

Hardening Enterprise Apache Installations Against Attacks

Sander Temme
sander@temme.net

November 6, 2008

Abstract

Enterprise installations of Apache are particularly attractive targets for malicious attacks including Denial of Service, defacement, theft of data or service and installation of zombies or viruses. Hardening your deployment against such attacks calls for some special techniques and tactics. This article discusses attack detection techniques, server protection, secure deployment of multiple servers, configuration of firewall "demilitarized zones" and judicious use of SSL encryption. How do you deploy an off-the-shelf application that insists on writing to the file system? And what steps do you take to securely deploy Apache on Windows or UNIX?

1 Introduction

This is the companion paper to the ApacheCon session *Hardening Enterprise Installations against Attacks*. It describes the threat model that faces these installations, the security and vulnerability mitigation process at the Apache HTTP Server project, and how to securely deploy the Apache HTTP Server (httpd). Finally, it discusses vulnerabilities specific to web applications that run on top of Apache httpd, and gives some specific examples of how common exposures can be mitigated.

1.1 An Enterprise Installation

For the purpose of this paper, an *Enterprise* installation is defined as one where the web application and its servers are managed directly by the organization that owns the site. The responsibility for the servers, operating system and application patching and the components of the application itself lie with members of the organization, and are not part of an outsourced Service Level Agreement.

According to this definition, a mom-and-pop store that runs its web site from a server in the back room is an Enterprise installation, and a Fortune 500 company that outsources its website to Amazon.com is not.

2 The Threat Model

Before we organize our defense, we need to know what to defend against.

2.1 Attack Motives

In the early, friendlier days of the web, most attacks were carried out with the intent to deface the targeted websites. The attacker would find a way to upload their own content to the webserver, often proudly displaying a message boasting the attacker's hacking prowess. White Hat hackers would expose vulnerabilities in the configuration of a web server installation by subtly altering the content, then alerting the owner. This happened to the apache.org webserver in 2000, when vulnerabilities in the FTP server and MySQL service (but not in the Apache web server) were exploited to put a "powered by Microsoft BackOffice" logo on the home page of www.apache.org¹.

Unfortunately, those days are long past. While defacements still occur, to-days attacks on web server installations often occur with monetary gain as primary motive. Broadly speaking, attack motives can be categorized as follows:

- Data theft—for instance, an attacker sends malicious SQL queries into your system to read the customer database of your application, which may contain personal data or credit card numbers
- Zombify your server—Make your server execute the attackers' applications for their nefarious purposes in addition to your own. For instance, your server could be used as Command & Control for a Botnet, or as a mail relay for spam campaigns
- Upload rogue content—the traditional defacement falls under this, but also for instance uploading malicious JavaScript code into an online forum to effect Cross-Site Scripting (XSS) attacks on their users' browsers. Another use for rogue content is hosting malicious scripts that can be executed by other websites, or a form page that can be the target of Phishing e-mails

An attack, perpetrated for financial gain, can be much harder to detect than a simple defacement. The attackers have nothing to gain by making themselves known, so they will not alter or erase your content. They will add their own, and subvert the software on your system to suit their needs and avoid detection. Fortunately, as described in [9], they do not always succeed in remaining undetected.

¹<http://www.dataloss.net/papers/how.defaced.apache.org.txt>

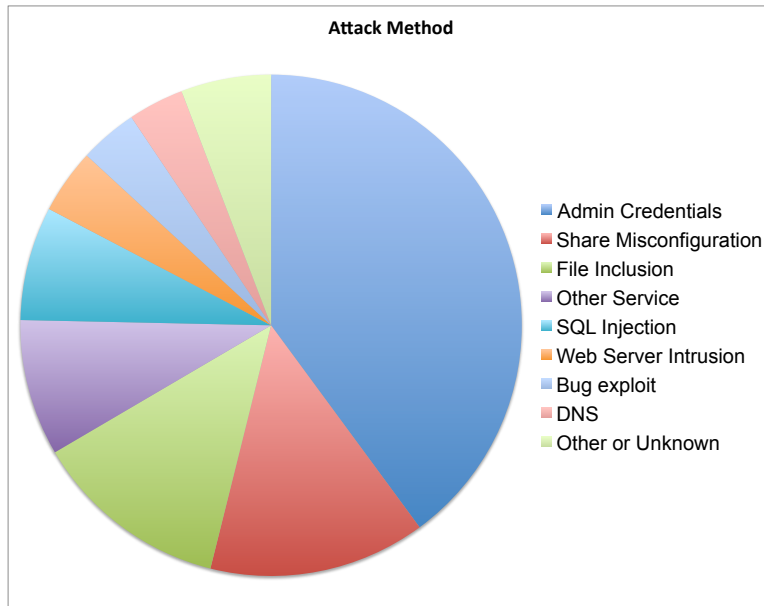


Figure 1: Attack methods recorded in a 2007 report on defacements by zone-h.org

3 Apache HTTP Server Security

3.1 Security Track Record

When determining how secure the Apache HTTP Server is, it is important to make a distinction between the HTTP Server core, and the applications that run on top of it. Web application vulnerabilities will be discussed later. By itself, the Apache HTTP Server has an excellent security track record. You can find information on HTTP Server vulnerabilities on the project web site² and the page on Apache 2.2 shows that there have been no critical vulnerabilities discovered in that version of the web server. And, all of the vulnerabilities that have been found are fixed in the latest version. Also in that directory are an XML file that describes all vulnerabilities in any version of Apache, and a document that describes the impact levels assigned to vulnerabilities as they are discovered and the criteria the security team uses to assign these levels.

²<http://httpd.apache.org/security/>

3.2 Security Process

The Apache HTTP Server project has a process in place to handle vulnerabilities in the Apache code as they are discovered. As described on the website³, when someone discovers a vulnerability in the Apache source code, they are asked to report this to the security@apache.org e-mail address. When the security team receives a report that describes a new, hitherto unknown vulnerability, they assign it an identifier with the Common Vulnerabilities and Exposures list (CVE)⁴. The vulnerability is then classified and fixed, and a new version of httpd is released.

Not all reports that arrive at the Apache Security Team mailbox discuss actual vulnerabilities. To quote a report made by the security team to the August, 2008 Apache Software Foundation board meeting⁵:

Attachment 4: Status report for the Apache Security Team Project

There continues to be a steady stream of reports of various kinds arriving at security@apache.org. These continue to be dealt with promptly by the security team. For July 2008:

- 1 Support question
- 1 Security vulnerability question, but not a vulnerability report
- 1 Phishing/spam/attacks point to site "powered by Apache"
- 3 Vulnerability report

Note that the statistics given each month are for queries sent to security@apache.org and does not include any that are sent to specific project lists advertised separately such as security@tomcat.apache.org. Most projects do not advertise separate lists (or really need to given the low volume of issues affecting most projects), and the only one which gets really any direct reports is security@tomcat. We'd only advise a project advertising a separate security response address if they get or expect a significant number of issues.

For these board reports we do not plan on giving more detail about specific issues unless they are significant in some way (critical vulnerability or threat) as issues can take several months through the lifecycle of dealing with the reporter during which time they are usually non-public.

For interest now we have two years of data, here is the cumulative total emails to security@apache.org for each type:

	Jul-Dec06	Jan-Jun07	Jul-Dec07	Jan-Jun08	Total
Support	24	14	25	13	[76]
Query	11	10	4	11	[36]
PoweredBy	17	20	19	11	[67]
NotASFHack	7	5	0	3	[15]
Report	24	23	23	20	[90]
Total	[83]	[72]	[71]	[58]	[284]

Support : Support question, not vulnerability related. We won't answer these but will refer them to some public list.

Query : Security vulnerability question, but not a vulnerability report. We answer some of these but in most cases refer to a public list for discussion.

³http://httpd.apache.org/security_report.html

⁴<http://cve.mitre.org/>

⁵http://www.apache.org/foundation/records/minutes/2008/board_minutes_2008_08_20.txt

PoweredBy : Phishing/spam/attacks point to site "powered by Apache". We try to help the users understand what happened, but many still don't believe us, or don't understand.

NotASFHack : User was hacked, but after investigation it turns out it wasn't ASF software at fault. Note that there isn't a "WasASFHack" row because we've not yet heard from anyone whose machine was compromised where it turned out to be via some flaw (fixed or unfixed) in ASF software.

Report : What the list is designed for, a vulnerability report. We include here all reports of possible vulnerabilities even if they turn out not to be vulnerabilities (as they require effort to investigate and/or respond). It's pretty constant though the years.

The Apache Security Team prefers to follow the Responsible Disclosure approach where a vulnerability is brought to the attention of the software developer before it is disclosed to the general public. This gives the httpd development community the opportunity to implement a fix. Additionally, the Apache HTTP Server has been adopted by many organizations including Linux distributors, software and operating system vendors etc. The disclosure time window allows these third parties to catch up and release a fix before a vulnerability becomes publicly known.

If a vulnerability is discovered and considered *Critical*, the reaction is typically swift: a release that fixes the issue can happen in days. If an issue is not considered critical, like a minor vulnerability that occurs under very specific circumstances in a module that is not enabled by default, a fix may have to wait for the next regular release of httpd. This release schedule, and the vulnerability impact assessment, may not be according to the expectations of the original reporter. It has happened in the past that the Apache Security Team considered a reported issue non-critical and deferred resolution to a regular release, only to have the reporter go public and shout from the rooftops that they had discovered a vulnerability in Apache, and the Apache development team were unresponsive. Outbursts like that serve nobody's purpose: not the httpd project, not the users and ultimately not the reporter who invariably see their credibility hurt when the Security Team has to speak up and put their reports in their place in public.

3.3 How to Keep Up-to-date

When a new version of the Apache HTTP Server is released, an message is posted to the `announce@apache.org` mailinglist. Anyone who uses httpd and has built it directly from the source code made available on `apache.org` should subscribe to this low-volume mailinglist or its RSS feed⁶. If a new release fixes any security issues or vulnerabilities, this will be stated in the announcement.

If you have installed Apache through a vendor package, you can typically subscribe to an information channel from that vendor to be apprised of updates. You may even get automated updates as they are released by the vendor. However, if your vendor does not have such an information channel, the

⁶http://mail-archives.apache.org/mod_mbox/httpd-announce/?format=atom

announce@apache.org mailinglist is a good source of information. It can serve as a heads-up that a new package should be on its way, or a reminder to ask the vendor for an update.

Finally, to get the most up-to-date information and learn about security releases before they are announced to the general public, subscribe to the dev@httpd.apache.org mailinglist. On this mailinglist, fixes to security issues are discussed before they are applied to the source code, and calls are made to test upcoming releases before they are announced to the general public. And by weighing in on issues as they are discussed on the list, you become a member of the Apache development community yourself.

4 Securely Deploying Apache

4.1 Installing from Source

When the Apache Software Foundation releases the Apache HTTP Server, the release consists of the source code to httpd. To use the web server, you must compile this source code. Occasionally, members of the Apache committer community make compiled binaries available for various platforms, but this is done as a courtesy and the binaries are not the official release.

To compile Apache from source, you need a C compiler on your machine, the *make* program and a Perl interpreter. Perform the following steps:

1. Download the source code from <http://httpd.apache.org/download.cgi> . Be sure to download the signature and hash as well as the source code archive (tarball) itself
2. Verify the integrity of the archive by running `md5 -r httpd-2.2.XX.tar.gz` and comparing the output to the contents of the `httpd-2.2.XX.tar.gz.md5` file. On some systems, this program is called `md5sum`
3. Verify the digital signature on the archive (see Section 4.2 below)
4. Run `./configure --prefix ...` to configure the build. Invoke `configure` with `--help` to see the available options.
5. run `make` to build the server
6. run `su make install` to install the server

A more comprehensive guide to compiling the server is available on the httpd website⁷. Building your own Apache HTTP Server allows you to:

- have the opportunity to always work with the latest version of Apache
- decide how the server is configured and built

⁷<http://httpd.apache.org/docs/2.2/install.html>

- decide how dependencies are met
- apply your own patches to the source code or configuration files before building the server

Disadvantages of building your own server include:

- You need a compiler on your server system. Not everyone wants to have a compiler on their production boxes, since it may introduce attack vectors
- You can't take advantage of automated updates and patches from your distributor, but have to build updated versions of Apache as they are released, and test them in your environment
- Building your own web server and associates libraries and modules may require expertise that you don't have or in which you don't want to invest

4.1.1 Security Aspects of Building Apache

As you can see in Section 4.1, installing Apache from source involves few if any explicit security-related steps. There is, frankly, little you can do at compile time to improve the security of the software. The real difference is made when the software is configured, and this will be discussed in Section 4.3.

Some insist that Apache be built with statically compiled modules, to prevent attackers from substituting compromised modules for the real ones. However, if the server is so thoroughly compromised that an attacker can replace Apache modules on disk, or write to the Apache configuration file, they could just replace the entire Apache binary, whether it has loadable modules or not.

4.2 Digital Signatures

The Apache source code distribution, as well as many vendor-supplied binary packages, are digitally signed. Package installation utilities tend to verify the digital signature on the packages as part of the installation process. Unless the verification fails, this is completely transparent to the user experience. The public key for the signature on the packages is typically installed with the operating system, so the trust model is limited to the user's confidence that their installation media were created by the operating system vendor in question.

The Apache Software Foundation creates a PGP signature for every package released under the auspices of the foundation, using the personal PGP key of the Project Management Committee member who makes the release. The members of the various PMCs within the Apache Software Foundation should make themselves available on a regular basis to have people sign their keys, and expand their web of trust. This allows users who download Apache software to verify the signatures on the source code releases, and find the signature trusted by either a direct signature on the releaser's key, or an indirect line of trust between them and the key that signed the release. The PGP Keysigning Parties held at ApacheCon conferences present an excellent opportunity for this.

While Apache source code is generally downloaded from a mirror server, the signatures are served from an `apache.org` server. To verify the signature on a download, place the download and the signature file in the same directory and run `gpg --verify` with the signature file as argument. For instance:

```
[sctemme@Legadema] dist $ gpg --verify httpd-2.2.10.tar.bz2.asc
gpg: Signature made Tue Oct  7 10:41:26 2008 CDT using DSA key ID 08C975E5
gpg: Good signature from "Jim Jagielski <jim@apache.org>"
gpg:          aka "Jim Jagielski <jim@zend.com>"
gpg:          aka "Jim Jagielski <jim@jimjag.com>"
gpg:          aka "Jim Jagielski <jim@jaguNET.com>"
gpg:          aka "Jim Jagielski <jimj@covalent.net>"
gpg:          aka "Jim Jagielski <jim.jagielski@springsource.com>"
```

4.3 Configuring Apache for Security

Tips:

- Remove unused modules from configuration, both their `LoadModule` statement and their configuration
- Remove conditionals from the configuration file: if a module you need isn't present, your server should fail instead of running without that necessary functionality. If the functionality of a particular module isn't necessary, it doesn't need to be in your configuration
- Do not forget to lock down the default virtual host. No browser will land there, but clients that simply connect to the IP address of the server, or that don't send a `Host:` header for whatever reason, will see the default `Vhost`.

5 Practical Considerations

5.1 Running Apache on Windows

Ever since the days of Apache 2.0, the Microsoft Windows operating system has been a fully supported platform. While the Windows port of Apache 1.3 was of a highly experimental nature, the Apache Portable Runtime library and the Multi-Process Module design of Apache 2.0 have allowed Apache to take full advantage of the Win32 APIs. This opens up possibilities for those who are considering deploying Apache, but are not comfortable managing or unable to deploy a Unix-like platform on which to run it. Discussion of the relative security merits of each individual platform falls outside the scope of this paper. However, in general it is advisable to deploy on a platform that is known to its administrators, whatever it is. There are several unique aspects of deploying Apache on Windows that have a direct bearing on security:

- The Windows MPM spawns one child process, with `MaxClients` threads. The process model on Windows makes deploying multiple child processes

(like the Worker MPM does on Unix) very inefficient, but a single, multi-threaded child can perform quite well. If an attacker succeeds in crashing this child process, the parent will take some time (several seconds) to spawn a replacement. Once a crash has been identified, this makes it relatively easy to carry out a Denial Of Service (DOS) attack. This is unlikely to change.

- Unlike the Unix MPMs, the Windows MPM does not change the user ID of the child process. This means the child runs as the same user ID as the parent, and has the same filesystem privileges as the parent. The user should take care to limit the write privileges of the Apache server to as few locations as possible, preferably only the logs directory. This will be discussed below.

5.1.1 Reconfigure Server Root

When you run the Apache HTTP Server installer, it will install the entire server, including the configuration, content and log directories, under `C:\Program Files (C:\Program Files (x86) on Windows Server 2008)`. This is undesirable, because it mixes the program code with your website content and makes it harder to upgrade your software without affecting the local configuration. Make an easily accessible directory (for instance `C:\website`) to hold the local configuration and content, and create underneath that the directories `conf`, `logs` and `htdocs`. Put your static content in the `htdocs` subdirectory, and create a file named `httpd.conf` in the `conf` subdirectory. This will hold your custom site configuration, but for now it only needs to have a `Listen 80` statement in it. Then, stop the `Apache2.2` service and reconfigure it to use the custom server root directory. Issue the following commands on a Command Prompt:

```
net stop Apache2.2

cd "C:\Program Files (x86)\Apache Software Foundation\Apache 2.2

bin\httpd -n Apache2.2 -d C:\website -k config

net start Apache2.2
```

The `-n` parameter to `httpd` is the Service name. The installer sets this to `Apache2.2`, but you can define your own service name if you want. You will have to fill out the `httpd.conf` file to make the server do anything useful, but from now on the Windows Service will use `c:\website` as server root.

5.1.2 Run Apache as a Custom User

As discussed above, Apache does not change its user ID on Windows. The default installation runs as the Local System account, which has way too many write privileges for comfort. Create a Local User on the system, for instance

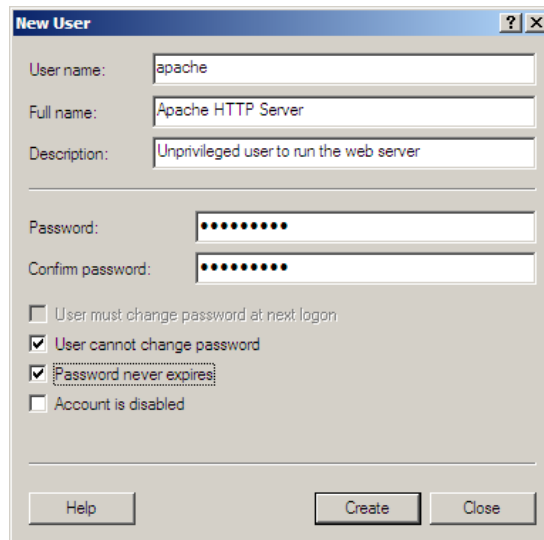


Figure 2: Make a custom local user to run the Apache HTTP Server

with username *apache*, and instruct the `Apache2.2` Service to log on as that user.

5.1.3 Limit Write Access

Now that you have the `Apache2.2` Service running as a custom user, you should limit that user's write privileges to the file system. Ideally, the *apache* user should only have write access to its own `c:\website\logs` directory. Use the `AccessChk`⁸ utility to quickly determine the privileges of your user to a given directory.

You may have to remove your *apache* user from the *Users* group, and break inheritance on the `C:\website` directory in order to more specifically assign permissions. The *apache* user needs to be able to read the directories and files under the server root. In addition, it needs to be able to create, write to and delete files in the `logs` subdirectory, but nothing else. The *apache* user will end up with write access elsewhere on the system but, as long as it can't overwrite parts of the operating system, this is generally not of concern. The main objective of restricting write permissions is to prevent the *apache* user from writing files it could then serve out in some way.

⁸<http://technet.microsoft.com/en-us/sysinternals/bb664922.aspx>

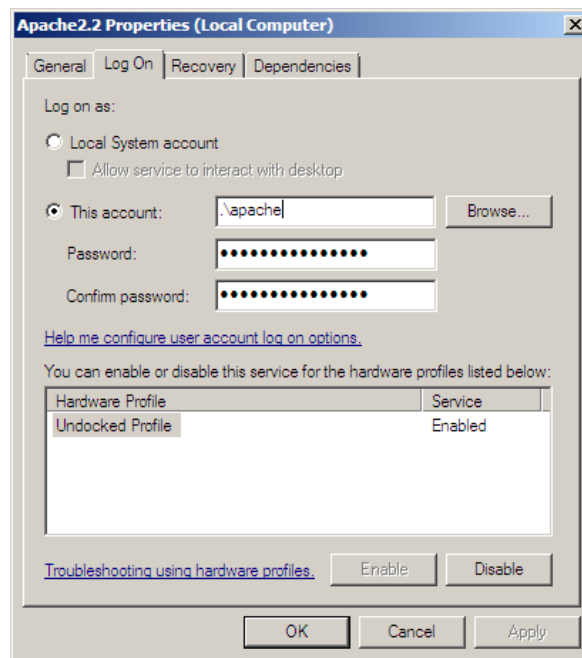


Figure 3: Have the Apache2.2 Service log on as a custom user

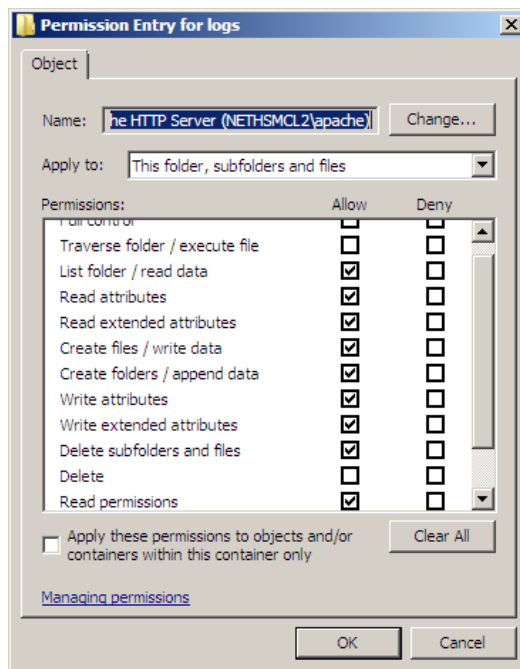


Figure 4: Give the *apache* user write access only to the *logs* directory under the server root

5.1.4 Restarting Apache

When you tell the Windows Service manager to restart a Service, it will stop, then start the Service in question. This obviously causes an interruption in service. However, you can gracefully restart Apache, just like on Unix. If you just want it to re-read its configuration file, or turn over log files, a graceful restart is all you need. Simply issue

```
\path\to\httpd.exe -n Apache2.2 -k restart
```

or use the Apache Monitor in your Sys tray to gracefully restart the server.

References

- [1] Mike Andrews and James A. Whittaker. *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Addison-Wesley Professional, 2006.
- [2] Ryan C. Barnett. *Preventing Web Attacks with Apache*. Addison-Wesley Professional, 2005.
- [3] Jon Erickson. *Hacking: The Art of Exploitation*. No Starch Press, San Francisco, CA, USA, 2003.
- [4] OWASP Foundation. The ten most critical web application security vulnerabilities, 2007 update, 2002-2007. http://www.owasp.org/index.php/Top_10_2007.
- [5] Tony Mobily. *Hardening Apache*. APress, 2004.
- [6] The Apache HTTP Server Project. Reporting security problems with apache, March 2008. http://httpd.apache.org/security_report.html.
- [7] Ivan Ristic. *Apache Security*. O'Reilly Media, Inc., 2005.
- [8] Erik Schetina, Ken Green, and Jacob Carlson. *Internet Site Security: Architecture to Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [9] Lars Strand. Holiday cracking, April 2007. <http://blog.gn1st.org/article.php?story=HolidayCracking>.
- [10] Zone-H. Statistics report 2005-2007, March 2008. <http://www.zone-h.org/content/view/14928/30/>.