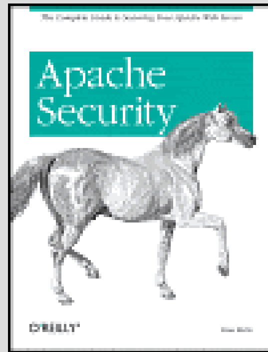
Securing Apache on Unix/Linux



Apache security books reviewed at
http://www.apachetutor.org/security/

(and a bunch of FAQs)

Bugs can be anywhere

- Apache itself
- Library
- Third-party Module
- Own module
- Scripting Language or Module
- Own or third-party script
- Application server
- Own or third-party application
- Configuration

Risk factors

- Scrutiny. On a scale from Apache itself to in-house application.
- Complexity. More modules, libs, apps mean more places for bugs to hide.
- Tradeoffs. PHP gets lots of scrutiny, but famously puts features and ease-of-use ahead of security.

Risks to Apache

- Application module, or script running under mod_php, mod_perl, etc have potential to kill or compromise Apache.
- Application server transfers the risk to itself.
- A CGI script under mod_cgi can only kill itself

- BUT, any of the above can potentially threaten the host system!

Risk Classification

WASC – Web Application Security Consortium

- Authentication
- Authorization
- Clientside Attacks
- Command Execution
- Information Disclosure
- Logical Attacks

www.webappsec.org

Protection

- Multi-level security
- Programmers and sysops
- Scrutiny
- Containment
- Detection

Your tradeoff

- You need complex custom applications, homebrew or untrusted scripts, etc.
- You need security.
- How to reconcile the two?

Non-Defences

Little or no effect (in general):

- Firewall
- DMZ
- IDS
- SSL

Defences

- Due diligence
- Sandboxing
- Log analysis
- Operating system limits
- Apache limits
- Active checking of incoming data:
  - Perl taint - the pioneer
  - mod_security – SecFilter, SecFilterSelective

Unix Basics

- Users and Groups
- File ownership, chown
- File permissions, chmod, umask
- Special bits: setuid, setgid

Advanced:

- Filesystem mount flags
- chroot
- SELinux

Obtaining Apache

- From apache.org mirror or trusted third-party (distro).
- Checking the download
  - pgp secure
  - md5 checksums better than nothing but can be forged

Apache Installation

- System files usually determined by packager
- Apache User: no privileges, no files, no shell

User apache

apache:x:123:456:web server:/dev/null:/bin/false

Group apache

apache:x:456

Apache Installation

- System files usually determined by packager
- Apache User: no privileges, no files, no shell
- Site Owners: some privileges (normal users)

All webpages, scripts, etc owned by site owners. Not by apache user or root.

chroot/jail

A highly-restricted sandbox within a computer.
A jailed program is strictly confined by the O/S.

Both security books have chroot howtos

Terminology: when is chroot the same as jail?

- chroot: a jail available everywhere
- FreeBSD jail: a richer deluxe sandbox

chroot(2)

- Runs Apache in a tightly-controlled sandbox with no access to anything outside it.
- Presents a major barrier to escalating privileges in the event of a successful attack on Apache.
- Keeps useful tools out of reach of an attacker.
- Not easy to administer!

chroot(3)

- Classic chroot
  - Import everything Apache and applications need into the jail
  - When running “kitchen sink” apps like PHP, the jail ends up containing a full toolkit for the benefit of an intruder
- mod_security or mod_chroot
  - Easy setup and startup
  - Reduces (but doesn't eliminate) need for useful tools accessible within the jail
  - Restart/graceful fails

Filesystem privileges

Most permissive:

- 644 -rw-r--r-- Files
- 755 -rwxr-xr-x Directories and Scripts

Note: Apache requires access to parent directories in the filesystem, or you'll get 403 (forbidden) errors!

Protecting users from each other

- 640 -rw-r-----
- 750 -rwxr-x---

Requires Apache user to be made a member of users groups, subject to local policies.

- Group1 = user1, apache
- Group2 = user2, apache

suexec: allows scripts to be segregated.

Writing to the filesystem

- Inherently Dangerous!

The worst real-life exploits to have affected Apache servers involve uploading and executing a malicious script or program.

Good News: We can protect against this, even if we are running untrusted and possibly-buggy applications!

Protecting the filesystem

Apache owns nothing!
Where is world-writable?
Where is group-writable by Apache?

/tmp, /var/tmp, /your/writable/path:

- mount with noexec,nosuid flags, so the operating system prevents upload+exec

$USER/some_path (outside your control)

suexec

- Protect users from each other while allowing scripts
- Delegates security to users
- Imposes a lot of 'good practice' checks
- Potentially circumvents apache security!
- Don't forget to read the suexec log!

User Directories and files

Users areas are outside your control. It may be impossible to enforce filesystem-based security. selinux offers a more flexible but more complex alternative.

Fedora: security context (user:role:type)

root_u:system_r:**httpd_t**

allowing access to **httpd_sys_content_t**

selinux

From Fedora Wiki:
http://docs.fedoraproject.org/selinux-apache-fc3/

```
ls -aZ /var/www/
drwxr-xr-x  root     root     system_u:object_r:httpd_sys_content_t .
drwxr-xr-x  root     root     system_u:object_r:var_t             ..
drwxr-xr-x  root     root     system_u:object_r:httpd_sys_script_exec_t
cgi-bin
drwxr-xr-x  root     root     system_u:object_r:httpd_sys_content_t
error
drwxr-xr-x  root     root     system_u:object_r:httpd_sys_content_t html
drwxr-xr-x  root     root     system_u:object_r:httpd_sys_content_t
icons
```

Secure Defaults

- New contents are inaccessible to the webserver by default.
- Use chcon explicitly to make contents available
- Strict control over server-writable and executable contents in userspace.
- Impose restrictions on scripts, including suexec.

Web Application Protection

- Apache Configuration
- ulimit
- Perl Taint checking
- mod_security

Configuration

- Limit resources available to Apache and applications
  - Containment of buggy scripts, modules, etc

- Limit the length of request line, HTTP headers, request body
  - DoS protection
  - Buffer overflow

Taint checking

Proactively cleanse all input from untrusted sources.

```
$untrusted =~ /^(\w+)\.(\w{3})$/
        or input_error($untrusted);
my $filename = $1.$2;
```

More restrictive than necessary, but safe!

Taint checking

- Proactively cleanse all input from untrusted sources.
- Powerful protection against malicious attacks.
- Targeted: untrusted input can be used raw in safe operations.
- A great learning tool for programmers.
- Perl-only.
- Sysop puts faith in programmer.

mod_security

- Puts sysop in charge of sanitising data
- Separates taint-like checking from the application
- Supports all application types
- Extensive logging

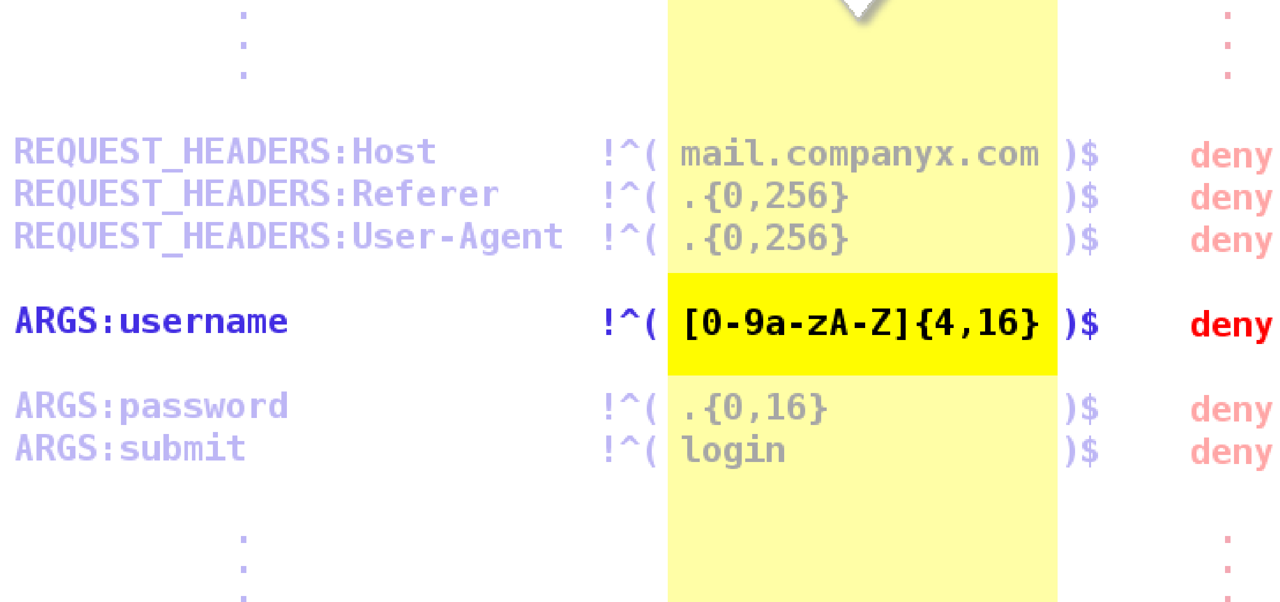
- Complex to administrate
- Slow if applied to request contents

mod_security

- Core Rules (open)
  - Reject bogus HTTP
  - Detect common attacks and common bots
  - Detect access to trojans/backdoors
  - Suppress information leaks in server errors.
- Console (commercial): manage multiple servers, with reporting and alerts.
- Remo: whitelist-oriented ruleset editor

POST /action/submit.php
- Headers
  - Host: mail.companyx.com
  - Referer: .{0,256}
  - User-Agent: .{0,256}
- Postparameters
  - username: [0-9a-zA-Z]{4,16}
  - password: .{0,16}
  - submit: login

The positive rule from Remo is translated into a whitelist ModSecurity rule. This means, that you have to define the good arguments in Remo. Requests with arguments, that do not match this positive definition, are considered bad requests (note the exclamation point in the regex below). They are dropped by ModSecurity.



Information Disclosure

- Security by Obscurity doesn't work
- BUT, the converse isn't true: exposing system information may help an attacker!
- mod_security rulesets – clues on information that should be filtered.
- mod_security is not a good filter, but output filter modules such as mod_publisher or mod_line_edit can strip out sensitive info.

Virtual Hosts

- Running as different users: limited support (suexec, ruid, fastcgi, MPMs)
- Apache Configuration is normally per-virtualhost, but may not be.
- Ditto associated configuration such as PHP.
- You cannot load modules per-vhost!

Web Passwords

- Basic Authentication – simple, low-security
- Digest Authentication – better
- Basic + SSL; Custom schemes

- htpasswd / htdigest
- Database
- Directory Services

- SSL Client Certificates

Web Passwords

Problem: All forms of Web authentication are open to brute-force attack due to statelessness of HTTP. So there is a lower barrier to entry than with system passwords.

Consequence: Never use web credentials that also gain access to the operating system or other high-value target. Not even when cryptographically secure!