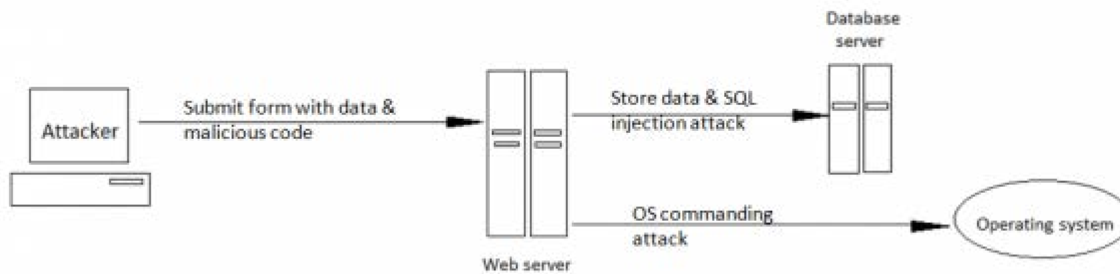
SECURING APACHE: FOOL-PROOFING THE SERVER OS

Moving deeper into Web application and Apache security, let's now focus on OS commanding attacks, and those that lead to the disclosure of crucial information and server directory paths. The attacks described below are often overlooked or trivialised.

OS Commanding

OS commanding (a.k.a. OS command execution, shell injection) is a vulnerability in websites that lets attackers execute operating system commands through the manipulation of application input. It occurs when scripts execute external commands that are constructed using unsanitised input data. Such commands will run with the privileges of the component (database server, Web application server, Web server, wrapper or application) that executed the command. This lets the attacker gain access to otherwise unreachable areas (e.g., operating system directories and files), inject unexpected and dangerous commands, upload malicious programs or even obtain passwords directly from the operating system. The diagram below illustrates this.



An OS commanding attack

OS commanding vulnerabilities are frequently found in Perl and PHP scripts, because these programming environments encourage programmers to reuse operating system binaries. However, there are also chances of such vulnerabilities in Java and ASP. Let’s take a look at a few Perl and PHP examples.

The following Perl CGI code is part of a Web application for server administration. This function allows administrators to specify a directory on the server, and view a summary of its disk usage:

```
#!/usr/bin/perl
use strict;
use CGI qw(:standard escapeHTML);
print header, start_html("");
print "<pre>";
my $command = "du -h --exclude php* /var/www/html";
$command= $command.param("dir");
$command=`$command`;
print "$commandn";
print end_html;
```

When used as intended, this script simply appends the value of the user-supplied dir parameter to the end of a specific command, which it executes, and displays the results. A normal usage URL might be something like http://www.example.com/cgi-bin/showinfo.pl?dir=/public.

This functionality can be exploited in various ways, by supplying crafted input containing shell characters that have a special meaning to the command interpreter. For example, the pipe character (|) is used to redirect output from one process to the input of another, enabling the chaining of multiple commands. An attacker can leverage this to inject a second command (and retrieve its output) with a malicious URL such as http://www.example.com/cgi-bin/showinfo.pl?dir=/public|%20cat&20/etc/passwd. Here, the original du command's output is redirected as input to the cat /etc/passwd command, which simply ignores it, and just outputs the contents of the passwd file.

Note: Most operating systems restrict access to system files. Also, the advent of shadow passwords has rendered this specific attack less useful than it previously was. Still, damaging attacks can be made by obtaining .php files which may contain database hostnames, usernames and passwords, or other configuration data, or by obtaining the database files themselves.

PHP also provides functions, such as passthru() and exec(), and the back-tick operator, to execute external programs. Consider the following PHP code, which displays a list of files in a particular user's home folder:

```
$output = `ls -al /home/$username`;
echo $output;
```

A valid URL for this might be http://www.example.com/viewfiles.php?username=arpit. If a semicolon is used

in the input, it will mark the end of the first command, and the beginning of a second. An example of a malicious URL would be http://www.example.com/viewfiles.php?username=arpit;cat%20/etc/passwd, which again displays the contents of the passwd file.

With such a vulnerability, attackers can execute any binary on the server like starting a telnet server, logging in to it with the privileges of a Web server user, performing exploits to gain root access, and perhaps attacking other hosts that are reachable from the compromised server.

The time for security

OS commanding is on the list of least-addressed vulnerabilities, but the following security measures can help curb it considerably:

1. Remove the execution bit from the *everyone* group (-rwxrwxrw-) for OS commands. The Web server user account will not be able to execute the targeted command even if an attacker is able to trick the Web application into attempting to execute it.
2. Validate all input fields for characters like ;, |, / or %00. Now, since the list of such characters can be endless, the best thing would be to only allow the expected input, and filter out everything else — create a whitelist of acceptable inputs that strictly conform to specifications, and reject any other input, or transform it into acceptable input. For example, to neutralise the ampersand character (&) that appears in user data, and which needs to be a part of an HTML page, it must be converted to &.
3. As additional security, you can also try filtering out command directory names such as bin,sbin, opt.

Path-traversal (a.k.a. directory traversal) attacks

Web servers are generally set up to restrict public access to a specific portion of the server's filesystem, typically called the "Web document root" directory. This directory contains files and any scripts that provide Web application functionality.

In a path-traversal attack, an intruder manipulates a URL in such a way that the Web server executes, or reveals the contents of, a file anywhere on the server — including outside the document root. Such attacks take advantage of special-character sequences in URL input parameters, cookies and HTTP request headers.

The most basic path traversal attack uses the `../` character sequence to alter the document or resource location requested in a URL. Although most Web servers prevent this method from escaping the Web document root, alternate encodings of the `../` sequence, such as Unicode-encoding, can bypass basic security filters. Even if a Web server properly restricts path-traversal attempts in the URL path, any application that exposes an HTTP-based interface is also potentially vulnerable to such attacks.

Note: For UNIX systems, the parent directory is `../` while in Windows it is `...`

Attack Scenario 1

The valid URL `http://www.example.com/scripts/database.php?report=quarter1.txt` is used to display a text file. However, manipulating it into a malicious URL like `http://www.example.com/scripts/database.php?report=../scripts/database.php%00txt` will force the PHP application to display the source code of the `database.php` file, treating it as a text file whose contents are to be displayed.

The attacker uses the `../` sequence to traverse one directory above the "current" directory, and enter the `/scripts` directory. The `%00` sequence is used both to bypass a simple file extension check, and to cutoff the extension when the file is read and

processed by PHP. This example highlights the critical importance of always checking and cleaning user-supplied input before allowing it to be processed.

Attack Scenario 2

The PHP code below accepts a username, and then opens a file specific to that username. It can be exploited by passing a username that causes it to refer to a different file:

```
$username = $_GET['user'];
$filename = "/home/users/$username";
readfile($filename);
```

Let's suppose a normal URL is www.example.com/profile.php?user=arpit.pdf. If an attacker passes a changed query string to make a malicious URL like www.example.com/profile.php?user=../../etc/passwd then PHP will read /etc/passwd and output that to the attacker.

Path-traversal attacks mostly target an application's file upload, download, and display functionalities, like those often found in work-flow applications (where users can share documents); in blogging and auction applications (when users upload images); and in informational applications (when users retrieve documents like ebooks, technical manuals, and company reports).

In many cases, there may be security measures in place — filters for forward or backward slashes — but here too, attackers can try simple encoded representations of traversal sequences, such as those shown the following table.

Table 1: Some encoding schemes

Character	URL encoding	16-bit Unicode encoding	Double URL encoding</>
dot	%2e	%u002e	%252e
forward slash	%2f	%u2215	%252f
backslash	%5c	%u2216	%255c

The time for security

Path traversal attacks can be addressed with the following security measures:

1. There's really no good reason for Apache to be allowed to serve files outside of its document root. Any request for files outside the document root is highly suspect, so we'll restrict it to a directory structure with the following directives in the httpd.conffile:

```
<Directory />
Order Deny, Allow
Deny from all
Options none
AllowOverride none
</Directory>
<Directory www>
Order Allow, Deny
Allow from all
Options -Indexes
</Directory>
```

2. (Replace the `www` directory name with whatever you’ve called your Web server’s document root. The `Options -Indexes` line in the `<Directory www>` section disables directory browsing, securing the server from directory-traversal attacks.)
3. Apart from this, ensure the user account of the Web server or Web application is given the least read permissions possible for files outside the Web document root. Also, change the default locations of your Web root directories.
4. After performing all relevant decoding and Canonicalisation of user-submitted filenames, validate all inputs so that only an expected character set (such as alpha-numeric) is accepted. The validation routine should be especially aware of shell meta-characters such as `/` and “and” command-concatenation characters (`&&` for Windows shells and the semi-colon for UNIX shells).
5. Set a hard limit for the length of a user-supplied value. Note that this step should be applied to every parameter passed between the client and server, not just parameters the user is expected to modify via text boxes or similar input fields.
6. The application should use a predefined list of permissible file types, and reject any request for a different type. It is better to do this before the decoding and Canonicalisation has been performed.
7. Any request containing path-traversal sequences should be logged as an attempted security breach, generating an alert to an administrator, terminating the user’s session, and if applicable, suspending the user’s account.
8. `realpath()` and `basename()` are two functions PHP provides to help avoid directory-traversal attacks. `realpath()` translates any `.` or `..` in a path, resulting in the correct absolute path for a file. For example, the `$filename` in *Attack Scenario 2*, passed to `realpath()`, would return just `/etc/passwd`. On the other

hand, basename() strips the directory part of a name, leaving just the filename itself. Using these two functions, it is possible to rewrite the script of *Attack Scenario 2*in a much more secure manner:

```
$username = basename(realpath($_GET['user']));
$filename = "/home/users/$username";
readfile($filename);
```

Tools of the secure trade

1. Dotdotpwn is a very flexible Perl-based intelligent fuzzer tool, which detects several directory-traversal vulnerabilities on HTTP/FTP servers. For Windows systems, it also detects the presence of boot.ini on vulnerable systems through directory-traversal vulnerabilities. It is available for free download on its website, along with its documentation.
2. This ~~web resource~~ contains many path-traversal URLs that are frequently used by attackers. This domain also contains other good resources on security and ethical hacking.

Source-code disclosure

Source-code disclosure, another variant of the path-traversal attack, is a widely prevalent vulnerability in Web applications, which lets attackers extract source code and configuration files. Such vulnerabilities are found mostly in websites that offer to download files using dynamic scripts.

The attacker uses this technique to obtain the source code of server-side scripts like ASP, JSP or PHP files, to discover Web application logic, including database structure, source code comments, parameters and other possibly exploitable vulnerabilities of the code. Let’s understand this using a simple attack scenario.

An attack scenario

Let’s assume a website uses the following PHP code, which initiates a file download from the server:

```
<?php
if(isset($_GET[‘file’]))
{
   $file = $_GET[‘file’];
   readfile($file);
}
?>
```

A valid URL for the above script is http://www.example.com/downloads.php?file=arpit.zip, but the attacker’s malicious URL could be http://www.example.com/downloads.php?file=login.php, which returns to the attacker the contents of the file login.php. With this, the attacker learns about the filters and checks in login.php, and even the names of other crucial database and systems configuration files.

To secure your application from source-code disclosure, the application should use a predefined list of permissible file types, and reject any request for a different type.

Directory listing leakage

This is a commonly-found vulnerability in many Web servers. When a Web server receives a request for a directory rather than an actual file, it may respond in one of three ways:

1. It may return a (configurable) default resource within the directory, such as index.html,home.html, default.htm, default.asp, default.aspx, index.php, etc.

2. It may return an HTTP status code 403 error message, indicating that the request is not permitted.
3. It may return a listing showing the contents of the directory. This happens when default resources are not present in the directory.

In many situations, directory listings do not have any relevance to security. For example, disclosing the listing of an images directory may be completely inconsequential. Indeed, directory listings are often intentionally allowed because they provide a built-in means of navigating sites containing static content. But still, some files and directories are often, unintentionally, left within the Web root of servers, including:

- **Application-generated files:** Web-authoring applications often generate files that find their way to the server. A good example is a popular FTP client, WS_FTP, which places a log file into each folder it transfers to the Web server. Since people often transfer folders in bulk, the log files themselves are transferred, exposing file paths and allowing the attacker to enumerate all files. Another example is CityDesk, which places a list of all files in the root folder of the site, in a file named citydesk.xml.
- **Configuration-management files:** Configuration-management tools create many files with metadata. Again, these files are frequently transferred to the website. CVS, the most popular configuration-management tool, keeps its files in a special folder named CVS. This folder is created as a sub-folder of every user-created folder, and it contains the files Entries, Repository, and Root.
- **Backup files:** Text editors often create backup files with extensions such as ~, .bak, .old,.bkp, and .swp. When changes are performed directly on the server, backup files remain there. Even when created on a development server or

workstation, by virtue of a bulk folder FTP transfer, they end up on the production server.

- **Exposed application files:** Script-based applications often consist of files not meant to be accessed directly, but instead used as libraries or subroutines. Exposure happens if these files' extensions are not recognised by the Web server as a script. Instead of executing the script, the server sends the full source code in response to a request. With access to the source code, the attacker can look for security-related bugs. Also, these files can sometimes be manipulated to circumvent application logic.
- **Web server's crucial information:** This is often displayed at the end of the listing page, and contains the server name, version number and other important information. Such information can be used to launch specific exploits against the Web server.

Apart from the above, there are many other files which should not be disclosed publicly, such as temporary files, renamed old files, user's private home folders, etc.

Now, the question is regarding how attackers can gain access to directory listings. I am not going the route of guessing the path of a crucial directory via URI. For this, attackers can simply use Google's advanced search operators (now it's obviously a prerequisite to know Google's advanced search operators, site:, inurl:, intext:, intitle:, etc).

Note: I once again stress that neither LFY nor I are responsible for the misuse of the information given in this article. The attack techniques are meant to give you the knowledge that you need to protect your own infrastructure. You will be held solely responsible for any misuse of this knowledge.

Google searches for directory listings

Searching for intitle:index.of site:example.com will give you the directory listing ofexample.com if the directory listing is enabled on the server — and in most cases, it is. Such a result may contain any of the above crucial files. Similarly, searching for intitle:index.of ws_ftp.log or intitle: index.of 'parent directory' 'ws_ftp.ini' will return directory listings that contain the files ws_ftp.log or ws_ftp.ini. In the same manner, attackers can also search for .bak, .bkp, .swp and other vulnerable extensions and files.

Another popular query among attackers is site:example.com AND intitle: 'index.of' 'backup' which yields the backup directory (if it exists on the example.com domain). A similar query, site:example.com AND ext:log, will yield all the log files on the domain.

Note: The AND boolean operator combines two or more conditions in the same query.

Another such query is "robots.txt" + "Disallow:" ext:txt site:example.com. This yields therobots.txt file for example.com, which tells the attacker about paths and directories that the website admin doesn't want Google's spider to crawl.

A more dangerous query is "phpMyAdmin" "running on" inurl:"main.php", which leads attackers straight to the phpMyAdmin page on the server — one of the worst nightmares of an administrator. If you try this out, please *don't* click through to any of the many results!

Other useful trial queries that you can guess the purpose of, are:

- intitle:index.of.password
- intitle:"Index of" _vti_inf.html
- allinurl:auth_user_file.txt
- allinurl:admin mdb
- site:edu filetype:xls +bank account

- filetype:bak inurl:"htaccess|passwd|shadow|htusers"

The story does not end here… there are many more queries that can be generated and tested.

Time for security

Information leakage via directory listing and Google is a big issue, and also requires almost no expertise; hence, it falls in the category of dangerous vulnerabilities. You can follow these security steps to safeguard yourself:

1. Putting in blank index.htm files simply prevents directory listings from being displayed to site visitors. Add some text like, "Your access has been logged" if you wish to fool script kiddies.
2. To disable directory listing in Apache, open the .htaccess file and look for Options Indexesunder the directory you want to disable, and modify it to Options –Indexes. Do the same in your httpd.conf file, and then restart Apache. Directory listing should be disabled now.
3. If you need to keep listing enabled, then do not link any files that may provide crucial information to attackers.
4. It's a common misconception that you can rely on the Disallow feature of robots.txt, because some search engines still crawl unwanted pages/directories of the server.

Tools of the secure trade

- Goolag, an automated Google search application, helps you find some impressive information about your Web application. Download it along with its documentation, from here.

- Gooscan is another tool that automates queries against Google search appliances but these particular queries are designed to find potential vulnerabilities on Web pages.

Information leakage

Information leakage occurs when a website reveals sensitive data which may aid an attacker in exploiting the system. It does not necessarily represent a security breach, but it provides the attacker information that may be useful for future exploitation. There are many ways of forcing a Web application to leak this type of information.

HTML comments

Website programmers often leave comments in HTML source code. These are mainly for their self-reference, and sometimes for other programmers to see. However, often these comments help attackers in understanding the target better. For example, look at the HTML code below:

```
<TABLE border="0" cellPadding="0" cellSpacing="0" height="59"
width="591">
<TBODY>
<TR>
<!--If the image files are missing, restart ZBEAST -->
<TD bgColor="#ffffff" colSpan="5" height="17" width="587"> </TD>
```

Here we see a comment left by the programmer indicating what one should do if image files do not show up. The key information here is the host name of the server, which is mentioned explicitly in the code, ZBEAST.

JavaScript code

Parts of JavaScript code, especially those dealing with data validation, can reveal a lot of information about application business rules. Often, programmers implement data validation on the client side itself, thinking that this will reduce server-side load. This is a mistake.

Tools’ comments and metadata

Tools that are used to create Web pages (either server-side or client-side) also often leave comments in the code, which could include paths in the filesystem.

WebDAV

Web Distributed Authoring and Versioning (WebDAV) is an extension of the HTTP protocol. It consists of several new request methods that are added on top of HTTP to allow functionality, such as editing and managing files stored on WWW servers even from remote systems. If it is enabled by default on a website, WebDAV will allow anyone to enumerate files on the site, even with all directory indexes in place and directory listings turned off.

PROPFIND (an HTTP method used by WebDAV) is used to retrieve properties (stored as XML) from a resource, and also allows one to retrieve the collection structure (a.k.a. directory hierarchy) of a remote system. This can be used by attackers as shown below, using telnet to connect to www.example.com:

```
$ telnet example.com 8080
Trying 217.160.182.153...
Connected to example.com.
Escape character is '^]'.
PROPFIND / HTTP/1.1
Depth: 1
```

The above process will reveal the contents of the Web server root folder. Users browsing normally would get served index.html, but when using WebDAV, the attacker can reveal many secret files.

Verbose error messages

These are messages which are mainly responses to an invalid query. A prominent example is the error message associated with SQL queries. SQL injection attacks typically require the attacker to have prior knowledge of the structure or format used to create SQL queries on the site. The information leaked by a verbose error message can provide the attacker with crucial information on how to construct valid SQL queries for the backend database. (For information on SQL injection, refer to Part 1 of this series.) The following error message was returned by placing an apostrophe into the username field of a login page:

An Error Has Occurred. Error Message: System.Data.OleDb.OleDbException: Syntax error (mis
System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling (Int32 hr) at System
dbParams, Object& executeResult) at

In the first error statement, a syntax error is reported. The error message reveals the query parameters that are used in the SQL query: username and password. The leaked information here is the missing link for an attacker to begin to construct SQL injection attacks against the site.

A common error that we can see during our search is "HTTP 404 Not Found". Often, this error code provides useful details about the underlying Web server and associated components. For example:

Not Found

The requested URL /page.html was not found on this server.

Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g DAV/2 PHP/5.1.2 Server at

localhost Port 80

This error message can be generated by requesting a non-existent URL. After the common message that shows a page as not found, there is information about the Web server version, OS, modules and other products used. This information can be very useful to attackers.

Other methods include guessing the files with no direct link pointing to them. The best example of this is guessing the URL for the admin interface of the Web application — for example, guessing the URL www.example.com/wp-admin/ for any WordPress-based website is quite easy.

Time for security

1. Disable WebDAV in Apache if you don't need it. To do this, find the following entries inhttpd.conf and change On to Off.
   <IfDefine DAV>
   DAV On
   </IfDefine>
2. By default, /usr/local/httpd/htdocs is the only directory with the IfDefine DAV directive. Other directories that have this directive will also need to be changed. After this, restart Apache.
3. When dealing with information disclosures in HTML comment tags, it would not be appropriate to deny the entire request for a Web page due to comment tags. To handle this, there's a really cool feature in Apache 2.0, called filters (refer to official Apache documentation). The basic premise of filters is that they read from standard input and print to standard output. This feature becomes intriguing from a security perspective when dealing with this type of information disclosure prevention.

4. The application should never return verbose error messages or debug information to the user's browser. When an unexpected event occurs (such as an error in a database query, a failure to read a file from disk, or an exception in an external API call), the application should return the same generic message, informing the user that an error occurred.
5. In Apache, custom error pages can be configured using the ErrorDocument directive inhttpd.conf. For example: ErrorDocument 500 /generalerror.html. This will redirect users togeneralerror.html, which can be your own modified error page.
6. Disable or limit detailed error handling. In particular, do not display debug information, stack traces, or path information to end-users. Moreover, overriding the default error handler so that it always returns *'200' (OK)* error screens reduces the ability of automated scanning tools to determine if a serious error occurred.
7. Lastly, programmers need to be more aware about whether they are unintentionally providing any crucial information.

Source : http://www.opensourceforu.com/2011/03/securing-apache-part-7-fool-proofing-server-os/