation of the script [bluewavenet]
- Change - Use hid instead of tok when fas_secure_enabled >= 1 [bluewavenet]
- Add - base64 encoding to fas_secure_enabled level 1 [bluewavenet]
- Add - gatewyname, clientif, session_start, session_end and last_active to ndsctl json [bluewavenet]
- Add - support for RFC6585 Status Code 511 - Network Authentication Required [bluewavenet]
- Add - Client Status Page UI with Logout [bluewavenet]
- Add - GatewayFQDN option [bluewavenet]

- Add - client interface to status page query string [bluewavenet]
- Add - support using base 64 encoded custom string for BinAuth and replace tok with hid [bluewavenet]
- Add - base 64 decode option to ndsctl [bluewavenet]
- Add - b64 encoding of querystring for level 1 [bluewavenet]
- Add - Improved performance/user-experience on congested/slow systems using php FAS scripts [bluewavenet]
- Add - support for ndsctl auth by hid in client_list [bluewavenet]
- Add - Ensure faskey is set to default value (always enabled) [bluewavenet]
- Add - Display error page on login failure in login.sh [bluewavenet]
- Add - splash.html, add deprecation notice [bluewavenet]
- Add - authmon, improved lock checking and introduce smaller loopinterval [bluewavenet]
- Add - client_params, wait for ndsctl if it is busy [bluewavenet]
- Add - fas-aes-https, allow progressive output to improve user experience on slow links [bluewavenet]
- Fix - Block access to /opennds_preauth/ if PreAuth not enabled [bluewavenet]
- Fix - On startup, call iptables_fw_destroy before doing any other setup [bluewavenet]
- Fix - missing final redirect to originurl in fas-hid [bluewavenet]
- Fix - ensure gatewayname is always urlencoded [bluewavenet]
- Fix - client session end not set by binauth [bluewavenet]
- Fix - Session timeout, if client setting is 0, default to global value [bluewavenet]
- Fix - missing trailing separator on query and fix some compiler errors [bluewavenet]
- Fix - ensure authmon daemon is killed if left running from previous crash [bluewavenet]
- Fix - add missing query separator for custom FAS parameters [bluewavenet]
- Fix - ndsctl auth, do not set quotas if client is already authenticated [bluewavenet]
- Fix - client_params, show “Unlimited” when “null” is received from ndsctl json [bluewavenet]
- Update configuration files [bluewavenet]
- update documentation [bluewavenet]

—Rob White <dot@blue-wave.net> Sat, 2 Jan 2021 16:38:14 +0000

openNDS (7.0.1)

- This version contains fixes and some minor updates
- Fix - Failure of Default Dynamic Splash page on some operating systems [bluewavenet]
- Fix - A compiler warning - some compiler configurations were aborting compilation [bluewavenet]
- Update - Added helpful comments in configuration files [bluewavenet]
- Remove - references to deprecated RedirectURL in opennde.conf [bluewavenet]
- Update - Documentation updates and corrections [bluewavenet]

—Rob White <dot@blue-wave.net> Wed, 7 Nov 2020 12:40:33 +0000

openNDS (7.0.0)

- This version introduces major new enhancements and the disabling or removal of deprecated functionality
- Fix - get_iface_ip in case of interface is vif or multihomed [bluewavenet]
- Fix - Add missing client identifier argument in ndsctl help text [bluewavenet]
- Deprecate - ndsctl clients option [bluewavenet]
- Add - global quotas to output of ndsctl status [bluewavenet]
- Fix - fix missing delimiter in fas-hid [bluewavenet]
- Add - Report Rate Check Window in ndsctl status and show client quotas [bluewavenet]
- Add - Quota and rate reporting to ndsctl json. Format output and fix json syntax errors [bluewavenet]
- Fix - get_client_interface for case of iw utility not available [bluewavenet]
- Fix - php notice for pedantic php servers in post-request [bluewavenet]
- Add - built in autonomous Walled Garden operation [bluewavenet]
- Remove - support for deprecated RedirectURL [bluewavenet]
- Add - gatewaymac to the encrypted query string [bluewavenet]
- Deprecate - legacy splash.html and disable it [bluewavenet]
- Add - support for login mode in PreAuth [bluewavenet]
- Add - Support for Custom Parameters [bluewavenet]

—Rob White <dot@blue-wave.net> Wed, 5 Nov 2020 18:22:32 +0000

openNDS (6.0.0)

- This version - for Openwrt after 19.07 - for compatibility with new MHD API
- Set - minimum version of MHD to 0.9.71 for new MHD API [bluewavenet]
- Set - use_outdated_mhd to 0 (disabled) as default [bluewavenet]
- Add - Multifield PreAuth login script with css update [bluewavenet]
- Add - Documentation and config option descriptions for configuring Walled Garden IP Sets

—Rob White <dot@blue-wave.net> Wed, 21 Aug 2020 15:43:47 +0000

openNDS (5.2.0)

- This version - for backport to Openwrt 19.07 - for compatibility with old MHD API
- Fix - Failure of MHD with some operating systems eg Debian [bluewavenet]
- Fix - potential buffer truncation in ndsctl
- Set - use_outdated_mhd to 1 (enabled) as default [bluewavenet]
- Set - maximum permissible version of MHD to 0.9.70 to ensure old MHD API is used [bluewavenet]

—Rob White <dot@blue-wave.net> Wed, 12 Aug 2020 17:43:57 +0000

openNDS (5.1.0)

- Add - Generic Linux - install opennds.service [bluewavenet]
- Add - Documentation updates [bluewavenet]
- Add - config file updates [bluewavenet]

- Add - Install sitewide username/password splash support files [bluewavenet]
- Add - quotas to binauth_sitewide [bluewavenet]
- Add - Splash page updates [bluewavenet]
- Add - Implement Rate Quotas [bluewavenet]
- Fix - check if idle preauthenticated [bluewavenet]
- Add - support for rate quotas [bluewavenet]
- Fix - Correctly compare client counters and clean up debuglevel messages [bluewavenet]
- Add - Implement upload/download quotas Update fas-aes-https to support quotas [bluewavenet]
- Add - Rename demo-preauth scripts and install all scripts [bluewavenet]
- Add - fas-aes-https layout update [bluewavenet]
- Add - Set some defaults in fas-aes-https [bluewavenet]
- Add - custom data string to ndsctl auth [bluewavenet]
- Add - custom data string to fas-hid.php [bluewavenet]
- Add - Send custom data field to BinAuth via auth_client method [bluewavenet]
- Fix - missing token value in auth_client [bluewavenet]
- Add - upload/download quota and rate configuration values [bluewavenet]
- Add - Send client token to binauth [bluewavenet]
- Add - Rename upload_limit and download_limit to upload_rate and download_rate [bluewavenet]
- Fix - Pass correct session end time to binauth [bluewavenet]
- Add - some debuglevel 3 messages [bluewavenet]
- Add - description of the favicon and page footer images [bluewavenet]
- Add - Authmon collect authentication parameters from fas-aes-https [bluewavenet]
- Add - sessionlength to ndsctl auth [bluewavenet]
- Fix - Page fault when ndsctl auth is called and client not found [bluewavenet]
- Add - Enable BinAuth / fas_secure_enabled level 3 compatibility [bluewavenet]
- Fix - Correctly set BinAuth session_end [bluewavenet]
- Add - Updates to Templated Splash pages [bluewavenet]
- Add - Community Testing files [bluewavenet]
- Fix - BinAuth error passing client session times [bluewavenet]
- Fix - PHP notice - undefined constant [bluewavenet]
- Fix - OpenWrt CONFLICTS variable in Makefile [bluewavenet]

—Rob White <dot@blue-wave.net> Wed, 24 Jun 2020 20:55:18 +0000

openNDS (5.0.1)

- Fix - Path Traversal Attack vulnerability allowed by libmicrohttpd's built in unescape functionality [bluewavenet] [lynxis]

—Rob White <dot@blue-wave.net> Wed, 06 May 2020 19:56:27 +0000

openNDS (5.0.0)

- Import - from NoDogSplash 4.5.0 allowing development without compromising NoDogSplash optimisation for minimum resource utilisation [bluewavenet]
- Rename - from NoDogSplash to openNDS [bluewavenet]
- Create - openNDS avatar and splash image [bluewavenet]
- Move - wait_for_interface to opennds C code ensuring consistent start at boot time for all hardware, OpenWrt and Debian [bluewavenet]
- Add - Enable https protocol for remote FAS [bluewavenet]
- Add - trusted devices list to ndsctl json output [bluewavenet]
- Add - option unescape_callback_enabled [bluewavenet]
- Add - get_client_token library utility [bluewavenet]
- Add - utf-8 to PreAuth header [bluewavenet]
- Add - PreAuth Support for hashed id (hid) if sent by NDS [bluewavenet]
- Add - library script shebang warning for systems not running Busybox [bluewavenet]
- Add - htmlentityencode function, encode gatewayname in templated splash page [bluewavenet]
- Add - htmlentity encode gatewayname on login page (PreAuth) [bluewavenet]
- Add - Simple customisation of log file location for PreAuth and BinAuth [bluewavenet]
- Add - option use_outdated_mhd [bluewavenet]
- Add - url-encode and htmlentity-encode gatewayname on startup [bluewavenet]
- Add - Allow special characters in username (PreAuth) [bluewavenet]
- Add - Documentation updates [bluewavenet]
- Add - Various style and cosmetic updates [bluewavenet]
- Fix - Change library script shebang to bash in Debian [bluewavenet]
- Fix - Remove unnecessary characters causing script execution failure in Debian [bluewavenet]
- Fix - Add missing NULL parameter in MHD_OPTION_UNESCAPE_CALLBACK [skra72] [bluewavenet]
- Fix - Script failures running on Openwrt 19.07.0 [bluewavenet]
- Fix - Preauth, status=authenticated [bluewavenet]
- Fix - Prevent ndsctl from running if called from a Binauth script. [bluewavenet]
- Fix - Minor changes in Library scripts for better portability [bluewavenet]
- Fix - Prevent php notices on pedantic php servers [bluewavenet]
- Fix - broken remote image retrieval (PreAuth) [bluewavenet]
- Fix - Allow use of "#" in gatewayname [bluewavenet]

—Rob White <dot@blue-wave.net> Sat, 03 Apr 2020 13:23:36 +0000

CHAPTER 3

Installing openNDS

3.1 Prerequisites

openNDS is designed to run on a device configured as an IPv4 router and will have at least two network interfaces:

A WAN interface (Wide Area Network). This interface must be connected to an Internet feed:

- Either an ISP CPE (Internet Service Provider Customer Premises Equipment)
- Or another router, such as the venue ADSL router.
- It must be configured as a DHCP client, obtaining its IPv4 address and DNS server from the connected network.

A LAN interface (Local Area Network). This interface MUST be configured to:

- Provide the Default IPv4 gateway in a private IPv4 subnet that is different to any private subnets between it and the ISP CPE
- Provide DHCP services to connected clients
- Provide DNS services to connected clients
- Provide Network Address Translation (NAT) for all outgoing traffic directed to the WAN interface.

3.2 Installing on OpenWrt

- Have a router working with OpenWrt. At the time of writing, openNDS has been tested with OpenWrt 18.06.x, 19.7.x 21.2.x and Snapshot.
- OpenWrt version 19.07.x or less requires theoogle.com updated libmicrohttpd-no-ssl_0.9.71-1 package (or higher version) from Openwrt 21.2.x or higher.
- openNDS may or may not work on older versions of OpenWrt.

- Make sure your router is working correctly before you try to install openNDS. In particular, make sure your DHCP daemon is serving addresses on the interface that openNDS will manage.

  The default interface is br-lan but can be changed to any LAN interface by editing the /etc/config/opennds file.

- To install openNDS on 21.3.x or higher, you may use the OpenWrt Luci web interface or alternatively, ssh to your router and run the command:

  ```
  opkg update
  ```

  followed by

  ```
  opkg install opennds
  ```

- **To install on 19.7.x or lower, download libmicrohttpd-no-ssl_0.9.71-1 or higher to the /tmp directory on your router. Then**
  `opkg install <packagename> --force-reinstall`
- **Similarly, download openNDS v9.0.0 or higher to /tmp, again installing using the command:** `opkg install <packagename> --force-reinstall`
- openNDS is enabled by default and will start automatically on reboot or can be started and stopped manually.
- If the interface that you want openNDS to manage is not br-lan, edit /etc/config/opennds and set GatewayInterface.
- To start openNDS, run the following, or just reboot the router:

  ```
  service opennds start
  ```

- To test the installation, connect a client device to the interface on your router that is managed by openNDS (for example, connect to the router's wireless lan).

  Most client device operating systems and browsers support Captive Portal Detection (CPD) and the operating system or browser on that device will attempt to contact a pre defined port 80 web page.

  CPD will trigger openNDS to serve the default splash page where you can click or tap Continue to access the Internet.

  See the Authentication section for details of setting up a proper authentication process.

  If your client device does not display the splash page it most likely does not support CPD.

  You should then manually trigger openNDS by trying to access a port 80 web site (for example, http://gnome.org is a good choice).

- To stop openNDS:

  ```
  service opennds stop
  ```

- To uninstall openNDS:

  ```
  opkg remove opennds
  ```

3.3 Generic Linux

openNDS and libmicrohttps-ssl can be compiled in place for most distributions of Linux.

To compile libmicrohttpd and openNDS, see the chapter "How to Compile and install openNDS".

CHAPTER 4

How openNDS (NDS) works

openNDS is a Captive Portal Engine. Any Captive Portal, including NDS, will have two main components:

- Something that does the capturing, and
- Something to provide a Portal for client users to log in.

openNDS MUST run on a device configured as an IPv4 router.

A wireless router will typically be running OpenWrt or some other Linux distribution.

A router, by definition, will have two or more interfaces, at least one to connect to the wide area network (WAN) or Internet feed, and at least one connecting to the local area network (LAN).

Each LAN interface must also act as the Default IP Gateway for its LAN, ideally with the interface serving IP addresses to client devices using DHCP.

Multiple LAN interfaces can be combined into a single bridge interface. For example, ethernet, 2.4Ghz and 5Ghz networks are typically combined into a single bridge interface. Logical interface names will be assigned such as eth0, wlan0, wlan1 etc. with the combined bridge interface named as br-lan.

NDS will manage one or more of them of them. This will typically be br-lan, the bridge to both the wireless and wired LAN, but could be, for example, wlan0 if you wanted NDS to work just on the wireless interface.

4.1 Summary of Operation

By default, NDS blocks everything, but intercepts port 80 requests.

An initial port 80 request will be generated on a client device, usually automatically by the client device's built in Captive Portal Detection (CPD), or possibly by the user manually browsing to an http web page.

This request will of course **be routed by the client device to the Default Gateway** of the local network. The Default Gateway will, as we have seen, be the router interface that NDS is managing.

4.1.1 The Thing That Does the Capturing (NDS)

As soon as this initial port 80 request is received on the default gateway interface, NDS will "Capture" it, make a note of the client device identity, allocate a unique token for the client device, then redirect the client browser to the Portal component of NDS.

4.1.2 The Thing That Provides the Portal (FAS or PreAuth)

The client browser is redirected to the Portal component. This is a web service that is configured to know how to communicate with the core engine of openNDS.

This is commonly known as the Splash Page, or more correctly, the entry point to the portal splash page sequence.

openNDS has its own web server built in and this can be used to serve the Portal "Splash" pages to the client browser, or a separate web server can be used.

openNDS comes with numerous standard Splash Page Sequence options pre-installed.

One provides a trivial Click to Continue splash page and another provides a Client User form requiring Name and Email address to be entered.

All of these can be customised or a complete specialised Portal can be written by the installer (See FAS, PreAuth).

FAS, or Forward Authentication Service may use the web server embedded in openNDS, a separate web server installed on the NDS router, a web server residing on the local network or an Internet hosted web server.

The user of the client device will always be expected to complete some actions on the splash page.

Once the user on the client device has successfully completed the splash page actions, that page then links directly back to NDS.

For security, NDS expects to receive the same valid token it allocated when the client issued its initial port 80 request. If the token received is valid, NDS then "authenticates" the client device, allowing access to the Internet. openNDS uses various methods to encrypt this token, ensuring verification cannot be bypassed.

Post authentication processing extensions may be added to openNDS (See BinAuth). Once openNDS has received a valid token it calls a BinAuth script if enabled.

If the BinAuth script returns positively (ie return code 0), NDS then "authenticates" the client device, allowing access to the Internet.

Note: FAS and Binauth can be enabled together. This can give great flexibility, with FAS providing remote verification and Binauth providing local post authentication processing closely linked to openNDS.

4.2 Captive Portal Detection (CPD)

All modern mobile devices, most desktop operating systems and most browsers now have a CPD process that automatically issues a port 80 request on connection to a network. NDS detects this and serves a special "splash" web page to the connecting client device.

The port 80 html request made by the client CPD can be one of many vendor specific URLs.

Typical CPD URLs used are, for example:

- *http://captive.apple.com/hotspot-detect.html*
- *http://connectivitycheck.gstatic.com/generate_204*
- *http://connectivitycheck.platform.hicloud.com/generate_204*
- *http://www.samsung.com/*
- *http://detectportal.firefox.com/success.txt*
- Plus many more

It is important to remember that CPD is designed primarily for mobile devices to automatically detect the presence of a portal and to trigger the login page, without having to resort to breaking SSL/TLS security by requiring the portal to redirect port 443 for example.

Just about all current CPD implementations work very well but some compromises are necessary depending on the application.

The vast majority of devices attaching to a typical Captive Portal are mobile devices. CPD works well giving the initial login page.

For a typical guest wifi, eg a coffee shop, bar, club, hotel etc., a device connects, the Internet is accessed for a while, then the user takes the device out of range.

When taken out of range, a typical mobile device begins periodically polling the wireless spectrum for SSIDs that it knows about to try to obtain a connection again, subject to timeouts to preserve battery life.

Most Captive Portals have a session duration limit (NDS included).

If a previously logged in device returns to within the coverage of the portal, the previously used SSID is recognised and CPD is triggered and tests for an Internet connection in the normal way. Within the session duration limit of the portal, the Internet connection will be established, if the session has expired, the splash page will be displayed again.

Early mobile device implementations of CPD used to poll their detection URL at regular intervals, typically around 30 to 300 seconds. This would trigger the Portal splash page quite quickly if the device stayed in range and the session limit had been reached.

However it was very quickly realised that this polling kept the WiFi on the device enabled continuously having a very negative effect on battery life, so this polling whilst connected was either increased to a very long interval or removed all together (depending on vendor) to preserve battery charge. As most mobile devices come and go into and out of range, this is not an issue.

A common issue raised is:

My devices show the splash page when they first connect, but when the authorization expires, they just announce there is no internet connection. I have to make them "forget" the wireless network to see the splash page again. Is this how is it supposed to work?

The workaround is as described in the issue, or even just manually disconnecting or turning WiFi off and on will simulate a "going out of range", initialising an immediate trigger of the CPD. One or any combination of these workarounds should work, again depending on the particular vendor's implementation of CPD.

In contrast, most laptop/desktop operating systems, and browser versions for these still implement CPD polling whilst online as battery considerations are not so important.

For example, Gnome desktop has its own built in CPD browser with a default interval of 300 seconds. Firefox also defaults to something like 300 seconds. Windows 10 is similar.

This IS how it is supposed to work, but does involve some compromises.

The best solution is to set the session timeout to a value greater than the expected length of time a client device is likely to be present. Experience shows a limit of 24 hours covers most situations eg bars, clubs, coffee shops, motels etc. If for example an hotel has guests regularly staying for a few days, then increase the session timeout as required.

Staff at the venue could have their devices added to the Trusted List if appropriate, but experience shows, it is better not to do this as they very soon learn what to do and can help guests who encounter the issue. (Anything that reduces support calls is good!)

4.3 Network Zone Detection (Where is the Client Connected?)

Client devices can be connected to one of a number of local WiFi SSIDs, connected directly or indirectly by ethernet, or connected via a wireless mesh network. Each connection type available is considered as a Network Zone.

NDS detects which zone each client is connected to. This information can be used to dynamically customise the login for each zone.

For example a coffee shop might have two SSIDs configured:

- Staff (Secure SSID ie with access code)
- Customers (open SSID with login form)

In this example SSID "Staff" is configured on interface wlan0, and considered as Zone "Private".

However, SSID "Customers" is configured on virtual interface wlan0-1, and considered as Zone "Public".

NDS detects which zone is being used by a client and a relevant login page can be served.

4.4 Packet filtering

openNDS considers four kinds of packets coming into the router over the managed interface. Each packet is one of these kinds:

1. **Blocked**, if the MAC mechanism is block, and the source MAC address of the packet matches one listed in the BlockedMACList; or if the MAC mechanism is allow, and source MAC address of the packet does not match one listed in the AllowedMACList or the TrustedMACList. These packets are dropped.
2. **Trusted**, if the source MAC address of the packet matches one listed in the TrustedMACList. By default, these packets are accepted and routed to all destination addresses and ports. If desired, this behavior can be customized by FirewallRuleSet trusted-users and FirewallRuleSet trusted-users-to-router lists in the opennds.conf configuration file, or by the EmptyRuleSetPolicy trusted-users EmptyRuleSetPolicy trusted-users-to-router directives.
3. **Authenticated**, if the packet's IP and MAC source addresses have gone through the openNDS authentication process and has not yet expired. These packets are accepted and routed to a limited set of addresses and ports (see FirewallRuleSet authenticated-users and FirewallRuleSet users-to-router in the opennds.conf configuration file).
4. **Preauthenticated**. Any other packet. These packets are accepted and routed to a limited set of addresses and ports (see FirewallRuleSet preauthenticated-users and FirewallRuleSet users-to-router in the opennds.conf configuration file). Any other packet is dropped, except that a packet for destination port 80 at any address is redirected to port 2050 on the router, where openNDS's built in libhttpd-based web server is listening. This begins the 'authentication' process. The server will serve a splash page back to the source IP address of the packet. The user clicking the appropriate link on the splash page will complete the process, causing future packets from this IP/MAC address to be marked as Authenticated until the inactive or forced timeout is reached, and its packets revert to being Preauthenticated.

openNDS implements these actions by inserting rules in the router's iptables mangle PREROUTING chain to mark packets, and by inserting rules in the nat PREROUTING, filter INPUT and filter FORWARD chains which match on those marks.

Because it inserts its rules at the beginning of existing chains, openNDS should be insensitive to most typical existing firewall configurations.

4.5 Data volume and Rate Quotas

openNDS (NDS) has built in *Data Volume* and *Data Rate* quota support.

Data volume and data rate quotas can be set globally in the config file.

The global values can be overridden on a client by client basis as required.

4.6 Traffic Shaping

openNDS (NDS) supports Traffic Shaping (Bandwidth Limiting) using the SQM - Smart Queue Management (sqm-scripts) package, available for OpenWrt and generic Linux.

CHAPTER 5

The Splash Page Sequence

As you will see mentioned in the "How openNDS (NDS) Works" section, an initial port 80 request is generated on a client device, either by the user manually browsing to an http web page, or, more usually, automatically by the client device's built in Captive Portal Detection (CPD).

This request is intercepted by NDS and an html Splash Page Sequence is served to the user of the client device to enable them to authenticate and be granted Internet access.

5.1 Types of Splash Page

This Splash page can be one of the following:

5.1.1 A Dynamic Web Page served by the built in openNDS web server (PreAuth)

A script or executable file is called by openNDS. This script or executable will generate html code for the openNDS web server.

In openNDS this is called "PreAuth", as openNDS itself serves the splash page as a PRElude to AUTHentication.

Simple coding in the script enables a dialogue with the client user, for dissemination of information, user response and authentication.

This is an implementation of Forward Authentication Services **(FAS)**, *without the resource utilisation of a separate web server*, particularly useful for legacy devices with limited flash and RAM capacity.

5.1.2 A Dynamic Web Page served by an independent web server

This independent web server can be on the same device as openNDS, on the same Local Area Network as NDS, or on External Web Hosting Services.

A script or executable file is called by openNDS on the independent web server.

This has the advantage having the full flexibility of using readily available mainstream web servers, located anywhere, enabling full flexibility in design and implementation of the captive portal functionality, ranging from a self contained system through to a fully integrated multi site system with a common database.

5.2 The Pre-Installed Basic Splash Pages

By default, the Splash pages consist of a simple click to continue dialogue followed by a Welcome or advertising page. A simple config option allows you to select instead a Name/EmailAddress login dialogue.

In many instances, one or other of these simple methods will be all that is required, but the power of FAS, PreAuth and BinAuth can be used to create very sophisticated Captive Portal Systems.

5.3 The Legacy splash.html Static Web Page

The legacy static splash.html page has been deprecated for some time and in openNDS v 9.0.0 support has been removed.

5.4 Displaying Remote Content

5.4.1 openNDS supports the download on demand of remote content

openNDS can display content from third party web hosting, on the client user login screen.

This is ideal for serving information, banner advertising etc. and is configured simply by adding a custom_image or custom_file configuration option.

5.4.2 Walled Garden support

Sophisticated remote content can be served, with access controlled by the openNDS Walled Garden. Simple configuration options can enable such things as a Paypal payment system or access to Facebook resources.

For details see the Walled Garden Section.

CHAPTER 6

The Error511/Client Status Page

From version 8.0.0 onwards, openNDS has a Client Status Page.

This page is accessible by any connected client by accessing the default url:

http://status.client

If the client has been authenticated in the normal way by the client CPD process, a page is served displaying the Gatewayname and the Network Zone the client device is currently using.

A list of allowed quotas and current usage is displayed along with "Refresh" and "Logout" buttons.

If the client has not been authenticated, or has been deauthenticated due to timeout or quota usage, then a "*Error 511 Network Authentication Required*" page is displayed with "Refresh" and "Portal Login" buttons.

The "Portal Login" button allows the client to immediately attempt to login without waiting for the client CPD to trigger.

The URL used to access this page can be changed by setting the config option gatewayfqdn.

For best results it is recommended that gatewayfqdn is set to two words separated by a single period eg in OpenWrt:

```
option gatewayfqdn 'my.status'
```

Warning - if set, services on port 80 of the gateway will no longer be accessible (eg Luci AdminUI)

By default, the Error511/Status page will be found at `http://status.client/` by a redirection of port 80 to `http://gatewayaddress:gatewayport/`

Disable GatewayFQDN by setting the option to 'disable' ie:

```
option gatewayfqdn 'disable'
```

An alternate Useful Example:

```
option gatewayfqdn 'login.page'
```

CHAPTER 7

Custom Parameters, Variables, Images and files

Custom Parameters were first introduced in openNDS version 7. With version 9.0.0, custom variables, images and files have been added.

7.1 Custom parameters

Custom parameters are defined in the config file and are sent as fixed values to FAS in the encoded/encrypted query string where they can be parsed and used by the FAS.

This is particularly useful in a remote or centralised FAS that is serving numerous instances of openNDS at different locations/venues.

- Any number of Custom Parameters can be listed in the configuration file, but each one must be in a separate entry in the form of "param_name=param_value"
- param_name and param_value must be urlencoded if containing white space or single quotes.

For example in the OpenWrt UCI config file:

```
list fas_custom_parameters_list '<param_name1=param_value1>'
list fas_custom_parameters_list '<param_name2=param_value2>'
```

etc.

A real example might be:

```
list fas_custom_parameters_list 'location=main_building'
list fas_custom_parameters_list 'admin_email=norman@bates-motel.com>'
```

For a small text strings with spaces, replace each space with %20

The following Working Example applies to the installed ThemeSpec scripts:

theme_click-to-continue-custom-placeholders

and

theme_user-email-login-custom-placeholders

```
list fas_custom_parameters_list 'logo_message=openNDS:%20Perfect%20on%20OpenWrt!
'

list fas_custom_parameters_list 'banner1_message=BlueWave%20-%20Wireless%20Network%20Specia

list fas_custom_parameters_list 'banner2_message=HMS%20Pickle'

list fas_custom_parameters_list 'banner3_message=SeaWolf%20Cruiser%20Racer'
```

7.2 Custom Variables

Custom variables are defined in the config file and are sent as fixed placeholders to FAS in the encoded/encrypted query string where they can be parsed and used by the FAS, generally for user input via html forms added by the installer to suit the requirements of the venue.

7.2.1 Custom Dynamically Generated Form Fields

Custom Dynamically Generated Form Fields are a special case of Custom Variables.

ThemeSpec scripts can dynamically generate Form Field html and inject this into the dynamic splash page sequence.

This is achieved using a SINGLE line containing the keyword "input", in the form: fieldname:field-description:fieldtype

Numerous fields can be defined in this single "input=" line, separated by a semicolon (;).

The following Working Example applies to the installed ThemeSpec scripts:

theme_click-to-continue-custom-placeholders

and

theme_user-email-login-custom-placeholders

```
list fas_custom_variables_list 'input=phone:Phone%20Number:text;
postcode:Home%20Post%20Code:text'
```

7.3 Custom Images

Custom images are defined in the config file and are sent as fixed name/URL pairs to FAS in the encoded/encrypted query string where they can be parsed and used by the FAS, and added by the installer to suit the requirements of the venue.

The following Working Example applies to the installed ThemeSpec scripts:

theme_click-to-continue-custom-placeholders

and

theme_user-email-login-custom-placeholders

Note: These pre-installed ThemeSpec script files will automatically download remote images to volatile storage on the router.

```
list fas_custom_images_list 'logo_png=https://openwrt.org/_media/logo.png'
```

```
list fas_custom_images_list 'banner1_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerbw.jpg'

list fas_custom_images_list 'banner2_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerpickle.jpg'

list fas_custom_images_list 'banner3_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerseawolf.jpg'
```

7.4 Custom Files

Custom files are defined in the config file and are sent as fixed name/URL pairs to FAS in the encoded/encrypted query string where they can be parsed and used by the FAS, and added by the installer to suit the requirements of the venue. (in the same way as custom images)

The following Working Example applies to the installed ThemeSpec scripts:

theme_click-to-continue-custom-placeholders

and

theme_user-email-login-custom-placeholders

Note: These pre-installed ThemeSpec script files will automatically download remote files to volatile storage on the router.

```
list fas_custom_files_list 'advert1_htm=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerpickle.htm'
```

CHAPTER 8

Forwarding Authentication Service (FAS)

8.1 Overview

openNDS (NDS) has the ability to forward requests to a third party authentication service (FAS). This is enabled via simple configuration options.

These options are:

1. **fasport**. This enables Forwarding Authentication Service (FAS). Redirection is changed from the default login.sh to a separate FAS. The value is the IP port number of the FAS.
2. **fasremoteip**. If set, this is the remote ip address of the FAS, if not set it will take the value of the NDS gateway address.
3. **fasremotefqdn** If set, this is the remote fully qualified domain name (FQDN) of the FAS
4. **faspath**. This is the path from the FAS Web Root (not the file system root) to the FAS login page.
5. **fas_secure_enabled**. This can have four values, "0", "1", "2" or "3" providing different levels of security.
6. **faskey** Used in combination with fas_secure_enable level 1, 2 and 3, this is a key phrase for NDS to encrypt data sent to FAS.

Note: FAS (and Preauth/FAS) enables pre authentication processing. NDS authentication is the process that openNDS uses to allow a client device to access the Internet through the Firewall. In contrast, Forward Authentication is a process of "Credential Verification", after which FAS, if the verification process is successful, passes the client token to NDS for access to the Internet to be granted.

8.2 Using FAS

Note: All addresses (with the exception of fasremoteip) are relative to the *client* device, even if the FAS is located remotely.

When FAS is enabled, openNDS automatically configures firewall access to the FAS service.

The FAS service must serve a splash page of its own to replace the openNDS served output of the default login.sh script. For fas_secure_enable "0", "1", and "2" this is enforced as http. For fas_secure_enable level "3", it is enforced as https.

Typically, the FAS service will be written in PHP or any other language that can provide dynamic web content.

FAS can then provide an action form for the client, typically requesting login, or self account creation for login.

The FAS can be on the same device as openNDS, on the same local area network as NDS, or on an Internet hosted web server.

8.3 Security

If FAS Secure is enabled (Levels 1 (default), 2 and 3), the client authentication token is kept secret at all times. Instead, faskey is used to generate a hashed id value (hid) and this is sent by openNDS to the FAS. The FAS must then in turn generate a new return hash id (rhid) to return to openNDS in its authentication request.

If set to "0" The FAS is enforced by NDS to use **http** protocol. The client token is sent to the FAS in clear text in the query string of the redirect along with authaction and redir. This method is easy to bypass and useful only for the simplest systems where security does not matter.

If set to "1" The FAS is enforced by NDS to use **http** protocol. A base64 encoded query string containing the hid is sent to the FAS, along with the clientip, clientmac, gatewayname, client_hid, gatewayaddress, authdir, originurl, clientif and custom parameters and variables.

Should sha256sum not be available, then openNDS will terminate with an error message on startup.

If set to "2" The FAS is enforced by NDS to use **http** protocol.

clientip, clientmac, gatewayname, client_hid, gatewayaddress, authdir, originurl and clientif are encrypted using faskey and passed to FAS in the query string.

The query string will also contain a randomly generated initialization vector to be used by the FAS for decryption.

The cipher used is "AES-256-CBC".

The "php-cli" package and the "php-openssl" module must both be installed for fas_secure level 2.

openNDS does not depend on this package and module, but will exit gracefully if this package and module are not installed when this level is set.

The FAS must use the query string passed initialisation vector and the pre shared fas_key to decrypt the query string. An example FAS level 2 php script (fas-aes.php) is stored in the /etc/opennds directory and also supplied in the source code. This should be copied the the web root of a suitable web server for use.

If set to "3" The FAS is enforced by openNDS to use **https** protocol. Level 3 is the same as level 2 except the use of https protocol is enforced for FAS. In addition, the "authmon" daemon is loaded. This allows the external FAS, after client verification, to effectively traverse inbound firewalls and address translation to achieve NDS authentication without generating browser security warnings or errors. An example FAS level 3 php script (fas-aes-https.php) is pre-installed in the /etc/opennds directory and also supplied in the source code. This should be copied the the web root of a suitable web server for use.

Option faskey has a default value. It is recommended that this is set to some secret value in the config file and the FAS script set to use a matching value, ie faskey must be pre-shared with FAS.

8.4 Example FAS Query strings

Level 0 (fas_secure_enabled = 0)

NDS sends the token and other information to FAS as clear text.

http://fasremoteip:fasport/faspath?authaction=http://gatewayaddress:gatewayport/opennds_auth/?clientip=[clientip]&gate

Note: a knowledgeable user could bypass FAS, so running fas_secure_enabled at level 1, 2 or 3 is recommended.

Level 1 (fas_secure_enabled = 1)

NDS sends a query string containing a base64 encoded string containing required parameters and variables, plus any FAS variables generated in the client dialogue, such as username and email address. The client token is never exposed.

Example Level 1 query string:

http://fasremotefqdn:fasport/faspath?fas=[b64encodedstring]&username=[client_username]&emailaddr=[client_ema

The b64encoded string will contain a comma space separated list of parameters.

The decoded string received by FAS will be of the form:

[varname1]=[var1], [varname2]=[var2], ….. etc (the separator being comma-space).

For example:

clientip=[clientipaddress], clientmac=[clientmacaddress], gatewayname=[gatewayname], client token, gatewayaddress, authdir, originurl

The FAS must return the hash of the concatenated hid value and the value of faskey identified in the query string as "tok". NDS will automatically detect this.

Levels 2 and 3 (fas_secure_enabled = 2 and fas_secure_enabled = 3), openNDS sends encrypted information to FAS.

http://fasremotefqdn:fasport/faspath?fas=[aes-256-cbc data]&iv=[random initialisation vector] (level 2)

https://fasremotefqdn:fasport/faspath?fas=[aes-256-cbc data]&iv=[random initialisation vector] (level 3)

It is the responsibility of FAS to decrypt the aes-256-cbc data it receives, using the pre shared faskey and the random initialisation vector.

The decrypted string received by FAS will be of the form: [varname1]=[var1], [varname2]=[var2], ….. etc. (the separator being comma-space).

For example:

clientip=[clientipaddress], clientmac=[clientmacaddress], gatewayname=[gatewayname], client token, gatewayaddress, authdir, originurl

It is the responsibility of FAS to parse the decrypted string for the variables it requires.

8.4.1 Example scripts

Full details of how to use FAS query strings can be seen in the example scripts, login.sh, fas-hid.php, fas-aes.php and fas-aes-https.php

8.5 Custom Parameters

Custom Parameters are primarily intended to be used by remote configuration tools and are generally useful for passing static information to a remote FAS.

A list of Custom Parameters can be defined in the configuration file. Once a custom parameter is defined in the configuration, its value will be fixed.

Parameters must be of the form param_name=param_value and may not contain white space or single quote characters.

Custom parameters are added to the base64 encoded query string when FAS level 1 is set or the basic login option is used. Note the basic login option is a special case of FAS level 1 running the login.sh script.

Custom parameters are added to the encrypted query string when FAS levels 1, 2 and 3 are set.

The fas_custom_parameters_list option in the configuration file is used to set custom parameters. This is detailed in the default configuration file.

It is the responsibility of FAS to parse the query string for the custom parameters it requires.

8.6 Network Zones - Determining the Interface the Client is Connected To

The Network coverage of a Captive Portal can take many forms, from a single SSID through to an extensive mesh network.

Using FAS, it is quite simple to dynamically adapt the Client Login page depending on the Network Zone a client is connected to. NDS can determine the local interface or 802.11s mesh network node a client is using. A simple lookup table can then be included in a custom FAS, relating interfaces or mesh nodes to sensibly named coverage zones.

A very simple example would be a captive portal set up with a wireless network for “Staff”, another for “Guests” and office machines connected via ethernet.

- Ethernet connected office machines would gain access by simply clicking “Continue”.
- Staff mobiles connect to the Staff WiFi using a standard access code then clicking “Continue”.
- Guests connect to the open Guest Wifi and are required to enter details such as Name, email address etc.

NDS is aware of the interface or mesh node a client is using.

For a FAS using *fas_secure_enabled = 2*, an additional variable, clientif, is sent to the FAS in the encrypted query string (local or remote FAS).

For all other levels of fas_secure_enabled, PreAuth and BinAuth, the library utility “get_client_interface” is required to be used by the relevant script (local FAS only).

Working examples can be found in the included scripts:

- fas-aes.php
- login.sh
- demo-preauth.sh
- demo-preauth-remote-image.sh

For details of the clientif variable and how to use get_client_interface, see the section **Library Utilities**.

8.7 After Successful Verification by FAS

If the client is successfully verified by the FAS, FAS will send the return hash id (rhid) to openNDS to finally allow the client access to the Internet.

8.8 Post FAS processing

Once the client has been authenticated by the FAS, NDS must then be informed to allow the client to have access to the Internet.

The method of achieving this depends upon the fas_secure_enabled level.

8.8.1 Authentication Method for fas_secure_enabled levels 0,1 and 2

Once FAS has verified the client credentials, authentication is achieved by accessing NDS at a special virtual URL.

This virtual URL is of the form:

http://[nds_ip]:[nds_port]/[authdir]/?tok=[token]&redir=[landing_page_url]

This is most commonly achieved using an html form of method GET. The parameter redir can be the client's originally requested URL sent by NDS, or more usefully, the URL of a suitable landing page.

The "login_option" special case

The default "login_option" script, login.sh, must always be a local script so has access to ndsctl auth method of authentication without needing the authmon daemon so uses this rather than use the authdir GET method detailed above. This means login.sh can directly set client quotas without requiring BinAuth.

8.8.2 Authentication Method for fas_secure_enabled level 3 (Authmon Daemon)

When fas_secure_enabled level 3 is used (https protocol), post verification authentication is achieved by the openNDS Authmon daemon.

Authmon is started by openNDS to facilitate NAT traversal and allow a remote https FAS to communicate with the local openNDS.

FAS will deposit client authentication variables for the Authmon daemon to use for the authentication process. These variables are as follows:

- clientip: The ip address of the client to be authenticated
- sessionlength: length of session - minutes
- uploadrate: maximum allowed upload data rate - kbits/sec
- downloadrate: maximum allowed download data rate - kbits/sec
- uploadquota: allowed upload data quota - kBytes
- downloadquota: allowed download data quota - kBytes
- custom: A custom data string that will be sent to BinAuth

Details can be found in the example script fas-aes-https.php

Be aware that many client CPD processes will **automatically close** the landing page as soon as Internet access is detected.

8.9 BinAuth Post FAS Processing

As BinAuth can be enabled at the same time as FAS, a BinAuth script may be used for custom post FAS processing. (see BinAuth).

The example BinAuth script, binauth_log.sh, is designed to locally log details of each client authentication and receives client data including the token, ipaddress and macaddress. In addition it receives the custom data string sent from FAS.

8.10 Manual Access of NDS Virtual URL

If the user of an already authenticated client device manually accesses the NDS Virtual URL, they will be redirected back to FAS with the "status" query string.

This will be of the form:

http://fasremoteip:fasport/faspath?clientip=[clientip]&gatewayname=[gatewayname]&status=authenticated

FAS should then serve a suitable error page informing the client user that they are already logged in.

8.11 Running FAS on your openNDS router

FAS has been tested using uhttpd, lighttpd, ngnix, apache and libmicrohttpd.

Running on OpenWrt with uhttpd/PHP:

A FAS service may run quite well on uhttpd (the web server that serves Luci) on an OpenWrt supported device with 8MB flash and 32MB ram but shortage of ram will be an issue if more than two or three clients log in at the same time.

For this reason a device with a **minimum** of 8MB flash and 64MB ram is recommended.

A device with 16MB flash or greater and 128MB ram or greater is recommended as a target for serious development.

Although port 80 is the default for uhttpd, it is reserved for Captive Portal Detection so cannot be used for FAS. uhttpd can however be configured to operate on more than one port.

We will use port 2080 in this example.

Install the module php7-cgi. Further modules may be required depending on your requirements.

To enable FAS with php in uhttpd you must add the lines:

```
list listen_http 0.0.0.0:2080
list interpreter ".php=/usr/bin/php-cgi"
```

to the /etc/config/uhttpd file in the config uhttpd 'main' or first section.

The two important NDS options to set will be:

1. fasport. We will use port 2080 for uhttpd
2. faspath. Set to, for example, /myfas/fas.php, your FAS files being placed in /www/myfas/

8.12 Using a Shared Hosting Server for a Remote FAS

A typical Internet hosted **shared** server will be set up to serve multiple domain names.

To access yours, it is important to configure the two options:

fasremoteip = the **ip address** of the remote server

AND

fasremotefqdn = the **Fully Qualified Domain name** of the remote server

8.13 Using a CDN (Content Delivery Network) Hosted Server for a Remote FAS

This is essentially the same as using a Shared Hosting Server with the additional requirement of *also* adding fasremotefqdn to the Walled Garden configuration.

The setting for fasremoteip should be one of the valid IPv4 addresses of your CDN service.

8.14 Using the FAS Example Scripts (fas-hid, fas-aes.php and fas-aes-https.php)

These three, fully functional, example FAS scripts are included in the package install and can be found in the /etc/opennds folder. To function, they need to be copied to the web root or a folder in the web root of your FAS http/php server.

8.14.1 fas-hid.php

You can run the FAS example script, fas-hid.php, locally on the same device that is running NDS, or remotely on an Internet based FAS server.

The use of http protocol is enforced. fas-hid is specifically targeted at local systems with insufficient resources to run PHP services, yet facilitate remote FAS support without exposing the client token or requiring the remote FAS to somehow access the local ndsctl.

If run locally on the NDS device, a minimum of 64MB of ram may be sufficient, but 128MB or more is recommended.

If run on a remote FAS server, a minimum of 32MB of ram on the local device may be sufficient, but 64MB or more is recommended.

8.14.2 fas-aes.php

You can run the FAS example script, fas-aes.php, locally on the same device that is running NDS (A minimum of 64MB of ram may be sufficient, but 128MB is recommended), or remotely on an Internet based FAS server. The use of http protocol is enforced.

8.14.3 fas-aes-https.php

You can run the FAS example script, fas-aes-https.php, remotely on an Internet based https FAS server. The use of https protocol is enforced.

On the openNDS device, a minimum of 64MB of ram may be sufficient, but 128MB is recommended.

8.14.4 Example Script File fas-aes.php

Http protocol is enforced.

Assuming you have installed your web server of choice, configured it for port 2080 and added PHP support using the package php7-cgi, you can do the following.

(Under other operating systems you may need to edit the opennds.conf file in /etc/opennds instead, but the process is very similar.)

- Install the packages php7-cli and php7-mod-openssl
- Create a folder for the FAS script eg: /[server-web-root]/nds/ on the Internet FAS server
- Place the file fas-aes.php in /[server-web-root]/nds/

  (You can find it in the /etc/opennds directory.)
- Edit the file /etc/config/opennds

  adding the lines:

  ```
  option fasport '2080'
  option faspath '/nds/fas-aes.php'
  option fas_secure_enabled '2'
  option faskey '1234567890'
  ```
- Restart NDS using the command `service opennds restart`

8.14.5 Example Script File fas-aes-https.php

Https protocol is enforced.

Assuming you have access to an Internet based https web server you can do the following.

(Under other operating systems you may need to edit the opennds.conf file in /etc/opennds instead, but the process is very similar.)

- Install the packages php7-cli and php7-mod-openssl on your NDS router
- Create a folder for the FAS script eg: /[server-web-root]/nds/ on the Internet FAS server
- Place the file fas-aes.php in /[server-web-root]/nds/

  (You can find it in the /etc/opennds directory.)
- Edit the file /etc/config/opennds

  adding the lines:

  `option fasport '443'` (or the actual port in use if different)

  ```
  option faspath '/nds/fas-aes-https.php'
  option fas_secure_enabled '3'
  ```

option faskey '1234567890'

option fasremoteip '46.32.240.41' (change this to the actual ip address of the remote server)

option fasremotefqdn 'blue-wave.net' (change this to the actual FQDN of the remote server)

- Restart NDS using the command `service opennds restart`

8.14.6 Example Script File fas-hid.php

Http protocol is enforced.

fas-hid.php can be configured to be run locally or remotely in the same basic way as fas-aes.

However it is targeted for use on devices with limited resources as it does not require openNDS to have locally installed php-cli modules.

It uses fas_secure_enabled level 1, but sends a digest of the client token to the remote FAS. The digest is created using faskey, so the client token is not exposed.

The fas-hid.php script then uses this digest along with the pre shared faskey to request authentication by openNDS, thus mitigating any requirement for remotely accessing ndsctl that otherwise would be required.

Assuming you have access to an Internet based http web server you can do the following.

(Under other operating systems you may need to edit the opennds.conf file in /etc/opennds instead, but the process is very similar.)

- Create a folder for the FAS script eg: /[server-web-root]/nds/ on the Internet FAS server
- Place the file fas-hid.php in /[server-web-root]/nds/

  (You can find it in the /etc/opennds directory.)
- Edit the file /etc/config/opennds

  adding the lines:

  option fasport '80' (or the actual port in use if different)

  option faspath '/nds/fas-hid.php'

  option fas_secure_enabled '1'

  option faskey '1234567890'

  option fasremoteip '46.32.240.41' (change this to the actual ip address of the remote server)

  option fasremotefqdn 'blue-wave.net' (change this to the actual FQDN of the remote server)
- Restart NDS using the command `service opennds restart`

8.15 Changing faskey

The value of option faskey should of course be changed, but must also be pre-shared with FAS by editing the example or your own script to match the new value.

CHAPTER 9

ThemeSpec Script Files

9.1 Overview

ThemeSpec Script files are an evolution of the original PreAuth/Login-Option scripts that were introduced to provide dynamic splash page sequences without requiring a separate web server or additional dependencies.

From openNDS version 9 onwards, the old login.sh script file is replaced by a new library script that provides resources for openNDS to use along with a standardised method for including themed specification files that generate the required html for the specified theme and functionality required for the venue.

9.2 Pre-installed ThemeSpec script files

Two default and pre-installed ThemeSpec script files are included. These provide the equivalent of the pre version 9 "Click to Continue" and "Username/Email" Login splash pages.

These options are selected in the same way as was the case in version 8, using:

`option login_option_enabled '1'` (for click to continue)

and

`option login_option_enabled '2'` (for username/email login)

Mode 1 selects the script file theme_click-to-continue.sh for inclusion.

Mode 2 selects the script file theme_user-email-login.sh for inclusion.

9.3 libopennds.sh

This utility controls many of the functions required for PreAuth/ThemeSpec scripts.

Warning: This file will not normally be modified for customisation of the splash page sequence. *Do not edit unless you know what you are doing!*

Customisation should be carried out in a related Themespec file.

Usage: libopennds arg1 arg2 . . . argN

arg1: "clean", removes custom files, images and client data

returns: tmpfsmountpoint (the mountpoint of the tmpfs volatile storage of the router.

arg1: "tmpfs", finds the tmpfs mountpoint

returns: tmpfsmountpoint (the mountpoint of the tmpfs volatile storage of the router.

arg1: "mhdcheck", checks if MHD is running (used by MHD watchdog) *returns*: "1" if MHD is running, "2" if MHD is not running

arg1: "?fas=<b64string>", generates ThemeSpec html using b64encoded data sent from openNDS

arg2: urlencoded_useragent_string

arg3: mode (1, 2 or 3) (this is the mode specified in option login_option in the config file.

arg4: themespecpath (if mode = 3. Will be supplied by the call from openNDS)

returns: html for the specified ThemeSpec.

9.4 ThemeSpec Templates

Two additional example Themespec script files are provided. These make full use of custom parameters, custom variables, custom images and custom files to define custom placeholders.

These are:

1. theme_click-to-continue-custom-placeholders.sh
2. theme_user-email-login-custom-placeholders.sh

The first gives a click to continue login, but with the custom placeholders in place.

The second gives the user email login with custom placeholders.

These two are very similar with a login form added to the second.

The second file will be used here as an example template.

9.5 Template: theme_user-email-login-custom-placeholders.sh

The default openNDS config file contains the custom options ready to uncomment.

These options are as follows:

```
list fas_custom_parameters_list 'logo_message=openNDS:%20Perfect%20on%20OpenWrt!
'

list fas_custom_parameters_list 'banner1_message=BlueWave%20-%20Wireless%20Network%20Specia

list fas_custom_parameters_list 'banner2_message=HMS%20Pickle'

list fas_custom_parameters_list 'banner3_message=SeaWolf%20Cruiser%20Racer'

list fas_custom_variables_list 'input=phone:Phone%20Number:text;
postcode:Home%20Post%20Code:text'
```

```
list fas_custom_images_list 'logo_png=https://openwrt.org/_media/logo.png'

list fas_custom_images_list 'banner1_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerbw.jpg'

list fas_custom_images_list 'banner2_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerpickle.jpg'

list fas_custom_images_list 'banner3_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerseawolf.jpg'

list fas_custom_files_list 'advert1_htm=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerpickle.htm'
```

As can be seen, these are of the form: `placeholder=value`

9.5.1 Custom Parameters

In this case the custom parameters each define a short text string. These strings will be used as titles for custom images.

9.5.2 Custom Images

The custom images define remote image files with the URL for download. The images will be downloaded and saved in the tmpfs volatile storage of the router. The themespec script will add the images and the custom parameters (as titles) to the output html served to the client.

9.5.3 Custom Files

A single custom file is defined. This is a remote html file that is downloaded in the same way as custom images are. This downloaded file is included in the html at the relevant placeholder location in the html served to the client.

9.5.4 Custom Variables

A single custom variable is defined. Instead of a single placeholder, in this case, the variable definition has the keyword "input=".

The value of this variable is used by the themespec script to inject a list of html form input fields, the location in the output html determined by placeholders.

In this case the custom variable value is:

```
phone:Phone%20Number:text;postcode:Home%20Post%20Code:text
```

This is a list of semi-colon separated fields.

Each field is a colon separated field specification in the form of name:title:type.

In this example we have two input fields:

- name=phone, title=Phone%20Number, and input type=text
- name=postcode, title=Home%20Post%20Code

The resulting html served to the client will have two additional input fields on the login page ie. phone number and post code.

Note: Spaces must be url encoded ie replaced with %20, to prevent parsing issues.

9.5.5 Serving the Splash Page Sequence

When a client connects, openNDS calls the libopennds.sh library script passing a request for client verification along with information about the client device. This information is b64encoded into a single argument.

This argument is identified by the initial character string "?fas="

The libopennds library then decodes the string and parses for data required for verification and logging.

The libopennds library then calls the themespec file configured in the openNDS config.

For this example theme_user-email-login-custom-placeholders is called:

- The themespec script sets Quotas and Data Rates that may be required for this theme, overriding global values. These new values, if set, can be set again later in this script on a client by client basis if required. In this case we will set them to "0" (zero), meaning the global values will take effect.
- The themespec script then configures itself for any custom requirements such as parameters, images, files and form inputs.
- Control is then passed back to libopennds
- libopennds then calls download_image_files() if required by themespec. Files are not downloaded if already present.
- libopennds then calls download_data_files() if required by themespec. Files are not downloaded if already present.
- libopennds then sends the html page header to openNDS to be served to the client.
- libopennds checks if "Terms of Service" has been clicked and if it has, calls display_terms().
- libopennds checks if the landing page has been requested and if it has, calls landing_page().
- libopennds calls generate_splash_sequence() in the themespec script.
- themespec checks if this is the initial redirect of the client. If is is, the first page of the splash page sequence is then served ie the "login page".
- themespec serves the second page of the splash sequence (thankyou page) once the login page is completed by the client.
- themespec returns to libopennds with a request for authentication once the "thankyou page" is accepted by the client.
- libopennds calls landing_page() - the landing page defined in themespec is served to the client.
- libopennds finally calls openNDS to authenticate the client, passing on any quotas specific to the theme or client.

CHAPTER 10

PreAuth Option

10.1 Overview

PreAuth is an implementation of FAS *without the resource utilisation of a separate web server*, particularly useful for legacy devices with limited flash and RAM capacity.

PreAuth is a pre-authentication process that enables NDS to directly serve dynamic web content generated by a script or executable program, using its own built in Web server.

Note: The Pre-Installed Basic Splash Pages are pre-installed PreAuth scripts with special configuration options to enable them.

A custom PreAuth script can be enabled by configuring openNDS FAS to point to a virtual URL in the openNDS webserver root instead of an independent FAS server. The location of the PreAuth script or program is configured in the config file.

The PreAuth script can be a shell script or any other script type that an interpreter is available for (for example, PHP-cli, Python etc.).

It can even be a compiled executable binary program if desired, for example, a compiled program written in C or any other language that has a compiler available for the platform.

The PreAuth script or program will parse the url encoded command line (query string) passed to it and output html depending on the contents of the query string it receives from openNDS. In turn, openNDS will serve this html to the client device that is attempting to access the Internet.

10.2 Configuring a Custom PreAuth

A custom PreAuth is set up using the standard NDS configuration for FAS (See the **Forwarding Authentication Service (FAS)** section of this documentation).

In addition a single PreAuth configuration option is required to inform NDS of the location of the PreAuth script or program.

In summary, the following configuration options should be set:

1. **fasport**. This enables FAS and *must* be set to the same value as the gateway port.
2. **faspath**. This *must* be set to the PreAuth virtual url, "/opennds_preauth/" by default.
3. **preauth**. This the path to the PreAuth script.

The remaining FAS configuration options must be left unset at the default values.

ie:

1. **fasremoteip**. Not set (defaults to the gateway ip address).
2. **fasremotefqdn**. Not set.
3. **fas_secure_enable**. Not set (defaults to enabled).

10.3 What Does the Default PreAuth Login Script Do?

It generates html output for openNDS to serve as a dynamic series of splash pages. The html it outputs can dynamically change according to the inputs received from a client in the html forms it generates.

10.4 Writing A PreAuth Script

A Preauth script can be written as a shell script or any other language that the system has an interpreter for. It could also be a complied program.

openNDS calls the PreAuth script with a b64 encoded argument containing the equivalent of an html query string but with ", " (comma space) in place of "&" (ampersand).

Full details of programming a custom PreAuth script can be found by reading and following the logig flow in the libopennds script and accompanying ThemeSpec files.

10.4.1 Custom Parameters, Variables, Images and Files

Custom Parameters, Variables, Images and Files, defined in the config and the definitions are passed to PreAuth in the b64 encoded query string.

CHAPTER 11

BinAuth Option

11.1 Overview

BinAuth provides a method of running a post authentication script or extension program. BinAuth is ALWAYS local to NDS and as such will have access to all the resources of the local system.

BinAuth is available when FAS is used at all levels of fas_secure_enabled (0, 1, 2 and 3).

A custom variable is forwarded to BinAuth This can contain an embedded payload of custom data defined by the FAS. As FAS is typically remote from the NDS router, this provides a link to the local system.

BinAuth has the means to override session timeout, data rate and data volume quotas on a client by client basis.

BinAuth is called by openNDS at the following times:

- After the client CPD browser makes an authentication request to openNDS
- After the client device is granted Internet access by openNDS
- After the client is deauthenticated by request
- After the client idle timeout interval has expired
- After the client session timeout interval has expired
- After a data upload or download quota has been exceeded
- After the client is authenticated by ndsctl command
- After the client is deauthenticated by ndsctl command
- After NDS has received a shutdown command

11.2 BinAuth Command Line Arguments

When OpenNDS calls the configured BinAuth script, it sends a set of command line arguments depending on the reason for the call.

11.2.1 BinAuth Command Methods

The first argument, arg[1], is always the "method".

The method will be set to one of the following values:

- "**auth_client**" This is a request for authentication by the client.
- "**client_auth**" This is an acknowledgement of successful authentication by NDS.
- "**client_deauth**" This is an acknowledgement that the client has been deauthenticated by NDS.
- "**idle_deauth**" - NDS has deauthenticated the client because the idle timeout duration has been exceeded.
- "**timeout_deauth**" - NDS has deauthenticated the client because the session length duration has been exceeded.
- "**downquota_deauth**" - NDS has deauthenticated the client because the client's download quota has been exceeded
- "**upquota_deauth**" - NDS has deauthenticated the client because the client's upload quota has been exceeded
- "**ndsctl_auth**" - NDS has authorised the client because of an ndsctl command (for example, sent by the NDS AuthMon daemon).
- "**ndsctl_deauth**" - NDS has deauthenticated the client because of an ndsctl command.
- "**shutdown_deauth**" - NDS has deauthenticated the client because it received a shutdown command.

Additional arguments depend on the method type:

11.2.2 Method auth_client

The first argument is auth_client and the following arguments are set to:

- arg[2] = client_mac
- arg[3] = username (deprecated)
- arg[4] = password (deprecated)
- arg[5] = url-escaped redir variable (the URL originally requested by the client.
- arg[6] = url-escaped client user agent string
- arg[7] = client_ip
- arg[8] = client_token
- arg[9] = url-escaped custom variable string

11.2.3 Method ndsctl_auth

The first argument is ndsctl_auth and the following arguments are set to:

- arg[2] = client_mac
- arg[3] = bytes_incoming (set to 0, reserved for future use)
- arg[4] = bytes_outgoing (set to 0, reserved for future use)
- arg[5] = session_start - the session start time
- arg[6] = session_end - the session end time

- arg[7] = client_token
- arg[8] = url-escaped custom variable string

11.2.4 All Other Methods

When the first argument is other than auth_client or ndsctl_auth, the following arguments are set to:

- arg[2] = client_mac
- arg[3] = bytes_incoming (total incoming bytes for client)
- arg[4] = bytes_outgoing (total incoming bytes for client)
- arg[5] = session_start - the session start time
- arg[6] = session_end - the session end time
- arg[7] = client_token

11.2.5 Using the Custom Variable string

Method auth_client - arg[9] and ndsctl_auth - arg[8], contain the url-escaped custom variable string. openNDS extracts this variable from the query string of the http auth_client call from a FAS or PreAuth page.

It is provided for general unspecified use and is url-escaped. A typical example of its use is for a level 0, 1, or 2 FAS to communicate special values for individual clients, or groups of clients.

11.3 Example BinAuth Scripts

An example BinAuth script is pre-installed. It provides a local NDS Access Log

FAS is often remote from the NDS router and this script provides a simple method of interacting directly with the local openNDS. FAS can send custom data to BinAuth as a payload in the custom variable string that is relayed to BinAuth.

The log file is stored by default in the operating system tmpfs volatile storage area to prevent flash storage wear.

On OpenWrt this location is /tmp/ndslog/

and on Debian and OpenSuse it is /run/ndslog/

The location is automatically detected on most operating systems.

Free space checking is done and if the log file becomes too large, logging ceases and an error is sent to syslog.

Log files do not persist through a reboot so if required the location can be changed to, for example, a USB stick.

The binauth_log example is pre-installed.

This script has a single component, the shell script:

binauth_log.sh

The file binauth_log.sh is preinstalled in the /usr/lib/opennds directory.

This is enabled by setting the BinAuth option in the config file (/etc/config/opennds on Openwrt, or /etc/opennds/opennds.conf on other operating systems.

This script is then activated with the command:

```
service opennds restart
```

CHAPTER 12

Library Utilities

12.1 Overview

A number of library utilities are included. These may be used by NDS itself, FAS and Preauth. These may in the future, be enhanced, have additional functionality added.

By default, library utilities will be installed in the folder

`/usr/lib/opennds/`

12.2 List of Library Utilities

12.2.1 get_client_token.sh

This utility allows the unique token of a client to be determined from the client ip address.

It can be used in PreAuth and local FAS scripts.

Usage: get_client_token.sh [clientip]

Returns: [client token]

Where: [client token] is the unique client token string.

12.2.2 get_client_interface.sh

This utility allows the interface a client is using to be determined from the client mac address.

It can be used in PreAuth and local FAS scripts.

It is used by NDS when fas secure levels 1 and 2 are set along with faskey also being set.

Its output is sent to FAS in the encrypted query string as the variable “clientif”

Usage: get_client_interface.sh [clientmac]

Returns: [local_interface] [meshnode_mac] [local_mesh_interface]

Where:

[local_interface] is the local interface the client is using.

[meshnode_mac] is the mac address of the 802.11s meshnode the client is using (null if mesh not present).

[local_mesh_interface] is the local 802.11s interface the client is using (null if mesh not present).

12.2.3 unescape.sh

This utility allows an input string to be unescaped. It currently only supports url-decoding.

It can be used by NDS as the unescape callback for libmicrohttpd.

To enable, set the unescape_callback_enabled option to "1"

To disable, set the unescape_callback_enabled option to "0"

The default is disabled (use internal MHD unescape)

eg In the OpenWrt configuration file

```
option unescape_callback_enabled '0'
```

Usage: unescape.sh [-option] [escapedstring]

Returns: [unescapedstring]

Where:

[-option] is unescape type, currently -url only

12.2.4 libopennds.sh

This utility controls many of the functions required for PreAuth/ThemeSpec scripts.

Usage: libopennds arg1 arg2 . . . argN

arg1: "clean", removes custom files, images and client data

returns: tmpfsmountpoint (the mountpoint of the tmpfs volatile storage of the router.

arg1: "tmpfs", finds the tmpfs mountpoint

returns: tmpfsmountpoint (the mountpoint of the tmpfs volatile storage of the router.

arg1: "mhdcheck", checks if MHD is running (used by MHD watchdog) *returns*: "1" if MHD is running, "2" if MHD is not running

arg1: "?fas=<b64string>", generates ThemeSpec html using b64encoded data sent from openNDS

arg2: urlencoded_useragent_string

arg3: mode (1, 2 or 3) (this is the mode specified in option login_option in the config file.

arg4: themespecpath (if mode = 3)

returns: html for the specified ThemeSpec.

CHAPTER 13

Data Quotas and Traffic Shaping

13.1 Data volume and Rate Quotas

openNDS (NDS) has built in *Data Volume* and *Data Rate* quota support.

Data volume and data rate quotas can be set globally in the config file.

The global values can be overridden on a client by client basis as required.

13.1.1 Global Data Volume Quota

If a client exceeds the global data volume quota, or the individual client quota, that client will be forced out by deauthentication. To continue, the client must re-authenticate.

Configuring Data Volume Quotas

Global volume quotas are configured in the config file.

Example UCI configuration options:

```
# If the client data quota exceeds the value set here, the client will be forced out
# Values are in kB
# If set to 0, there is no limit
# Integer values only
#
option uploadquota '0'
option downloadquota '0'
```

Note: upload means to the Internet, download means from the Internet

Quotas for individual clients will override configured global values and are set either by BinAuth or the Authmon Daemon (fas_secure_enable level 3) - see example BinAuth script (binauth_log.sh) and example FAS script (fas-aes-https.php).

13.1.2 Global Data Rate Quota

The client data rate is calculated using a moving average. This allows clients to burst at maximum possible rate, only rate limiting if the moving average exceeds the specified upload or download rate.

The moving average window size is equal to ratecheckwindow times checkinterval (seconds).

The default value of ratecheckwindow is 2 and is a good working setting. Increasing the value allows a longer period of bursting. Setting to 0 disables all Data Rate Quotas.

Simply put, this means clients are able to transfer data at the maximum rate the router can achieve for short bursts.

Data Rate Quotas are ideal for allowing opening of web pages and emails etc in the fastest way possible, yet preventing an individual client from monopolizing all the available bandwidth by streaming or transferring large files.

Configuring Data Rate Quotas

Global Data Rate quotas are configured in the config file.

```
# Note: upload means to the Internet, download means from the Internet
# Defaults 0
# Integer values only
#
# If the client average data rate exceeds the value set here, the client will be rate␣
↪limited
# Values are in kb/s
# If set to 0, there is no limit
#
# Quotas and rates can also be set by FAS via Authmon Daemon, by BinAuth, and by␣
↪ndsctl auth.
# Values set by these methods, will be override values set in this config file.
#
option uploadrate '0'
option downloadrate '0'
################################################################################
↪#####
# The client data rate is calculated using a moving average.
# This allows clients to burst at maximum possible rate, only rate limiting if the␣
↪moving average
# exceeds the specified upload or download rate.
# The moving average window size is equal to ratecheckwindow times checkinterval␣
↪(seconds)
# Default 2
option ratecheckwindow '2'
```

Note: upload means to the Internet, download means from the Internet

13.1.3 Data Rate Quotas for Individual Clients

Data Rate Quotas for individual clients will override configured global values and are set by ThemeSpec, BinAuth or the Authmon Daemon.

FAS Level 3

FAS level 3 uses the Authmon daemon to set quota values determined by the FAS. The example script fas-aes-https.php shows how to implement this.

FAS level 0, 1 and 2

These levels can either use BinAuth to set quota values determined by the FAS, or if local to the router can use ndsctl authentication with quota values passed as arguments.

If using BinAuth, the FAS would utilise the BinAuth custom variable to send quota values to a BinAuth script configured to interpret the data passed to it in the variable. There is no set method for doing this, it is left to individual installers to develop their own method.

13.2 Traffic Shaping

openNDS (NDS) supports Traffic Shaping (Bandwidth Limiting) using the SQM - Smart Queue Management (sqm-scripts) package, available for OpenWrt and generic Linux.

https://github.com/tohojo/sqm-scripts

SQM does efficient bandwidth control, independently for both upload and download, on an IP connection basis. This ideal for enforcing a fair usage policy on a typical Captive Portal implementation.

In addition the Queue management SQM provides, results in significantly improved WiFi performance, particularly on the modern low cost WiFi routers available on the market today.

Finally, SQM controls quality of service (QOS), allowing priority for real time protocols such a VOIP.

Overall, SQM can enhance significantly the experience of clients using your Captive Portal, whilst ensuring a single client is unlikely to dominate the available Internet service at the expense of others.

13.3 Installing SQM

The generic Linux scripts can be downloaded from the link above.

On OpenWrt, SQM can be installed from the LuCi interface or by the following CLI commands on your router:

opkg update

opkg install sqm-scripts

Note: The standard and default SQM installation expects monitoring of the interface connecting to the WAN. What we need is for SQM to monitor the interface NDS is bound to. This of course will be a LAN interface. The default configuration will limit bandwidth from the WAN connection to services on the Internet. Our configuration will limit client bandwidth TO NDS, thus enabling a true fair usage policy.

To prevent confusion it is important to understand that SQM defines "Upload" as traffic "Out" of the interface SQM is monitoring and "Download" as traffic "In" to the SQM interface.

In the default SQM configuration, Upload will mean what is normally accepted, ie traffic to the Internet and Download will mean traffic from the Internet.

In our case however the terms will be reversed!

The default SQM configuration file on OpenWrt is:

```
config queue
    option enabled '0'
    option interface 'eth1'
    option download '85000'
    option upload '10000'
```

(continues on next page)

(continued from previous page)

```
    option qdisc 'fq_codel'
    option script 'simple.qos'
    option qdisc_advanced '0'
    option ingress_ecn 'ECN'
    option egress_ecn 'ECN'
    option qdisc_really_really_advanced '0'
    option itarget 'auto'
    option etarget 'auto'
    option linklayer 'none'
```

For simple rate limiting, we are interested in setting the desired interface and the download/upload rates.

We may also want to optimize for the type of Internet feed and change the qdisc.

A typical Internet feed could range from a high speed fiber optic connection through fast VDSL to a fairly poor ADSL connection and configured rates should be carefully chosen when setting up your Captive Portal.

A typical Captive Portal however will be providing free Internet access to customers and guests at a business or venue, using their mobile devices.

A good compromise for a business or venue might be a download rate from the Internet of ~3000 Kb/s and an upload rate to the Internet of ~1000 Kb/s will be adequate, allowing for example, a client to stream a YouTube video, yet have minimal effect on other clients browsing the Internet or downloading their emails. Obviously the values for upload and download rates for best overall performance depend on many factors and are best determined by trial and error.

If we assume we have NDS bound to interface br-lan and we have a VDSL connection, a good working setup for SQM will be as follows:

- *Rate to* Internet 1000 Kb/s (but note this is from the perspective of the interface SQM is monitoring, so this means DOWNLOAD from the client).
- *Rate from* Internet 3000 Kb/s (also note this is from the perspective of the interface SQM is monitoring, so is means UPLOAD to the client).
- *VDSL* connection (usually an ethernet like connection)
- *NDS* bound to br-lan

We will configure this by issuing the following commands:

Note the reversed "upload" and "download" values.

```
uci set sqm.@queue[0].interface='br-lan'

uci set sqm.@queue[0].download='1000'

uci set sqm.@queue[0].upload='3000'

uci set sqm.@queue[0].linklayer='ethernet'

uci set sqm.@queue[0].overhead='22'

uci set sqm.@queue[0].qdisc='cake'

uci set sqm.@queue[0].script='piece_of_cake.qos'

uci set sqm.@queue[0].enabled='1'

uci commit sqm
```

(continues on next page)

(continued from previous page)

```
service sqm restart
```

Replace the linklayer and overhead values to match your Internet feed.

The following table lists LinkLayer types and Overhead for common feed types:

Connection Type	LinkLayer	Overhead
Fibre/Cable	Ethernet	18
VDSL2	Ethernet	22
Ethernet	Ethernet	38
ADSL/DSL	ATM	44

Some broadband providers use variations on the values shown here, contacting them for details sometimes helps but often the request will be "off script" for a typical helpdesk. These table values should give good results regardless. Trial and error and the use of a good speed tester is often the only way forward. A good speed tester web site is http://dslreports.com/speedtest

Further details about SQM can be found at the following links:

https://openwrt.org/docs/guide-user/network/traffic-shaping/sqm

https://openwrt.org/docs/guide-user/network/traffic-shaping/sqm-details

CHAPTER 14

Walled Garden

14.1 Manual Walled Garden

Preauthenticated clients are, by default, blocked from all access to the Internet.

Access to certain web sites can be allowed. For example, clients will automatically be granted access to an external FAS server for the purpose of authentication.

Access to other web sites may be manually granted so clients can be served content such as news, information, advertising, etc. This is achieved in openNDS by allowing access to the IP address of the web sites as required.

A set of such web sites is referred to as a Walled Garden.

14.2 Autonomous Walled Garden

Granting access by specifying the site IP address works well in the simplest of cases but the administrative overhead can rapidly become very difficult, for example with social media sites that load balance high volumes of traffic over many possible IP addresses.

In addition, the IP address of any web site may change.

Rather than using IP addresses, a much more efficient method of granting access would be by using the Fully Qualified Domain Name (FQDN) of each site and have a background process populate a database of allowed IP addresses.

openNDS supports Autonomous Walled Garden by means of a simple configuration option. This does have additional dependencies but these only apply if the Autonomous Walled Garden is enabled.

14.3 OpenWrt Walled Garden

The additional dependencies are the ipset and dnsmasq-full packages. Install these by running the following commands:

```
opkg update
opkg install ipset
opkg remove dnsmasq
opkg install dnsmasq-full
```

Configure as follows:

```
uci add_list opennds.@opennds[0].walledgarden_fqdn_list='<fqdn1> <fqdn2> <fqdn3> [....
↪..] <fqdnN>'
```

where <fqdn1> to <fqdnN> are the fully qualified domain names of the URLs you want to use to populate the ipset.

eg. For Facebook use facebook.com and fbcdn.net as fqdn1 and fqdn2

In addition you can limit access to the Walled Garden to a list of IP ports:

```
uci add_list opennds.@opennds[0].walledgarden_port_list='<port1> <port2> <port3> [....
↪..] <portN>'
```

Restart openNDS to activate the Walled Garden.

To make these changes permanent (eg survive a reboot), run the command:

```
uci commit opennds
```

14.4 Generic Linux Walled Garden

On most generic Linux platforms the procedure is in principle the same as for OpenWrt.

The ipset and full dnasmasq packages are requirements.

You can check the compile time options of dnsmasq with the following command:

```
dnsmasq --version | grep -m1 'Compile time options:' | cut -d: -f2
```

If the returned string contains “no-ipset” then you will have to upgrade dnsmasq to the full version.

To enable Walled Garden, add the following to the /etc/opennds/opennds.conf file

```
walledgarden_fqdn_list <fqdn1> <fqdn2> <fqdn3> [......] <fqdnN>
```

In addition you can specify a restricted set of ports for access to the walled garden by adding the line:

```
walledgarden_port_list <port1> <port2> <port3> [......] <portN>
```

Restart openNDS to activate the Walled Garden.

CHAPTER 15

Using ndsctl

A openNDS install includes ndsctl, a separate application which provides some control over a running openNDS process by communicating with it over a unix socket. Some command line options:

- To print to stdout some information about your openNDS process:

```
/usr/bin/ndsctl status
```

- To print to stdout the list of clients in human readable format:

```
/usr/bin/ndsctl clients
```

- To print to stdout the list of clients and trusted devices in json format:

```
/usr/bin/ndsctl json
```

- To print to stdout the details of a particular client in json format (This is particularly useful if called from a FAS or Binauth script.):

```
/usr/bin/ndsctl json [mac|ip|token]
```

- To block a MAC address, when the MAC mechanism is block:

```
/usr/bin/ndsctl block MAC
```

- To unblock a MAC address, when the MAC mechanism is block:

```
/usr/bin/ndsctl unblock MAC
```

- To allow a MAC address, when the MAC mechanism is allow:

```
/usr/bin/ndsctl allow MAC
```

- To unallow a MAC address, when the MAC mechanism is allow:

```
/usr/bin/ndsctl unallow MAC
```

- To deauthenticate a currently authenticated user given their IP or MAC address:

```
/usr/bin/ndsctl deauth IP|MAC
```

- To set the verbosity of logged messages to n:

```
/usr/bin/ndsctl debuglevel n
```

- debuglevel 0 : Silent (only LOG_ERR and LOG_EMERG messages will be seen, otherwise there will be no logging.)
- debuglevel 1 : LOG_ERR, LOG_EMERG, LOG_WARNING and LOG_NOTICE (this is the default level).
- debuglevel 2 : debuglevel 1 + LOG_INFO
- debuglevel 3 : debuglevel 2 + LOG_DEBUG

All other levels are undefined and will result in debug level 3 being set.

For more options, run ndsctl -h. (Note that if you want the effect of ndsctl commands to to persist across openNDS restarts, you have to edit the configuration file.)

CHAPTER 16

Customising openNDS

After initial installation, openNDS (NDS) should be working in its most basic mode and client Captive Portal Detection (CPD) should pop up the default Click to Continue page.

Before attempting to customise NDS you should ensure it is working in this basic mode before you start.

NDS reads its configuration file when it starts up but the location of this file varies depending on the operating system.

As NDS is a package that requires hardware configured as an IP router, perhaps the most common installation is using OpenWrt. However NDS can be compiled to run on most Linux distributions, the most common being Debian or one of its popular variants (eg Raspbian).

If NDS is working in the default, post installation mode, then you will have met the NDS dependencies and can now move on to your own customisation.

16.1 Rules for Customised Splash Pages

It should be noted when designing a custom splash page that for security reasons many client device CPD implementations:

- Immediately close the browser when the client has authenticated.
- Prohibit the use of href links.
- Prohibit downloading of external files (including .css and .js, even if they are allowed in NDS firewall settings).
- Prohibit the execution of javascript.

16.2 The Configuration File

In OpenWrt, or operating systems supporting UCI (such as LEDE) the configuration is kept in the file:

```
/etc/config/opennds
```

In other operating systems the configuration is kept in the file:

```
/etc/opennds/opennds.conf
```

Both of these files contain a full list of options and can be edited directly. A restart of NDS is required for any changes to take effect.

In the case of OpenWrt though, once you are confident in your configuration requirements you can use UCI to read and set any of the configuration options using simple commands, making this very convenient if making changes from scripts, such as those you may write to use with Binauth and FAS.

For example, to list the full configuration, at the command line type:

```
uci show opennds
```

To display the Gateway Name, type:

```
uci get opennds.@opennds[0].gatewayname
```

To set the Gateway Name to a new value, type:

```
uci set opennds.@opennds[0].gatewayname='my new gateway'
```

To add a new firewall rule allowing access to another service running on port 8888 on the router, type:

```
uci add_list opennds.@opennds[0].users_to_router='allow
tcp port 8888'
```

Finally you must tell UCI to commit your changes to the configuration file:

```
uci commit opennds
```

16.3 The Legacy Click and Go Splash Page

The legacy Click to Continue html splash page was deprecated and disabled at v8.0.0.

From v 9.0.0 it has been removed entirely.

16.4 Dynamic Splash Pages

16.4.1 Default Dynamic Click to Continue

The pre-installed dynamic click to continue page sequence is enabled by default using the ThemeSpec "theme_click-to-continue". The configuration default is equivalent to setting:

```
option login_option_enabled '1'
```

It generates a Click to Continue page followed by Thankyou and Landing pages.

User clicks on "Continue" are recorded in the log file /[tmpfs_dir]/ndslog/ndslog.log

Where [tmpfs_dir] is the operating system "temporary" tmpfs mount point. This will be /tmp /run or /var and is automatically detected.

Details of how the script works are contained in comments in the script theme_click-to-continue.sh

16.4.2 Pre-Installed dynamic User/email Login page sequence

The pre-installed dynamic login page is enabled by setting option:

```
option login_option_enabled '2'
```

It generates a login page asking for username and email address. User logins are recorded in the log file /[tmpfs_dir]/ndslog/ndslog.log

Where [tmpfs_dir] is the operating system "temporary" tmpfs mount point. This will be /tmp /run or /var and is automatically detected.

Details of how the script works are contained in comments in the script theme_user-email-login.sh

16.4.3 Custom Dynamic ThemeSpec Pages

Custom ThemeSpec page sequences can be added by setting option:

```
option login_option_enabled '3'
```

and option

```
option themespecpath '/path/to/themespec_script'
```

Two additional ThemeSpec files are included as examples:

/usr/lib/opennds/theme_click-to-continue-custom-placeholders.sh

and

/usr/lib/opennds/theme_user-email-login-custom-placeholders.sh

Both these also require custom parameter, variable, image and file lists:

```
list fas_custom_parameters_list 'logo_message=openNDS:%20Perfect%20on%20OpenWrt!
'

list fas_custom_parameters_list 'banner1_message=BlueWave%20-%20Wireless%20Network%20Specia

list fas_custom_parameters_list 'banner2_message=HMS%20Pickle'

list fas_custom_parameters_list 'banner3_message=SeaWolf%20Cruiser%20Racer'

list fas_custom_variables_list 'input=phone:Phone%20Number:text;
postcode:Home%20Post%20Code:text'

list fas_custom_images_list 'logo_png=https://openwrt.org/_media/logo.png'

list fas_custom_images_list 'banner1_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerbw.jpg'

list fas_custom_images_list 'banner2_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerpickle.jpg'

list fas_custom_images_list 'banner3_jpg=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerseawolf.jpg'

list fas_custom_files_list 'advert1_htm=https://raw.githubusercontent.com/
openNDS/openNDS/9.0.0/resources/bannerpickle.htm'
```

Once configured these two example ThemeSpec scripts will download custom image files, a custom html file and inject custom user input forms for phone number and home postcode.

16.4.4 Other Custom Designs

Custom designed dynamically generated ThemeSpec pages are supported using FAS and PreAuth. For details see the FAS and PreAuth chapters.

CHAPTER 17

Frequently Asked Questions

17.1 What's the difference between versions?

NoDogSplash and openNDS are derived from the same code base.

You cannot upgrade from NoDogSplash to openNDS, instead you must first uninstall NodogSplash before installing openNDS.

NoDogSplash is optimised for running on devices with very limited resources and supports only a single *static* templated html splash page.

openNDS supports dynamic html splash page generation (at default, still with minimal resource utilisation) and an API to support the coding of sophisticated Captive Portal solutions.

openNDS v5 This was the first release of openNDS after forking from NoDogsplash. The following enhancements are included:

- **openNDS API (FAS)**

  A forwarding authentication service. FAS supports development of "Credential Verification" running on any dynamic web serving platform, on the same device as openNDS, on another device on the local network, or on an Internet hosted web server.

- **PreAuth**

  An implementation of FAS running on the same device as openNDS and using openNDS's own web server to generate dynamic web pages. Any scripting language or even a compiled application program can be used. This has the advantage of not requiring the resources of a separate web server.

- **BinAuth**

  Enabling an external script to be called for doing post authentication processing such as setting session durations or writing local logs.

- **Enforce HTTPS option**

  This option enables *https* access to a remote, Internet based FAS server, ensuring the client device does not receive any security warnings or errors. Access to the FAS server using **https** protocol is enforced.

- **Data volume and Rate Quotas**

  This option enables built in *Data Volume* and *Data Rate* quota support. Data volume and data rate quotas can be set globally in the config file. The global values can be overridden on a client by client basis as required.

- **Introduction of library scripts**

  Numerous library scripts are introduced to simplify development of applications.

openNDS v6 This is the first version of openNDS to use the updated libmicrohttpd (MHD) API introduced with v0.9.71.

- openNDS **REQUIRES** MHD v0.9.71 or higher.

openNDS v7 This version contains several major enhancements, including:

- **Autonomous Walled Garden Support**

  A simple openNDS configuration option enables Autonomous Walled Garden operation based on a list of target FQDNs

- **Custom Parameter Support**

  A list of static Custom Parameters can be set as a configuration option. Once set, these parameters are fixed and will be sent to remote FAS servers.

  This functionality was added specifically to support remote configuration tools such as Opensync, but can be generally useful for passing local fixed information to a remote FAS.

  It is important that this is NOT confused with the dynamic Custom Variables that can be defined as a part of a FAS/Client dialogue.

- **Legacy Templated splash.html Deprecated and Disabled**

  The legacy splash.html page and its templated variables is deprecated and disabled. This is replaced by an enhanced login option script capable of generating a dynamic html dialogue. The default is a simple click to continue page, similar to the old splash.html.

openNDS v8 This version contains several major enhancements, including:

- **Base 64 encoding of the Query string**

  The introduction of Base 64 encoding of the query string into the FAS API, enables easier customisation of FAS scripts.

- **Global use of Hashed ID (hid)**

  Hashed ID (hid), carried in the base64 encoded query string for levels 1, 2 and 3, including the default click to continue page, provides a uniform level of security against portal bypass. The option "faskey" is now enabled at all times with a simple default value and settable in the config files and FAS scripts.

- **Client User Status Page**

  A client user status page is now available at a configurable Gateway FQDN. This page includes a listing of the client's quota allocation and usage, as well as a "Logout" button. A client that is connected butnot logged in for any reason will be redirected to an RFC6585 Status Code 511 "Network Authentication Required" page with a "Login" button. the default url for a client to access their status page is http://status.client

openNDS v9 This version contains several major enhancements, including:

- **Introduction of ThemeSpec Script Files**

  Option login_option is extended to include Themed page sequences with custom parameters, variables, images and files. A ThemeSpec file has the ability to inject html form fields, text strings, images from external sources and content files also from external sources, all defined in the config file.

- **Enhanced Data Rate Quotas**

  Data Rate Quotas are now enhanced with packet rate limiting being applied when a client average rate rises above its preset threshold. The time window over which the rate is averaged can be tuned using settings in the config file, allowing rate bursting for best user experience while still limiting the rate of larger downloads or uploads that might otherwise impact other clients.

17.2 Can I upgrade from NoDogSplash to openNDS?

No.

- You must first uninstall NoDogSplash before installing openNDS.

17.3 Can I upgrade from v5 to v6

Yes.

- But you must upgrade libmicrohttpd to version v0.9.71 or higher.

17.4 Can I upgrade from v6 to v7?

You can, if:

- You don't use RedirectURL (this has been deprecated for some time as it mostly did not work with client CPD implementations. It has now been removed. A reliable replacement is a FAS Welcome Page.
- You don't use the Templated html splash page (splash.html). Templated splash is now deprecated and disabled. It can be re-enabled by setting the allow_legacy_splash option to allow time for migration. Support will be removed entirely in a later version.

17.5 Can I upgrade from v7 to v8?

You can, if:

- You modify your FAS scripts to use the openNDS v8 API. The FAS query string is now either base64 encoded, or encrypted.
- In addition Hashed ID (hid) is used for authentication, removing the need for a FAS script to somehow obtain the client Token.

17.6 Can I upgrade from v8 to v9

You can, if:

- You modify your FAS scripts to use the openNDS v9 API
- You move to ThemeSpec scripts or FAS **from Legacy Splash**. Legacy Splash Pages are no longer supported. The default ThemeSpec (option login_option 1) is equivalent to the old splash.html click to continue page.

17.7 How can I add custom parameters, such as site specific information?

Custom parameters were introduced in openNDS version 7 and are defined simply in the config file. These parameters are passed to the FAS in the query string. Version 8 embeds any custom parameters in the encoded/encrypted query string, macing it much simpler to parse for them in the FAS script.

17.8 How can I add custom fields on the login page, such as phone number, car licence plate number etc.?

A simple configuration option allows fields to be added automatically to the pages of ThemeSpec login sequences.

17.9 Is it possible to display custom info or advertising on the login pages?

Yes! Simple config options specify the URLs of images and html content. These will be automatically downloaded and injected into the dynamic pages created by suitable Themespec scripts.

17.10 How do I manage client data usage?

openNDS (NDS) has built in *Data Volume* and *Data Rate* quota support.

- Data volume and data rate quotas can be set globally in the config file.
- The global values can be overridden on a client by client basis as required, either by FAS or BinAuth.
- If a client exceeds their volume quota they will be deauthenticated.
- If a client exceeds their rate quota, they will be packet rate limited to ensure their average rate stays below the rate quota value. This allows clients to burst at a higher rate for short intervals, improving performance, but prevents them from hogging bandwidth.

17.11 Can I use Traffic Shaping with openNDS?

SQM Scripts (Smart Queue Management), is fully compatible with openNDS and if configured to operate on the openNDS interface (br-lan by default) will provide efficient IP connection based traffic control to ensure fair usage of available bandwidth.

This can be installed as a package on OpenWrt. For other distributions of Linux it is available at: https://github.com/tohojo/sqm-scripts

17.12 Is an *https splash page* supported?

Yes. FAS Secure Level 3 enforces https protocol for the splash login page on an external FAS server.

17.13 Is *https capture* supported?

No.

- If it was supported, all connections would have a **critical certificate failure**.
- HTTPS web sites are now more or less a standard and to maintain security and user confidence it is essential that captive portals **DO NOT** attempt to capture port 443.
- All modern client devices have the built in, industry standard, *Captive Portal Detection (CPD) service*. This is responsible for triggering the captive portal splash/login page and is **specifically intended to make https capture unnecessary**.

17.14 What is CPD / Captive Portal Detection?

CPD (Captive Portal Detection) has evolved as an enhancement to the network manager component included with major Operating Systems (Linux, Android, iOS/MacOS, Windows).

> Using a pre-defined port 80 web page (the one that gets used depends on the vendor) the network manager will detect the presence of a captive portal hotspot and notify the user. In addition, most major browsers now support CPD.

CHAPTER 18

How to Compile openNDS

18.1 Linux/Unix - Compile in Place on Target Hardware

Make sure the development suite for your Linux distribution is installed.

The libmicrohttpd library (MHD) is a dependency of openNDS so compiling and installing this is a prerequisite.

First, create a working directory and "cd" into it.

Next, Download and un-tar the libmicrohttpd source files.

You can find a version number for MHD at https://ftp.gnu.org/gnu/libmicrohttpd/

The version number for MHD must not exceed 0.9.70 for versions of openNDS less than 6.0.0

```
wget https://ftp.gnu.org/gnu/libmicrohttpd/libmicrohttpd-0.9.71.tar.gz
tar  -xf libmicrohttpd-0.9.71.tar.gz
cd libmicrohttpd-0.9.71
```

where "0.9.71" is the MHD version number we are using in this example.

Now configure and compile:

```
./configure --disable-https
make
sudo rm /usr/local/lib/libmicrohttpd*
sudo make install
sudo rm /etc/ld.so.cache
sudo ldconfig -v
cd ..
```

Then proceed to download the opennds source files.

You can find a release version number for openNDS at https://github.com/openNDS/openNDS/releases

```
wget https://codeload.github.com/opennds/opennds/tar.gz/v7.0.1
tar -xf v7.0.1
cd openNDS-7.0.1
make
sudo make install
sudo systemctl enable opennds
```

Where "7.0.1" is the openNDS version we are using in this example.

openNDS should now start automatically at boot time.

It can be manually started, restarted, stopped or disabled with the following commands:

```
sudo systemctl start opennds

sudo systemctl restart opennds

sudo systemctl stop opennds

sudo systemctl disable opennds
```

The status of openNDS can be checked with the following command:

```
sudo ndsctl status
```

On most Linux distributions you can read the last few entries for openNDS in the system message log with the command:

```
sudo systemctl status opennds
```

If openNDS fails to start, check for error messages with the command:

```
sudo journalctl -e
```

18.2 OpenWrt Package

The OpenWrt package feed supports cross-compiled openNDS packages for all OpenWrt targets. See the "Installing openNDS" section of this documentation. The latest release of openNDS will be found in OpenWrt Snapshots, but will nevertheless be a stable release.

To include openNDS into your OpenWRT image or to create an .ipk package (similar to Debian's .deb files), you can build an OpenWRT image.

You need a Unix console to enter commands into.

Install the dependencies of the build environment (eg on Debian/Ubuntu):

```
sudo apt-get install git subversion g++ libncurses5-dev gawk zlib1g-dev build-
↪essential
```

Build Commands:

```
git clone https://git.openwrt.org/openwrt/openwrt.git
cd openwrt

./scripts/feeds update -a
```

(continues on next page)

(continued from previous page)

```
./scripts/feeds install -a
./scripts/feeds uninstall openndns

git clone git://github.com/openndns/openndns.git
cp -rf openndns/openwrt/openndns package/
rm -rf openndns/

make defconfig
make menuconfig
```

At this point select the appropriate "Target System" and "Target Profile" depending on what target chipset/router you want to build for. Now select the openNDS package in "Network —> Captive Portals".

Now compile/build everything:

```
make
```

The images and all ipk packages are now inside the bin/ folder. You can install the openNDS .ipk using *opkg install <ipkg-file>* on the router or just use the whole image.

For details please check the OpenWRT documentation.

Note for developers

Build Notes

You might want to use your own source location and not the remote repository. To do this you need to checkout the repository yourself and commit your changes locally:

```
git clone git://github.com/openndns/openndns.git
cd openndns
```

... apply your changes

```
git commit -am "my change"
```

Now create a symbolic link in the openNDS package folder using the abolute path:

```
ln -s /my/own/project/folder/openndns/.git openwrt/package/openndns/git-src
```

Also make sure to enable

```
"Advanced configuration options" => "Enable package source tree override"
```

in the menu when you do *make menuconfig*.

CHAPTER 19

Debugging openNDS

19.1 Syslog Logging

openNDS supports four levels of debugging to syslog.

- debuglevel 0 : Silent (only LOG_ERR and LOG_EMERG messages will be seen, otherwise there will be no logging.)
- debuglevel 1 : LOG_ERR, LOG_EMERG, LOG_WARNING and LOG_NOTICE (this is the default level).
- debuglevel 2 : debuglevel 1 + LOG_INFO
- debuglevel 3 : debuglevel 2 + LOG_DEBUG

All other levels are undefined and will result in debug level 3 being set.

To see maximally verbose debugging output from openNDS, set log level to 3.

On OpenWrt, you can use the following commands:

```
uci set opennds.@opennds[0].debuglevel='3'
uci commit opennds
service opennds restart
```

Debug messages are logged to syslog. You can view messages with the logread command.

The default level of logging is 1, and is more appropriate for routine use.

Logging level can also be set using ndsctl.

19.2 Firewall Cleanup

When stopped, openNDS deletes its iptables rules, attempting to leave the router's firewall in its original state. If not (for example, if openNDS crashes instead of exiting cleanly) subsequently starting and stopping openNDS should remove its rules.

On OpenWrt, restarting the firewall will overwrite openNDS's iptables rules, so when the firewall is restarted it will automatically restart openNDS if it is running.

19.3 Packet Marking

openNDS operates by marking packets. Many packages, such as mwan3 and SQM scripts, also mark packets.

By default, openNDS marks its packets in such a way that conflicts are unlikely to occur but the masks used by openNDS can be changed if necessary in the configuration file.

19.4 IPtables Conflicts

Potential conflicts may be investigated by looking at your overall iptables setup. To list all the rules in all the chains, run

```
iptables -L
```

For extensive suggestions on debugging iptables, see for example, Oskar Andreasson's tutorial at:

https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

CHAPTER 20

TODO List

Features should be aimed at providing tools to allow openNDS to be used as flexible Captive Portal engine, rather than building in specific solutions.

Here is a list of things that should be done soon:

- Use uci style config file for all OSes then remove opennds.conf
- Add refresh interval for download of external files in ThemeSpec. This will enable automatic update of informational content, banner advertising etc.
- Consider providing an openNDS-mini package for OpenWrt - for legacy devices with very restricted resources.

Here is a list of possible things TO DO

- Extend Status processing to display a page when a user's authentication is rejected, e.g. because the user exceeded a quota or is blocked etc.
- ip version 6 is not currently supported by NDS. It is not essential or advantageous to have in the short term but should be added at some time in the future.
- Automatic Offline mode/ Built in DNS forwarder. Some thought and discussion has been put into this and it is quite possible to achieve.

CHAPTER 21

Indices and tables

- genindex
- search