Introducing *Open*BSD 's new httpd

Reyk Floeter (reyk@openbsd.org)

March 2015

Abstract

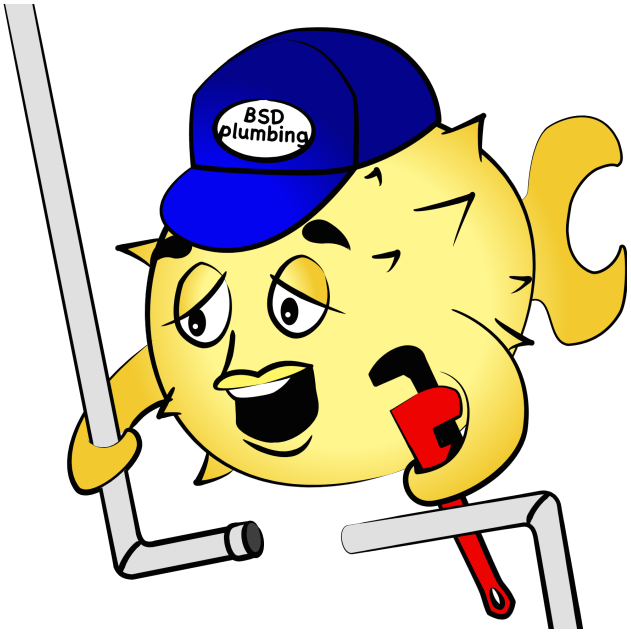
OpenBSD includes a brand new web server that was started just two weeks before the 5.6[11] release was finished. Work is in active progress and significant improvements have been done since its initial appearance. But why do we need another web server? This paper is about the history, design and implementation of the new httpd(8). About 17 years ago, OpenBSD first imported the Apache[8] web server into its base system. It got cleaned up and improved and patched to drop privileges and to chroot itself by default. But years of struggle with the growing codebase, upstream, and the inacceptable disaster of Apache 2 left OpenBSD with an unintended fork of the ageing Apache 1.3.29 for many years. When nginx[10] came up, it promised a much better alternative of a popular, modern web server with a suitable BSD license and a superior design. It was patched to drop privileges and to chroot itself by default and eventually replaced Apache as OpenBSD's default web server. But history repeated itself: a growing codebase, struggle with upstream and the direction of its newly formed commercial entity created a discontent among many developers. Until one day at OpenBSD's g2k14 Hackathon in Slovenia, I experimented with relayd[5] and turned it into a simple web server. A chain of events that were supported by Bob Beck and Theo de Raadt turned it into a serious project that eventually replaced nginx as the new default. It was quickly adopted by many users: "OpenBSD httpd" was born, a simple and secure web server for static files, FastCGI and LibreSSL-powered TLS. And, of course, "httpd is web scale".

1 History

1.1 Introduction

This paper provides a brief description of OpenBSD's new httpd(8) web server, including technical background, history and my personal motivation of creating it. It is inteded to provide some complementary information for my talk at AsiaBSDCon 2015 .

In the beginning Theo de Raadt created OpenBSD and in the early days of the project, the Apache web server got imported. And Apache was not in a good shape, and almost void; and it had many dark parts in the deeps of the source code. And there were some efforts to clean it up, to bring some light; Henning Brauer did it, and it was good. And the source code was divided and much unused code was removed from the base system. And this new web server became known as OpenBSD's Apache, which was different to but based on Apache 1.3. This is how the first web server appeared in OpenBSD.



More than 16 years later, OpenBSD's old web server, based on Apache 1.3.29, was replaced by a totally new server that was written by Reyk Floeter. But first Apache had to be removed; a quest that took a long road. It first appeared in OpenBSD 2.3 of March 1998 and it has been patched and maintained for a long time. The server got modified with custom changes, IPv6 support, and all the security enhancements like chroot and some serious code cleanup. It really followed the tradition of being "a patchy web server". But at some point it ended up to be an inefficient

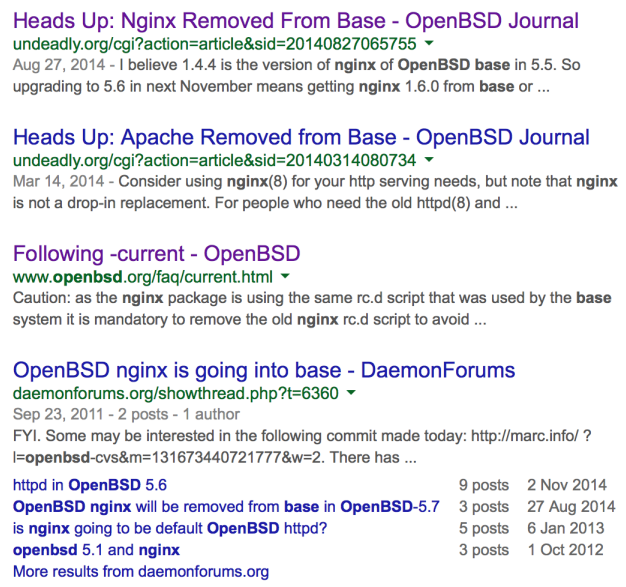
and somewhat ancient design that didn't match the standards of modern network daemons in OpenBSD anymore.

When nginx became more popular, it seemed to be the perfect replacement: a fast, modern, single-threaded, and non-blocking server with an async I/O design, that did FastCGI instead of forking classic CGI processes or loading all kinds of dangerous modules into the server itself. And, after all, it came with a 2-clause BSD-like license. It did not provide all the security improvements, but it got patched to run chroot'ed like OpenBSD's old Apache. nginx was imported into OpenBSD in late 2011 and first appeared in the 5.1 release in May 2012.

For some time, both web servers existed in OpenBSD at same time and it took until March 2014 before Apache got finally removed. nginx became the new default web server in OpenBSD. The delay was primarily caused by nginx *limitation* that it did not support classic CGI. Simple CGI scripts were commonly used by sysadmins on their OpenBSD servers, and even the base system ships with bgplg(8), a CGI-based looking glass for bgpd(8). Apache could not be removed before there was a way to run these scripts with FastCGI under nginx. A FastCGI wrapper in the base system, slowcgi(8), was implemeted by Florian Obser and imported in May 2013 to solve the problem.

nginx' monocracy didn't last very long. A series of events lead to its removal in August 2014. It has been replaced by a new web server: OpenBSD's new httpd by Reyk Floeter; the history and implementation is described in the following chapters.

The latest history of Apache, nginx, and httpd in OpenBSD's base can been summarized by a simple search in Google. The query for "nginx openbsd base" returned the following in the very first results:



1.2 Security Shokunin

The OpenBSD project holds Hackathons at different international locations throughout the year. The largest one is happening once a year, since 1999, and is attended by most active developers. The 2014 g2k14 general Hackathon was taking place July in Ljubljana, Slovenia, and attended by 49 developers. These developer gatherings are used for very active work on OpenBSD's source tree: to start new patches and projects, to finish older ones, and for careful optimizations. The security and code quality of OpenBSD is constantly being improved, a continuous effort to gain perfection that reminds a lot of the Japanese Shokunin.

OpenBSD had just introduced the new reallocarray(3) function with overflow protection in its C library and many developers have started to replace suspicious calls of calloc(3) and malloc(3) in the source tree. There was also the infamous "Heartbleed" bug that lead to OpenBSD's LibreSSL subproject, forked from OpenSSL. One major *exploit mitigation mitigation technique* of OpenSSL that made every system, including OpenBSD, vulnerable to the Heartbleed bug was its obscured memory allocator that silently bypassed OpenBSD's malloc protections by preallocating and reusing large chunks of memory.

Reyk had looked at nginx and wrote a large patch to replace all risky calls of malloc and calloc with array multiplications with reallocarray(3) - or its predecessor mallocarray. It turned out that nginx uses many calls with the idiom *malloc(num * size)* and does not attempt to detect integer overflows; this is safe as long as the values are small and it is guaranteed to not cause any integer overflows. But assumptions based on preconditions or non-obvious information, some of these values have size limits based on kernel-reported variables, are very dangerous and can always lead to mistakes. OpenBSD's reallocarray(3) has been designed to check for overflow, and to return NULL as failure in such a case (Copyright © 2008 Otto Moerbeek):

```
#define MUL_NO_OVERFLOW \
    (1UL << (sizeof(size_t) * 4))
if ((nmemb >= MUL_NO_OVERFLOW ||
    size >= MUL_NO_OVERFLOW) &&
    nmemb > 0 && SIZE_MAX / nmemb < size) {
        errno = ENOMEM;
        return NULL;
}
return realloc(optr, size * nmemb);
```

The aforementioned patch grew very big and eventually got discarded. It would have been very difficult to maintain it as a custom patch for nginx in OpenBSD's source tree, and there was no indication that upstream would have imported it any time soon

or even at all. But that wasn't the only aesthetically challenging issue in nginx, it turned out that nginx uses a number of custom memory allocators that preallocate and reuse almost all dynamic memory objects it needs. This reminded too much of the "Heartbleed" issue and OpenSSL's custom memory allocators that have been ripped out of LibreSSL. Software for OpenBSD must use the system's malloc(3) routines, and it must not sacrifice security for performance by bypassing all the randomization and the safety checks of "Otto Malloc" in libc.

Furthermore, the nginx code base grew larger since its initial import to OpenBSD in 2011. For good reason, OpenBSD had decided to follow stable nginx releases from upstream and to apply its local patches, like chroot, that did not make it back up. This work had its challenges and repeatedly reapplying the patches made it difficult to maintain. The nginx code base has many files and ever-increasing features, that might or might not matter for OpenBSD. The review of the code base revealed that it uses just too much code: custom reimplementations of standard C library functions, many layers, modules and even a local copy of the PCRE [Errata 1]library for extended regular expressions.

In June 2013, long before the g2k14 Hackathon, Reyk had disabled the SPDY module that is included in nginx and enabled by default [Errata 2]. That was a very controversial move that even raised dissent between senior developers in OpenBSD. *"Is there a reason to believe this specific piece of code is worse than any other?"*. There was no evidence - just the firm believe that *Security's worst enemy is complexity* and that too much code is always a potential risk. Additionally, SPDY got turned off because nobody in the OpenBSD project itself had spent time reviewing and understanding the protocol at this point. About a year later, this move has saved OpenBSD from two major security vulnerabilities in nginx: "SPDY heap buffer overflow"[1] and "SPDY memory corruption"[2].

1.3 OpenBSD's new httpd

There is a purely technical decision that lead to the creation of httpd, but there is also an anecdote about the motivation that kick-started the project.

After being frustrated about the impossibility to make larger changes in nginx, and the general direction of upstream and its associated company, Reyk tried to find an alternative. He is the main author and maintainer of relayd(8), OpenBSD's load balancer, and knew that it basically implements a highly-scalable HTTP engine. So while still being at the g2k14 Hackathon, he started one day in the afternoon experimenting with relayd to turn it into a simple chroot'ed web server that serves static files. A few hours later, the relayd-based server, conveniently called httpd, was able to serve a local copy of the OpenBSD web site.

At the same day, Reyk demonstrated the server experiment to Theo de Raadt and Bob Beck who sat and worked in the same "Hackroom" of the event. To the initial surprise of the author, they quickly got convinced about the attempt, and encouraged Reyk to import httpd into OpenBSD's source tree. Unlinked from the builds, but as visible part in the CVS repository. It was already a fairly late time of the day, and the developers had started drinking some sensible amounts of beer, when httpd was committed to OpenBSD. One day later, the developer wrote on Twitter: *"Today I woke up with sorrow and realized that I committed a web server last night"*[6].

It only took two more weeks, before httpd was linked to the builds, the OpenBSD 5.6 release was tagged and eventually shipped with the initial version of the new httpd. Many developers had contributed code, most notably Florian Obser for his implementation of FastCGI and Joel Sing for SSL/TLS. The 5.6 version of httpd was provided "as is", but fully functional as a basic web server that is secure, serves static files and supports FastCGI and TLS.

2 Design & Implementation

The design of httpd(8) is significantly influenced by its ancestor relayd(8): it inherited large parts of the code. It is a privilege-separated daemon, using `proc.c` and the *imsg* framework that is using async I/O with OpenBSD'S libevent. The web server is written with care and it is not obfuscated to shrink the size; but it consists of roughly 10,000 lines of code (relayd has about 24,000 lines of code).

The main development of the implementation happens in OpenBSD and the project's CVS repository. Occasionally, a mirror of the source code is pushed to the httpd repository on GitHub[7].

2.1 Simplicity

httpd(8) is designed to be a simple web server that performs some basic but yet powerful tasks. By design, it is intended to only support a selection of "core" features that are required for common, modern web applications. It shall not support special features and a major design objective is to keep it resonably small. Something is very unique to httpd: it uses a *Featuritis* label in its issue tracker to track rejected features[4] - and to remind everyone about the reasons later. Many feature requests have already been rejected because they would exceed the current scope or violate its simplicity.

Simplicity also means that it should be easy to use and provide a human-readble configuration file with sane defaults. `httpd.conf` is easy to use and a basic

web server configuration just needs 3 lines of configuration:

```
server "www.example.com" {
        listen on * port 80
}
```

2.2 Features

"Features" became a very negative word, as it indicates a need to have many of them. It is almost expected that new software releases introduce a new set of features. Developers add features, just to *add* new features and users demand new features, just *have* new features. It actually became very untypical to avoid features and to avoid *Featuritis*.

For httpd, it is more important what "core" functionality it needs, and what features it should not have. It is an ongoing struggle with users who want more features because only a few of them make it to the list. And, who knows, some of the existing features might be removed when they turn out to be unnecessary.

The following "core" functionality and concepts are currently supported and characterize the implementation:

Static files The main purpose is to serve static files and directories via optional auto-indexing. If auto-indexing is enabled, the directories are returned as HTML lists.

FastCGI The protocol provides the single and fast interface to serve dynamic content as an alternative to CGI or any other embedded scripting or CGI-like mechanism. It supports asynchronous and direct FastCGI via UNIX socket or TCP/IP.

Secure Non-optional security by running chroot'ed and with privilege separation by default.

TLS Supports secure connections via TLS powered by LibreSSL and the libtls API. The TLS stack uses state-of-the-art security standards and disables insecure or outdated SSL/TLS features.

Virtual servers Flexible configuration with support for name- and IP-based virtual servers on IPv4 and IPv6.

Reconfiguration The running configuration can be changed and reloaded without interruption or the need to restart the server.

Logging Supports per-server logging via local access and error files. Alternatively, logging via syslog is supported by default.

Blocking Blocking, dropping and redirections are required to prevent unauthorized access to subdirectories and to redirect to alternate URLs.

2.3 Security

OpenBSD runs its web servers in a chroot for many years; Apache and nginx have been patched to run chroot'ed by default. These patches have never been accepted by upstream, but yet they provide a significant benefit.

The world was shocked when the "shellshock"[13] vulnerability was discovered in 2014. While the vulnerbility itself was related to a bug in the GNU bash command line shell, the impact could have been mitigated by limiting the file system access of the affected web applications and servers. And web server should not be able to read raw database files, password files, or credentials from the file system - especially not as root. Shellshock exposed all kinds of problems that raised the questions like: *"Why does a web application of a multi-million dollar company execute privileged bash commands with full access to the file system?".*

httpd(8) has been designed to run chroot'ed by default. It cannot be turned off, the web server uses privilege separation to fork different processes, including the "parent", "logger" and "server" processes. The server [Errata 3]is the only privileged process that is running as root, the logger serializes log messages and writes them to the log files and the server processes accept and handle HTTP requests and FastCGI. All processes except the parent jail themselves to a chroot, typically to `/var/www`. Messaging between the processes is done with *imsg*.

The server is implemented with intensive peer review, following OpenBSD's coding practices, and uses style(9) aka. KNF. It implements file descriptor accounting to prevent starvation under heavy load and DoS.

2.4 TLS with LibreSSL

The OpenBSD project forked the LibreSSL SSL/TLS and crypto stack from OpenSSL in response to the Heartbleed vulnerability in its ancestor. The project was started *with goals of modernizing the codebase, improving security, and applying best practice development processes.*

As with any other portable project that originated from OpenBSD, primary development occurs in the OpenBSD source tree and portable version for various other operating systems like Linux, FreeBSD, Solaris, and Windows are regularly packaged based on this version.

httpd(8) uses LibreSSL's new libtls API that is provided as an alternative to the traditional libssl API from OpenSSL. It is designed to provide a simple and well-designed API that makes it easier and more secure to write TLS clients and servers. It is implemented on top of libssl and libcrypto without exposing the details of the underlying OpenSSL API; the backend library could even be replaced in the future.

The httpd(8) web server was used as a reference implementation for the server API, which was designed and written by Joel Sing with input from Ted Unangst, and Reyk Floeter. The client-side of the API was designed by replacing the libssl code in OpenBSD's ftp(1) tool. As httpd(8) was originally based on OpenBSD's relayd(8), some of it's SSL/TLS code was used as a reference for the implementation in libtls.

2.5 FastCGI

FastCGI is crucial to serve dynamic content, it is httpd's fast interface to run web applications, Apps, and to make it "web scale". It was initially designed to overcome the CGI protocol's scalability and resource sharing limitations. While CGI scripts need to be forked for every request, FastCGI scripts can be kept running and handle many HTTP requests. CGI is not directly supported by httpd.

The FastCGI implementation was written by Florian Obser and is based on his slowcgi server. slowcgi is a simple server that translates FastCGI requests to the CGI protocol; it runs chroot'ed and executes the requested CGI script, and translates its output back to the FastCGI protocol. The FastCGI implementation in httpd takes the HTTP requests, encapsulates them in FastCGI requests, and sends them to a FastCGI server such as slowcgi. The protocol itself is an open specification that is frame-based and allows to exchange encapsulated parameters and stdin/stdout payload between web- and fastcgi server.

When Reyk Floeter asked Florian Obser about his motivation to write slowcgi and httpd, he answered with the following statements:

> *"I implemented slowcgi because you didn't stop whining on icb that nginx can't execute bgplg".* And *"fastcgi in httpd: (Bob) Beck has asked me if I can help you with it".*

3 Configuration

The `httpd.conf` configuration file is using OpenBSD's modern style of a sane configuration language, that attempts to be flexible and humand-readable. It does not use a markup with semicola or tags, just english keywords and blocks identified by curly braces ("{}"). This is commonly called the `parse.y`-based configuration within OpenBSD, because it originates from the grammar and parser that was written for pf.

Within the configuration file there are four main sections: Macros (user-defined variables that may be defined and used later, simplifying the configuration file), Global Configuration (global settings for httpd(8)), Servers (Listening HTTP web servers), and Types (Media types and extensions).

Within the sections, a host address can be specified by IPv4 address, IPv6 address, interface name, interface group, or DNS hostname. If the address is an interface name, httpd(8) will look up the first IPv4 address and any other IPv4 and IPv6 addresses of the specified network interface. If '*' is given as an address, it will be used as an alias for 0.0.0.0 to listen on all IPv4 addresses. Likewise, '::' can be used to listen on all IPv6 addresses. A port can be specified by number or name; with names according to the `/etc/services` file.

The current line can be extended over multiple lines using a backslash. Comments can be put anywhere in the file using a hash mark ("#"), and extend to the end of the current line. Care should be taken when commenting out multi-line text: the comment is effective until the end of the entire block.

Argument names not beginning with a letter, digit, or underscore must be quoted. Additional configuration files can be included with the include keyword, for example:

```
include "/etc/httpd.conf.local"
```

3.1 Macros

As in all of OpenBSD's `parse.y`-based configuration files, macros can be defined that will later be expanded in context. Macro names must start with a letter, digit, or underscore, and may contain any of those characters. Macro names may not be reserved words (for example, directory, log, or root). Macros are not expanded inside quotes. For example:

```
ext_ip="10.0.0.1"
server "default" {
        listen on $ext_ip port 80
}
```

3.2 Global Configuration

The global configuration can be set to set defaults or to modify the runtime. Most of the global settings

cannot be changed on configuration reload.

chroot directory Set the chroot(2) directory. If not specified, it defaults to `/var/www`, the home directory of the www user.

logdir directory Specifies the full path of the directory in which log files will be written. If not specified, it defaults to `/logs` within the chroot(2) directory.

prefork number Run the specified number of server processes. This increases the performance and prevents delays when connecting to a server. httpd(8) runs 3 server processes by default.

3.3 Servers

The servers are specified as configuration blocks, marked by the "server" keyword, a name, and the actual server configuration encapsulated in curly braces ("{}").

alias name Specify an additional alias name for this server.

[no] authenticate [realm] with htpasswd Authenticate a remote user for realm by checking the credentials against the user authentication file htpasswd. The file name is relative to the chroot and must be readable by the www user. Use the no authenticate directive to disable authentication in a location.

block drop Drop the connection without sending an error page.

block [return code [uri]] Close the connection and send an error page. If the optional return code is not specified, httpd(8) denies access with a '403 Forbidden' response. The optional uri argument can be used with return codes in the 3xx range to send a 'Location:' header for redirection to a specified URI.

The url may contain predefined macros that will be expanded at runtime:

- $DOCUMENT_URI The request path.
- $QUERY_STRING Optional query string.
- $REMOTE_ADDR Remote IP address.
- $REMOTE_PORT Remote TCP source port.
- $REMOTE_USER Authenticated HTTP user.
- $REQUEST_URI Request path and optional query string.
- $SERVER_ADDR Server IP address.
- $SERVER_PORT Server TCP server port.
- $SERVER_NAME Server name.

connection option Set the specified options and limits for HTTP connections. Valid options are:

- **max request body number** Set the maximum body size in bytes that the client can send to the server. The default value is 1048576 bytes (1M).
- **max requests number** Set the maximum number of requests per persistent HTTP connection. Persistent connections are negotiated using the Keep-Alive header in HTTP/1.0 and enabled by default in HTTP/1.1. The default maximum number of requests per connection is 100.
- **timeout seconds** Specify the inactivity timeout in seconds for accepted sessions. The default timeout is 600 seconds (10 minutes). The maximum is 2147483647 seconds (68 years).

directory option Set the specified options when serving or accessing directories. Valid options are:

- **[no] auto index** If no index file is found, automatically generate a directory listing. This is disabled by default.
- **index string** Set the directory index file. If not specified, it defaults to index.html.
- **no index** Disable the directory index. httpd(8) will neither display nor generate a directory index.

[no] fastcgi [socket socket] Enable FastCGI instead of serving files. The socket is a local path name within the chroot(2) root directory of httpd(8) and defaults to `/run/slowcgi.sock`.

listen on address [tls] port number Set the listen address and port. This statement can be specified multiple times.

location path {...} Specify server configuration rules for a specific location. The path argument will be matched against the request path with shell globbing rules. A location section may include most of the server configuration rules except connection, listen on, location and tcp.

[no] log [option] Set the specified logging options. Logging is enabled by default using the standard access and error log files, but can be changed per server or location. Use the no log directive to disable logging of any requests. Valid options are:

- **access name** Set the name of the access log file relative to the log directory. If not specified, it defaults to access.log.
- **error name** Set the name of the error log file relative to the log directory. If not specified, it defaults to error.log.
- **style style** Set the logging style. The style can be common, combined or connection. The styles common and combined write a log entry after each request similar to the standard Apache and nginx access log formats. The style connection writes a summarized log entry after each connection, that can have multiple requests, similar to the format that is used by relayd(8). If not specified, the default is common.
- **[no] syslog** Enable or disable logging to syslog(3) instead of the log files.

pass Disable any previous block in a location.

root option Configure the document root and options for the request path. Valid options are:

- **directory** Set the document root of the server. The directory is a pathname within the chroot(2) root directory of httpd. If not specified, it defaults to `/htdocs`.
- **strip number** Strip number path components from the beginning of the request path before looking up the stripped-down path at the document root.

tcp option Enable or disable the specified TCP/IP options; see tcp(4) and ip(4) for more information about the options. Valid options are:

- **backlog number** Set the maximum length the queue of pending connections may grow to. The backlog option is 10 by default and is limited by the kern.somaxconn sysctl(8) variable.
- **ip minttl number** This option for the underlying IP connection may be used to discard packets with a TTL lower than the specified value. This can be used to implement the Generalized TTL Security Mechanism (GTSM) according to RFC 5082.
- **ip ttl number** Change the default time-to-live value in the IP headers.
- **[no] nodelay** Enable the TCP NODELAY option for this connection. This is recommended to avoid delays in the data stream.
- **[no] sack** Use selective acknowledgements for this connection.
- **socket buffer number** Set the socket-level buffer size for input and output for this connection. This will affect the TCP window size.

tls option Set the TLS configuration for the server. These options are only used if TLS has been enabled via the listen directive. Valid options are:

- **certificate file** Specify the certificate to use for this server. The file should contain a PEM encoded certificate.
- **ciphers string** Specify the TLS cipher string. If not specified, the default value "HIGH:!aNULL" will be used (strong crypto cipher suites without anonymous DH). See the CIPHERS section of openssl(1) for information about SSL/TLS cipher suites and preference lists.
- **dhe params** Specify the DHE parameters to use for DHE cipher suites. Valid parameter values are none, legacy and auto. For legacy a fixed key length of 1024 bits is used, whereas for auto the key length is determined automatically. The default is none, which disables DHE cipher suites.
- **ecdhe curve** Specify the ECDHE curve to use for ECDHE cipher suites. Valid parameter values are none, auto and the short name of any known curve. The default is auto.
- **key file** Specify the private key to use for this server. The file should contain a PEM encoded private key and reside outside of the chroot(2) root directory of httpd.
- **protocols string** Specify the TLS protocols to enable for this server. If not specified, the default value "all" will be used (all available protocols). Refer to the tls_config_parse_protocols(3) function for other valid protocol string values.

3.4 Types

Configure the supported media types. httpd(8) will set the Content-Type of the response header based on the file extension listed in the types section. If not specified, httpd(8) will use built-in media types for text/css, text/html, text/plain, image/gif, image/png, image/jpeg, and application/javascript.

The types section must include one or more lines of the following syntax:

type/subtype name [name ...] Set the media type and subtype to the specified extension name. One or more names can be specified per line. Each line may end with an optional semicolon.

include file Include types definitions from an external file, for example `/usr/share/misc/mime.types`.

3.5 Example

All of the previous configuration options can be used to create amazingly obvious configuration files. httpd(8) does not install a long default configuration file, the minimal file only needs 3 lines (as illustrated previously) and all the defaults are built-in. The following advanced example include some additional directives:

```
server "www.example.com" {
        listen on * port 80
        listen on * tls port 443

        # Logging is enabled by default
        #no log

        location "/download/*" {
                directory auto index
                log style combined
        }
        location "/pub/*" {
                block return 301 \
                    "http://ftp.example.com/\
                    $REQUEST_URI"
        }
        location "*.php" {
                fastcgi socket \
                    "/run/php-fpm.sock"
        }
        location "/cgi-bin/*" {
                fastcgi
                root "/"
        }
        root "/htdocs/www.example.com"
}
```

4 Conclusion

httpd has been implemented very quickly, it only took two weeks for OpenBSD 5.6 and an additional release cycle of about 4 months for the upcoming 5.7 release, and it became a serious web server and worthy replacement for OpenBSD's nginx/Apache. Many users have already started to use it, and the community has accepted it very well. It will take many more years, maybe forever, to make it perfect, but part of this effort will include removal and renewal of code.

5 Appendix

5.1 About the Author

Reyk Floeter is the founder of Esdenera Networks GmbH[3], a company that develops OpenBSD-based networking and security products for cloud-based and software-defined networks. For more than ten years, he gained experience in creating and commercially supporting enterprise-class products based on OpenBSD, like most recently the Esdenera Firewall. Reyk is located in Hannover, Germany, but works with international customers like Internet Initiative Japan Inc. (IIJ) in Tokyo[9]. He is the author of the popular relayd load balancer and a hacker in the OpenBSD[12] project, where he contributed various features, fixes, networking drivers and daemons since 2004, like OpenBSD's ath, trunk (a.k.a. lagg), vic, hostapd, relayd, snmpd, iked, and httpd.

References

[1] Maxim Dounin, *"SPDY heap buffer overflow", nginx security advisory (CVE-2014-0133)*, http://mailman.nginx.org/pipermail/nginx-announce/2014/000135.html?_ga=1.137394864.1013509854.1363164706.

[2] ———, *"SPDY memory corruption", nginx security advisory (CVE-2014-0088)*, http://mailman.nginx.org/pipermail/nginx-announce/2014/000132.html?_ga=1.169007649.1013509854.1363164706.

[3] Esdenera, *Esdenera Networks GmbH*, http://www.esdenera.com/.

[4] Reyk Floeter, *httpd Issue Tracker*, https://github.com/reyk/httpd/issues?q=label%3Afeaturitis+.

[5] ———, *relayd*, http://bsd.plumbing/.

[6] ———, *reykfloeter@ "I committed a web server last night"*, https://twitter.com/reykfloeter/status/488262609981145088.

[7] ———, *reyk/httpd at GitHub*, https://github.com/reyk/httpd.

[8] The Apache Software Foundation, *The Apache HTTP Server Project*, http://httpd.apache.org/.

[9] IIJ, *Internet Initiative Japan Inc.*, http://www.iij.ad.jp/.

[10] Nginx Inc., *NGINX*, http://nginx.com/.

[11] OpenBSD, *OpenBSD 5.6*, http://www.openbsd.org/56.html.

[12] ———, *The OpenBSD Project*, http://www.openbsd.org/.

[13] Florian Weimer, *"shellshock", CVE-2014-6271: remote code execution through bash*, http://seclists.org/oss-sec/2014/q3/649.

6 Errata

Section 1

Errata 1. (page 3) PCRE was only bundled with OpenBSD's version of nginx. The upstream version never had a local copy of the PCRE library (see `http://nginx.org/en/docs/configure.html`).
Errata 2. (page 3) SPDY is disabled by default in the upstream version of nginx (see `http://nginx.org/en/docs/http/ngx_http_spdy_module.html`). However, it was enabled by the developer who imported nginx 1.4.1 into OpenBSD at this time `http://marc.info/?l=openbsd-cvs&m=137010319309380&w=2`.

Section 2

Errata 3. (page 4) It should say: "The *parent* is the only privileged process that is running as root"