Apache Security -

Improving the security of your web server by breaking into it

Sebastian Wolfgarten,

21C3, December 2004

sebastian.wolfgarten@de.ey.com

Agenda

- Preface
- Introduction to Apache
- History of vulnerabilities
- Basic principles of Apache security
- Configuration analysis
- Advanced Apache security
- Demonstration
- Summary
- References

Preface

$ whoami

- Student of business & computer science at the University of Cooperative Education in Stuttgart/Germany
- Working with Ernst & Young's Risk Advisory Services (RAS) group for more than 2 years
- Specialized in network security, pen-testing and IT forensics
- Author of more than a dozen articles for various German IT magazines as well as three books (e.g. "Apache Webserver 2") for the Addison & Wesley publishing house
- Reviewer for Addison & Wesley and O'Reilly US

Introduction to Apache

There are many cowboys, but there is only one Apache

- Originally based on the NCSA httpd 1.3 written by Rob McCool (University of Illinois) and for the first time published in April 1995
- Powerful, modular, flexible, highly configurable, extensible and freely available Open Source web server
- Apache currently runs on approx. 68% of all web servers worldwide making it the #1 choice ever since 1996

Introduction to Apache (cont.)

There are many cowboys, but there is only one Apache

- Currently three different version branches (1.3.x, 2.0 and 2.1) available whereas only 2.0 and 2.1 are being actively developed
- 1.3.x is proven most stable but is feature-wise frozen (only bug-fixing)
- 2.x introduces a bunch of cutting-edge features including various runtime-models (MPMs), multi protocol support, APR, IPv6, in- and output filtering etc.

History of vulnerabilities

Buffer overflows and various other vulnerabilities

- In the past Apache (and its modules) suffered from various locally and remotely exploitable vulnerabilities including (digest):
  - mod_include Local Buffer Overflow Vulnerability (published: 20.10.2004, CVE: CAN-2004-0940)
  - mod_ssl SSLCipherSuite Restriction Bypass Vulnerability (published: 11.10.2004, CVE: CAN-2004-0885)
  - mod_proxy Remote Negative Content-Length Buffer Overflow Vulnerability (published: 10.06.2004, CVE: CAN-2004-0492)
  - Satisfy Directive Access Control Bypass Vulnerability (published: 23.09.2004, CVE: CAN-2004-0811)
  - mod_userdir Module Information Disclosure Vulnerability (published: 04.12.2003, CVE: n/a)
- However as a consequence of its design there hasn't been a single r00t vulnerability in Apache.

Basic principles of Apache security

Non-Apache related system security (digest)

- Host and system security is an important part of any (Apache) installation.
- Therefore minimize a server's exposure to current and future threats by fully configuring the operating system and removing unnecessary applications.
- Verify the authenticity and integrity of all software products used.
- Ensure to install the latest patches and versions available.

Basic principles of Apache security

Non-Apache related system security (cont.)

- Disable or restrict unnecessary system services
- Use strong encryption
- Use strong passwords and strong password policies
- Remove default accounts, change default passwords
- Apply OS hardening procedures
- Use firewalls/IDSs
- ….

Basic principles of Apache security

Security-related Apache configuration directives

- Apache provides quite a number of security-related configuration directives enabling the administrator to tighten the security, e.g.:
  - User / Group: Defines the user and group Apache should run as
  - AllowOverride: Types of directives allowed in external configuration files (aka .htaccess files)
  - LimitRequestBody: Restricts the total size of the HTTP request body sent from a client
  - LimitRequestFields: Limits the number of HTTP request header fields that will be accepted from the client

Basic principles of Apache security

Security-related Apache configuration directives (cont.)

- Furthermore:
  - LimitRequestFieldSize: Limits the size of the HTTP request header allowed from the client
  - LimitRequestLine: Limits the overall size of the HTTP request line that will be accepted
  - Listen: Defines the IP addresses and ports the server listens on
  - Options: Configures what features are available in a particular directory
  - Order: Controls the default access state and the order in which Allow and Deny are evaluated

Basic principles of Apache security

Security-related Apache configuration directives (cont.)

- Finally there is:
  - Proxy: Container for directives applied to proxied resources
  - ServerTokens: Configures the Server HTTP response header
  - ServerSignature: Defines the content of the footer available on server-generated documents
  - SSLEngine: This directive toggles the usage of the SSL/TLS protocol engine
  - UserDir: Indicates the location of user-specific directories
  - AuthDigestFile: Use HTTP Digest Authentication

Basic principles of Apache security

The do's and don'ts of httpd.conf

- User & Group directive:
  - Unix/Linux: Create a separate user and group to run the Apache, disable the login for that user and assign a non-existing home directory (e.g. useradd -g wwwgroup -d /dev/null -s /bin/false wwwuser)
  - Windows 2003 Server: Things on Microsoft Windows are a bit more complicated as the User and Group directive does not exist:
    - Install Apache as a service into a separate directory (e.g. C:\Apache2) and add a new group as well user to the system.
    - Assign a strong password, do not give the user the permission to change his password.
    - Add that user to the newly created group and remove it from all other groups (e.g. Users).
    - Ensure the user is not able to login remotely using Terminal Services, do not connect client drives or printers, do not grant the user access to the main (default) printer and deny any remote access.
    - Most importantly start the Apache service as that user instead of the Local System account.

Basic principles of Apache security

The do's and don'ts of httpd.conf (cont.)

- File & directory permissions:
  - Unix/Linux: Install Apache as root, chmod 600 to all config files and 500 to the httpd binary (optional)
  - Windows 2003 Server: Only grant all permissions (Full Control, Modify, Read & Execute, List Folder Contents, Read, Write) on the installation folder of the Apache to the Administrator as well as the new user id running the Apache and remove all users and permissions (including inherited permissions) from that folder.
- Logging:
  - Log and analyze everything (maybe even use mod_log_forensic) and ensure the logs have not been tampered with

Basic principles of Apache security

The do's and don'ts of httpd.conf (cont.)

- HTTP fingerprinting:
  - Use the ServerTokens and ServerSignature directive to prevent people from fingerprinting your HTTP server:
    - ServerSignature Off
    - ServerTokens Prod
    - Ultimately send a customized Server:-header (Apache 2 only!) or modify the source code directly
- Symbolic links:
  - Disable symbolic links
  - If necessary re-enable them for certain directories and use SymLinksIfOwnerMatch to make the server only follow symbolic links for which the target file or directory is owned by the same user id as the link

Basic principles of Apache security

The do's and don'ts of httpd.conf (cont.)

- Indexing:
  - Disable indexing to prevent content from being accidentally exposed to the public and eventually found by Google (e.g. "Index of /backup"):
    - Options None or Options -Indexes
  - If necessary, re-enable it only for certain directories you are aware of
- SSI:
  - Best practice: Disable server-side includes completely
  - If necessary, use suexec, enable SSI and disable certain commands (e.g. #exec cmd and #exec cgi):
    - Options –IncludesNOEXEC
    - XBitHack off
  - Note: Users will still be able to #include virtual CGI scripts from ScriptAliased directories.

Basic principles of Apache security

The do's and don'ts of httpd.conf (cont.)

- .htpasswd and .htaccess files:
  - Disable access to those files completely as they possibly contain sensitive information:

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

  - This is a default configuration but some people seem to disable this functionality?!
- AllowOverride:
  - Beware of the power of those directives that can be used in .htaccess files!
    - Use AllowOverride AuthConfig or AllowOverride None

Basic principles of Apache security

The do's and don'ts of httpd.conf (cont.)

- Default content:
  - Remove any default content (e.g. manual, icons, CGI scripts, samples etc.) as well as any third-party stuff or vendor gadgets
- Modules:
  - Simple rule: Due to performance and security reasons disable all modules that you do not explicitly need (candidates for instance are: mod_usertrack, mod_status, mod_proxy*, mod_isapi, mod_info, mod_include, mod_imap, mod_example, mod_dav*, mod_cern_meta, mod_autoindex, mod_userdir, mod_auth_anon, mod_asis)

Configuration analysis

Default httpd.conf

- Most administrators (>80%) use the default configuration provided by the Apache
- This configuration file is fine but may be optimized security-wise by
  - Define an explicit IP address and port Apache should listen on
  - Define a user and group Apache should run as
  - Remove any default content (e.g. manual, CGI scripts), unused modules as well as possibly vendor-provided extras (e.g. SDB)
  - Restrict access to local file system
  - Reduce amount of information leakage

Configuration analysis

Default httpd.conf (cont.)

- Set an interface to listen on as well as a user and group:
  - Listen A.B.C.D:80
  - User wwwuser
  - Group wwwgroup
- Disallow access to the root directory of the file system:

```
<Directory />
Options FollowSymLinks
AllowOverride None
</Directory>

<Directory />
Options None
AllowOverride None
Order Deny,Allow
Deny from all
</Directory>
```

Configuration analysis

Default httpd.conf (cont.)

- Disable directory indexing and symbolic links

```
<Directory “/usr/local/apache2/htdocs">
Options Indexes FollowSymLinks
Order allow,deny
Allow from all
</Directory>

<Directory “/usr/local/apache2/htdocs">
Options None
Order allow,deny
Allow from all
</Directory>
```

Configuration analysis

Default httpd.conf (cont.)

- Remove mod_userdir to disable user directories
- Restrict the banners displayed to a minimum

  ServerTokens Full
  ServerSignature On

  ServerTokens Prod
  ServerSignature Off
- If you are more paranoid use mod_headers to send a customized Server:-header (or modify the source code directly)

Configuration analysis

Default httpd.conf (cont.)

- Disable and remove default directories (e.g. /icons/, /manual/), e.g.

```
Alias /icons/ "/usr/local/apache2/icons/"
<Directory "/usr/local/apache2/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Configuration analysis

Typical configuration mistakes

- Not keeping up to date
- Use of standard Apache configuration (modules, information leakage, indexing, directory access)
- Improper privilege separation
- Insecure use of third party modules (e.g. PHP) or applications
- Open proxies
- Application-level vulnerabilities
- …

Advanced Apache security

Chrooting Apache

- Chroot replaces the root directory of a process with one of the users' choosing, effectively creating a sandbox in which he or she selectively allows access to a small number of operating system features. So even if an attacker compromises a system he or she has only access to the sandbox environment instead of the entire system.
- However the setup of a chrooted or jailed (*BSD) environment is a very complex and time-consuming task and should be performed by experienced admins only.

Advanced Apache security

Chrooting Apache (cont.)

- Instructions for chrooting Apache 2 are available on the web (e.g. slashr00t.org, securityfocus.com), however generally speaking the following steps must be taken:
  - Download and install Apache
  - Create a stripped-down copy of the local file system which should act as a basis for the chroot environment
  - Copy all files, directories, programs and libraries needed by the Apache into that environment
  - Optionally install the chroot environment on a separate partition with the nosuid option set and remove all suid as well as sgid programs etc. possibly allowing an attacker to escape
  - Modify Apache‘s start scripts etc. to reflect the new setup

Advanced Apache security

Watch your parameters with mod_parmguard

- Quoting securityfocus.com: “Mod_parmguard is a module that intercepts the requests and rejects those which are not compliant with the constraints expected by the scripts. Use a XML configuration file that describe the type and allowed values for the parameters. “
- Equipped with automated tools helping the administrator to build that XML-based configuration file possibly preventing hackers from tempering with the web applications available.
- Even though the module is considered stable, it is not yet ready for production (further testing necessary)

Advanced Apache security

Watch your parameters with mod_parmguard (cont.)

- Sample html form:

```
<HTML>
<BODY>
<FORM ACTION=“input.php” METHOD=GET>
Name: <INPUT TYPE=TEXT NAME=“name” SIZE=10>
Age: <INPUT TYPE=TEXT NAME=“age” SIZE=2>
Salutation:
<SELECT name=“salutation”>
<OPTION value=“ms”>Ms</OPTION>
<OPTION value="mr">Mr</OPTION>
</SELECT>
<INPUT TYPE=SUBMIT VALUE=“Submit”>
</FORM>
</BODY>
</HTML>
```

Advanced Apache security

Watch your parameters with mod_parmguard (cont.)

- Appropriate mod_parmguard XML config file (partially):

```
<match>input.php</match>
<parm name="name">
    <type name="string"/>
    <attr name="maxlen" value="10"/>
    <attr name="charclass" value="^[a-zA-Z]+$"/>
</parm>
<parm name="age">
    <type name="integer"/>
    <attr name="minval" value="10"/>
    <attr name="maxval" value="99"/>
</parm>
<parm name=“salutation">
    <type name="enum"/>
    <attr name="multiple" value="0"/>
    <attr name="option" value="ms"/>
    <attr name="option" value="mr"/>
</parm>
```

Advanced Apache security

Fighting denial of service-attacks

- According to its web site nuclearelephant.com, “mod_dosevasive is an evasive maneuvers module for Apache to provide evasive action in the event of an HTTP DoS or DDoS attack or brute force attack. It is also designed to be a detection and network management tool, and can be easily configured to talk to firewalls and routers and etc. mod_dosevasive presently reports abuses via email and syslog facilities.”

Advanced Apache security

Fighting denial of service-attacks (cont.)

- Detection is performed by creating an internal dynamic hash table of IP Addresses and URIs, and denying any single IP address from any of the following:
  - Requesting the same page more than a few times per second
  - Making more than 50 concurrent requests on the same child process per second
  - Making any requests while temporarily blacklisted (on a blocking list)
- In combination with various other techniques (e.g. ingress- and egress-filtering - see RFC2827, traffic shaping, forbid IP spoofing) the module can (partially) help mitigate against DoS and DDos attacks.

Advanced Apache security

Introducing mod_security

- Quoting modsecurity.org: “ModSecurity is an open source intrusion detection and prevention engine for web applications. Operating as an Apache web server module, the purpose of modsecurity is to increase web application security, protecting web applications from known and unknown attacks.”
- It basically provides request filtering, anti-evasion techniques, POST payload analysis, extensive audit logging, built-in chroot functions and HTTPS filtering.

Advanced Apache security

Request filtering with mod_security

- Define rules to target common web application attacks (selective targeting possible):

  # Command execution attacks
  - SecFilter /etc/password
  - SecFilter /bin/ls

  # Directory traversal and XSS attacks
  - SecFilter "\.\./“
  - SecFilter "<(.|\n)+>"
  - SecFilter "<[[:space:]]*script"

  # SQL injection attacks
  - SecFilter "delete[[:space:]]+from"
  - SecFilter "insert[[:space:]]+into"
  - SecFilter "select.+from"

Advanced Apache security

Request and response filtering with mod_security

- Continued:

  # Forbid file upload
  - SecFilterSelective "HTTP_CONTENT_TYPE" multipart/form-data

  # MS SQL specific SQL injection attacks
  - SecFilter xp_enumdsn
  - SecFilter xp_filelist
  - SecFilter xp_availablemedia
  - SecFilter xp_cmdshell
  - SecFilter xp_regread
  - SecFilter xp_regwrite
  - SecFilter xp_regdeletekey

  # Prevent a vulnerable script from being exploited
  - SecFilterSelective "ARG_recipient" "!@de.ey.com$

  # Output filtering
  - SecFilterSelective OUTPUT "Fatal error:" deny,status:500

Advanced Apache security

Logging with mod_security

- Mod_security provides enhanced logging capabilities:

  # Enable debugging and increase debug level
  - SecFilterDebugLog logs/mod_sec_debug.log
  - SecFilterDebugLevel 3

  # Audit logging and attack response
  - SecAuditLog /usr/local/apache2/logs/mod_sec_audit.log
  - SecAuditEngine DynamicOrRelevantOnly

Advanced Apache security

More magic with mod_security

- Mod_security provides built-in chroot support:
  SecChrootDir /usr/local/apache2
- Set any server signature
  - SecServerSignature “Microsoft-IIS/5.0”
- Verify uploaded files (e.g. use virus scanner):
  - SecUploadApproveScript /path/to/some/script
- Flexible attack response (e.g. log, deny, redirect, delay, exec etc.)
  - SecFilter KEYWORD “exec:/home/seb/report.pl”

Demonstration

A little less presentation, a little more action

Demonstration #1:

As part of a reverse proxy setup mod_security (on Apache) successfully shields a vulnerable web application on a Microsoft Internet Information server.

Summary

Coming closer to the end…

- First of all: There is no 100% security.
- But: After years of development (~9-10 years) and going through an enormous testing process, Apache IS quite secure (even out of the box) *hurray*.
- Additionally using the techniques described in this document (as well as those described in other guidelines) Apache's security can even be tightened more providing a presumably high level of security.

Further information

Good reads on Apache (offline)

- “Apache Security”, Ivan Ristic, O’Reilly, 2005 (ISBN: n/a), not yet released
- “Maximum Apache Security”, anonymous, Sams Publishing, 2002 (ISBN: 0-672-32380-X)
- “Apache Webserver”, Lars Eilebrecht et. al, Mitp, 2003 (ISBN: 3-826-61342-2)
- “Apache Webserver 2 - Installation, Konfiguration, Programmierung”, Sebastian Wolfgarten, Addison & Wesley, 2nd edition 2004 (ISBN: 3-827-32118-2)

Further information

Apache-related online resources (digest)

- Apache manual, http://httpd.apache.org
- ApacheWeek, http://www.apacheweek.com
- Ryan C. Barnett, “Securing Apache” (SANS), http://www.cgisecurity.com/lib/ryan_barnett_gcux_practical.html
- Artur Maj, “Securing Apache: Step-by-Step”, http://www.securityfocus.com/infocus/1694
- Ivan Ristic, http://www.modsecurity.org
- Denice Deatrich, “Chrooting Apache”, http://penguin.triumf.ca/chroot.html

Apache Security

Acknowledgements

- I would now like to thank the following people for helping me creating this presentation:
  - Ivan Ristic, Thinking Stone Ltd.
  - Hugh Callaghan, Ernst & Young Dublin/Ireland
  - Jens Wolfgarten, Pharmapp Solutions GmbH
  - Krisztian Piller, European Central Bank

Apache Security

The end.

Thanks for your (long) patience
and attention!

I would now like to
answer your questions.

By the way, this presentation (and various other Apache and security-related material) is available online at http://www.slashr00t.org.