Task Report for CloudSEK Security Challenge

Participant name: Prajwal Ghotage

Username: endless

E-mail: prajwalghotage2508@gmail.com

Category: Web

Challenge 1: Serialization Saga

Description:

This Capture The Flag (CTF) challenge is designed to assess your ability to identify and exploit fundamental insecure deserialization vulnerabilities. Can you successfully execute the necessary functions and retrieve the flags? Lesssgoo!

https://webctf.cloudsek.com/serialization-saga

Solution Methodology:

Clicking on the challenge URL we are greeted with a web page with the PHP source code of the challenge instance. Looking at the source code we can see that it accepts a "sess" URL parameter. The value is first decoded from base64 format and stored in the $data variable. It is then passed to the unserialize() function. This tells us that our payload should be a serialized PHP object in base64 encoding format.

Since the function we want has an func_no value of 3 and func_name value as GetMeDemFlagz, we use the values of 3, GetMeDemFlagz in the serialized object like so:

```
O:8:"CloudSEK":2:{s:7:"func_no";i:3;s:9:"func_name";s:13:"GetMeDemFlagz";}
```

This invokes the constructor and the __wakeup function for processing. However, this directly returns an Invalid object data error.

On a closer look at the code, the func_name parameter is being passed to the string_rot13() function. This means that the func_name to be passed **must initially be passed through rot13** so that when we invoke the constructor, the func_name is rotated again.

This plays to our advantage as the func_name then reverts back to the original func_name i.e., GetMeDemFlagz because it is rotated twice.

Thus, the final serialised payload looks like this:

```
O:8:"CloudSEK":2:{s:7:"func_no";i:3;s:9:"func_name";s:13:"TrgZrQrzSyntm";}
```

This creates an object of class CloudSEK, with values 3, TrgZrQrzSyntm. It is then processed in the __wakeup method. The respective function is then called giving us our flag.

FLAG: CSEK{PHP_0Bj3CT_D3$3R1L1Z@T10N}

Challenge 2: The SHA Juggler

Description:

Dive into the depths of "The SHA Juggler," a mysterious web challenge that tests your prowess in PHP type juggling, cunning encoding techniques,

and web exploitation. Your mission is to outwit the system, leveraging the peculiarities of PHP type comparisons, decipher the applied encodings,

and exploit vulnerabilities to retrieve the concealed flag. Can you navigate the enigmatic interplay of types and encodings and emerge victorious?

https://webctf.cloudsek.com/the-sha-juggler

Solution Methodology:

Upon visiting the challenge instance, we are greeted with a marquee text saying, "Do you know where to look?". Reading this, the first thing that comes to mind is to view page source. Looking at the HTML and JS, we find a JS constant defined and its value looks like a hex dump:

```
const isThisNormal = "50 44 39 77 61 .... 4b 50 7a 34 3d";
```

Decoding the value from hex we get a long base64 encoded string. Upon decoding this string from base64, we obtain the content of a PHP file named "you_found_me.PHP". The content is as follows:

```
<?PHP

// you_found_me.PHP

if (isset($_GET['hash'])) {

    if ($_GET['hash'] === "10932435112") {

        die('Do you think its that easy??');

    }

    $hash = sha1($_GET['hash']);

    $target = sha1(10932435112);

    if($hash == $target) {

        include('flag.PHP');

        print $flag;

    } else {

        print "CSEK{n0_4lag_4_u}";

    }

}

?>
```

We can see that it uses a URL parameter called 'hash'. There are two conditions, one of which is acting as the guarding clause for the file "flag.PHP".

Let us consider the first condition, which is using strict comparison with === operator. Since we know that global variables like $_GET, $_POST etc store values in the form of an array and in this case the array value is a string. Thus, the type check is sure to pass. This means that we cannot directly use the same value given as input i.e., 10932435112.

Basically, this means that we cannot use the value of 10932435112 in any form in the URL parameter or the code will exit.

Juggling the Types:

Consider the second condition. Before the condition we are hashing the input and the number 10932435112 using the sha1() PHP function. This function always returns the hash of the value as a string. The condition itself uses loose comparison operator (==). Here both variables are of string type. Hence, we need to send our input value such that both values will be equal.

We get the SHA-1 value for the $target as 0e07766915004133176347055865026311692244.

Now we need to choose our payload such that its hash will also start with "0e". The reasoning for selection of the payload is that it starts with "0e..", which is further explained below. After some research and googling I was able to find a GitHub repository with such payloads for different hashing techniques. There under the SHA-1 category, we use the value as hashcatU4BRJMv0wZQ9. The reason to choose this type of value has to do with the concept of **type juggling** in PHP which arises while using loose comparison. It means that if both the values to be compared are "numerical strings" or if one is a number and the other is a string **comparison is done numerically**. PHP also treats a string as a number if it begins with a number. This is some juicy information, and it is right out of the official PHP docs!!

And since both our values are not only numerical strings, but they also start with 0e, which indicates exponential notation. After e there could be any value and it would evaluate to 0 as 0 to the power of anything is 0. Hence the condition evaluates to true. This also means that we can use all those payloads that evaluate to a hash with a similar format i.e. "0e...".

Payload: hashcatU4BRJMv0wZQ9

We pass the payload as in the parameter, both conditions are bypassed and we are greeted with our flag. Finally, after all the "juggling"!!

FLAG: CSEK{typ3_juggl1ng_1n_PHP}

References:

- https://github.com/spaze/hashes/blob/master/sha1.md
- https://www.PHP.net/manual/en/language.operators.comparison.PHP
- https://security.stackexchange.com/a/268223