# 所代币代币_一个代币将其全部泄露：$ 8000 NPM_TOKEN的故事

翻译

weixin_26722031 于 2020-08-31 08:00:24 发布 77 收藏
文章标签： python
原文链接：https://medium.com/bugbountywriteup/one-token-to-leak-them-all-the-story-of-a-8000-npm-token-79b13af182a3
版权

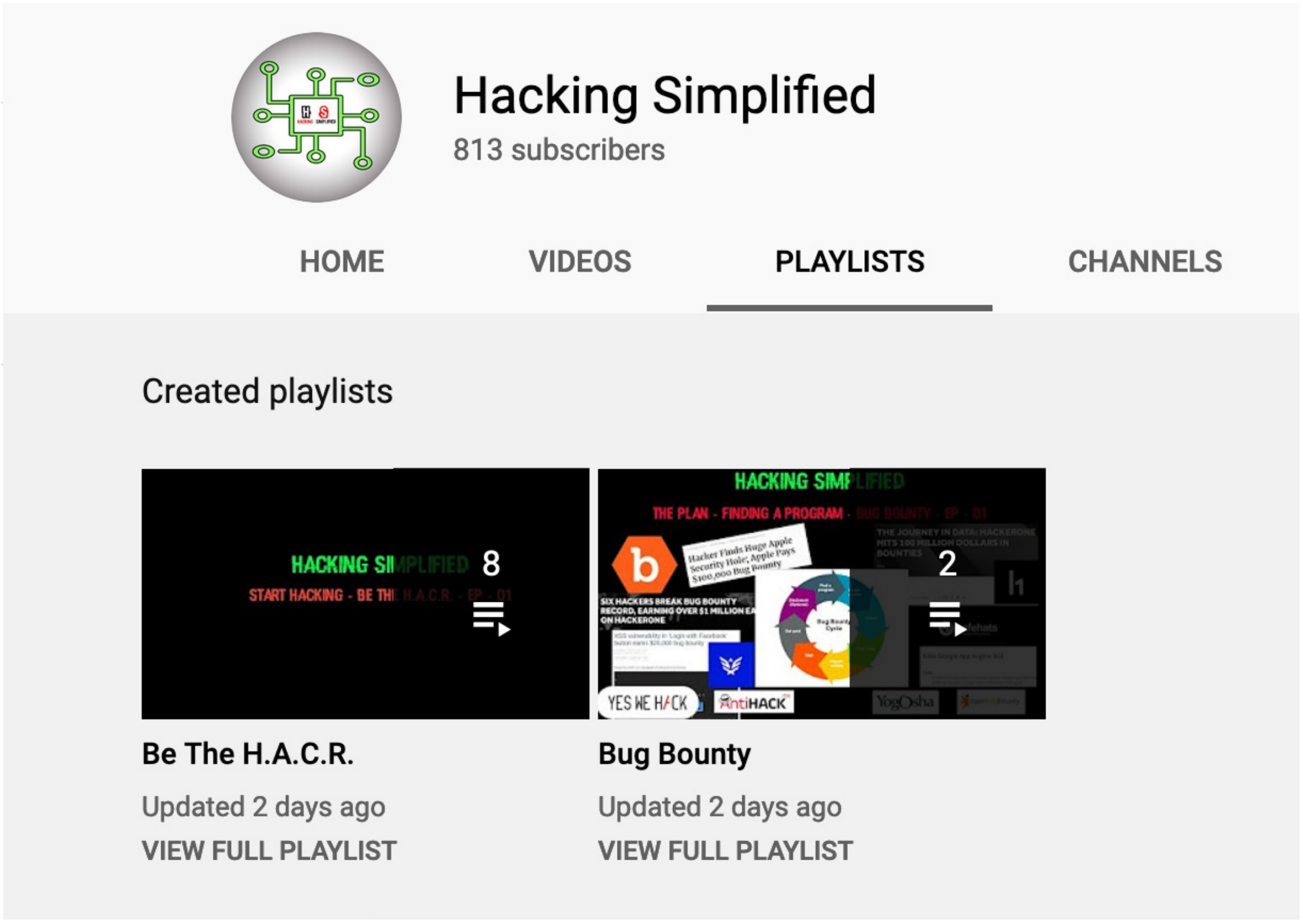
所代币代币

Not long ago, I started a youtube channel, **HackingSimplified**.

不久前，我开设了一个YouTube频道**HackingSimplified** 。

So *after a month of making videos on basics of web security attacks*, I started another series on the channel namely, **the bug bounty series**. Here, I am going to talk about the 7 stages of bug bounty and how to go about it. Since, I had started making videos on bug bounty so I thought to brush up my bug bounty skills and automation tools.

因此， *在制作了一个有关Web安全攻击基础知识的视频后* ，我在该频道上开始了另一个系列，即**漏洞赏金系列** 。 在这里，我将讨论漏洞赏金的7个阶段以及如何解决。 从那以后，我开始制作有关漏洞赏金的视频，因此我想提高我的漏洞赏金技巧和自动化工具。



The two playlists

这两个播放列表

The last time I had browsed through hackerone to do some bug bounty was some time back in Jan 2019. But in the past one month I had been reading a lot of reports, and these reports kindled my interest again. I started looking at the hackerone program directory and my private invites too for a program that satisfied all the conditions I mentioned in my last video : "Scope Review and Bug Hunting Using Github Dorks -Bug Bounty -Ep -02" which are :

我上次浏览hackerone进行漏洞赏金的时间是在2019年1月。但是在过去的一个月中，我一直在阅读很多报告，这些报告再次激发了我的兴趣。 我开始查看hackerone程序目录，并且我也私下邀请一个程序，该程序满足我在上一个视频中提到的所有条件：" 使用Github Dorks -Bug Bounty -Ep -02进行范围检查和错误查找 "，它们是：

1. No. of reports resolved
   已解决的报告数
2. Assets
   资产
3. Payout
   付款方式
4. Response efficiency
   响应效率
5. Time to triage and Time to bounty ( personal choice )
   分诊时间和赏金时间(个人选择)

This program happened to have all the stars aligned :)

这个程序碰巧使所有星星对齐:)

1. No. of reports resolved — Above 550
   已解决的报告数量-550以上
2. Assets — All subdomains
   资产-所有子域
3. Payout — Crtical was $1000-$4000 and lowest was from $50-$200

支出-Crtical是$ 1000- $ 4000，最低是$ 50- $ 200

4. Response Efficiency — 90%
   响应效率— 90％

5. Time to triage — 2days and Time to bounty — 10days
   分流时间-2天，赏金时间-10天

Since recently I had made a video on template injection so that was my main area of focus. I entered `{{7*7}}`and other payload, which was relevant to the templating engine being used on almost every input field. This didn't work though.

从最近开始，我制作了有关模板注入的视频，因此这是我的主要关注领域。 我输入了`{{7*7}}`和其他有效负载，这些负载与几乎在每个输入字段上使用的模板引擎有关。 不过这没有用。

Then I tried looking for IDORs and improper access control check using autorize, didn't have much luck there either.

然后，我尝试使用autorize寻找IDOR和不当的访问控制检查，也没有太多的运气。

I started looking at JS files for other endpoints and some secrets getting leaked. Downloaded all the JS files first and then started grepping into it for secrets and url endpoints.

我开始查看其他端点的JS文件以及一些机密信息。 首先下载所有JS文件，然后开始对其进行加密以获取秘密和url端点。

To download all JS files :

要下载所有JS文件：

1. If you've BURP Suite pro you could straightaway extract all scripts to one file : But this extracts all file into one file with the js url in it, which might be good in most use cases but I also wanted them in their individual files.
   如果您具有BURP Suite专业版，则可以直接将所有脚本提取到一个文件中：但这会将所有文件提取到一个包含js url的文件中，这在大多数情况下可能是不错的选择，但我也希望将它们放在单独的文件中。

2. So, I first extracted all JS urls and then wrote a small bash script to get all those files with their respective names :
   因此，我首先提取了所有JS URL，然后编写了一个小的bash脚本来获取所有具有各自名称的文件：

```
cat urls.txt | xargs -I{} wget "{}"# Assuming urls are clean i.e. they don't have any extra parameters in t
# if the url is like this : https://storage.googleapis.com/workbox-cdn/releases/5.1.2/workbox-cacheable-res
# Then you need to cut the part after '?' like the followingcat urls.txt | cut -d"?" -f1 | xargs -I{} wget
```

Tomnomnom's `gf` tool came handy here.Using `gf urls` to get url endpoints, I found private IP getting leaked : `http://172.x.x.x` . I tried looking near the place this was getting leaked and found there a `NPM_TOKEN` value. Immediately started looking for ways to use that. I hadn't used `npm` much, only knew that it was `node package manager` . Had used it once, earlier while developing a `VueJS` application.

Tomnomnom的`gf`工具在这里很方便，使用`gf urls`获取URL端点，我发现私有IP泄漏了：`http://172.xxx`：`http://172.xxx` 。 我试着靠近这处泄漏的地方，发现那里有一个`NPM_TOKEN`值。 立即开始寻找使用它的方法。 我没用过`npm` ，只知道它是`node package manager` 。 在开发`VueJS`应用程序时曾经使用过一次。

After researching for sometime I learnt the following :1. CI i.e. Continuous Integration systems such as Jenkins pipelines or Travis CI etc use these tokens to build and deploy a webapp in an automated fashion. This token helps them to get access to npm private repository.

经过一段时间的研究，我学到了以下内容：1。 CI，即Jenkins管道或Travis CI等持续集成系统，使用这些令牌以自动化方式构建和部署Web应用程序。 该令牌可帮助他们访问npm私有存储库。

2. Different types of tokens such as —Read and publish only, Readonly, CIDR whitelisted i.e. tokens which can be used from a specified IP address range only

2. 不同类型的令牌，例如-只读和发布，只读，CIDR列入白名单，即只能在指定的IP地址范围内使用的令牌

3. How to use npm tokens : So your npm tokens should be in the following format in `.npmrc` file —

3. 如何使用npm令牌 ： `.npmrc`文件中的npm令牌应采用以下格式-

```
registry=https://registry_link_here
//registry_link_here/:_authToken=auth_token_here
```

And a few more…

还有一些…

I tried accessing the npm registry using this token in my `.npmrc` file like this :

我试图在我的`.npmrc`文件中使用此令牌访问npm注册表，如下所示：

```
registry=https://registry.npmjs.org
//registry.npmjs.org/:_authToken=auth_token_here
```

But in vain. I couldn't get reply of `npm whoami`, which I should've if the token was valid. Some articles also suggested that you could also keep the `NPM_TOKEN` value in encrypted form in `.npmrc` . So, I concluded that this must be an encrypted token. This was Wednesday — Day 1 of hacking on this target.

但是徒劳。 我无法收到`npm whoami`回复，如果令牌有效，则应该回复。 有些文章还建议，你也可以保持`NPM_TOKEN`值以加密的形式在`.npmrc` 。 因此，我得出结论，这必须是加密令牌。 这是星期三-攻击该目标的第一天。

On the next day after office work, in the evening around 8pm I started looking into the program again. Earlier in January this year I was reading about CSWH i.e. Cross Site Websocket Hijacking ( however this might be unexploitable in sometime now, see this ) and this program was using websockets. So, I started looking into it.

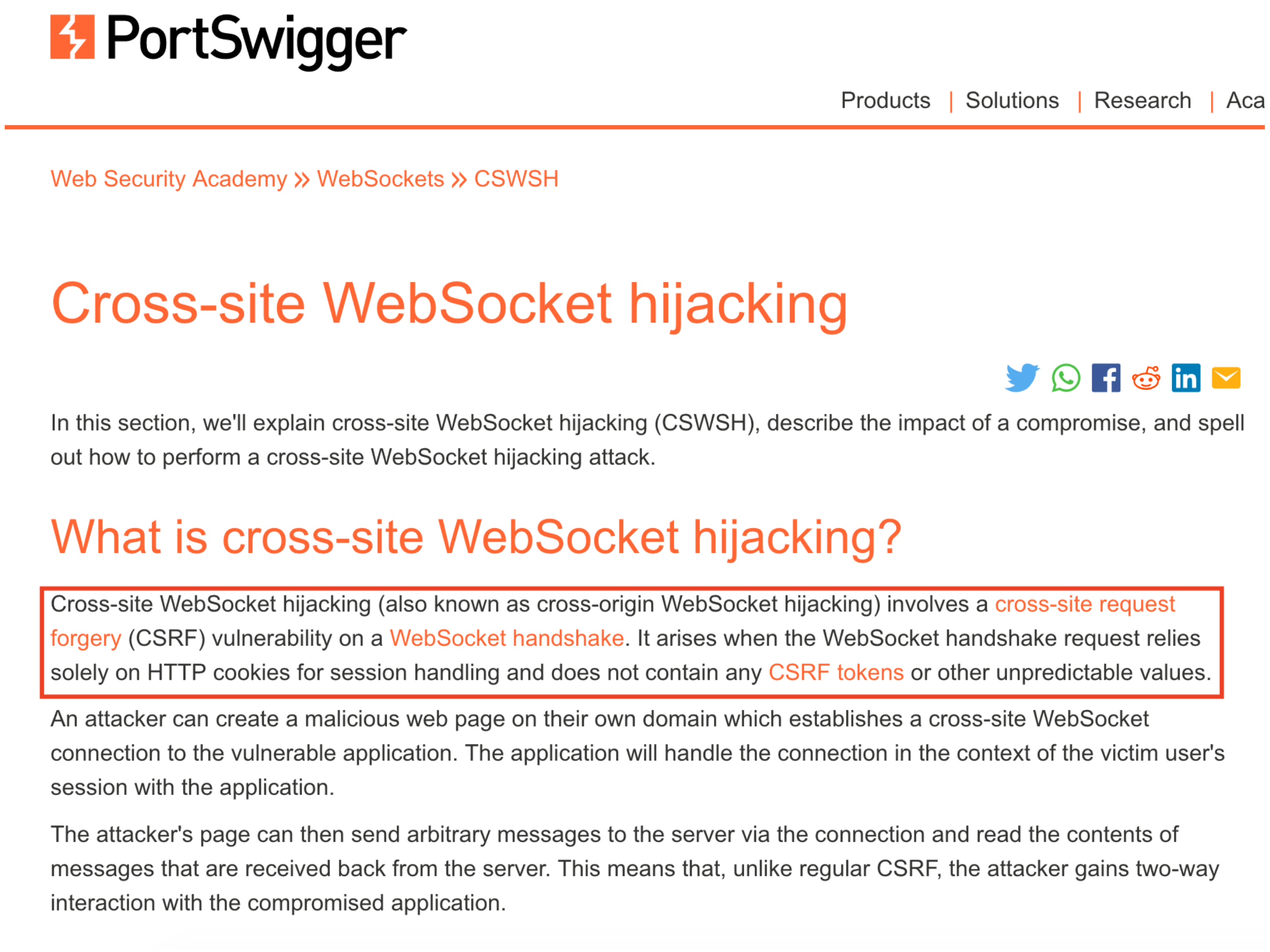
在办公室工作的第二天，晚上8点左右，我再次开始研究该程序。 今年一月初，我读到了有关CSWH的文章，即跨站点Websocket劫持 (但是现在可能无法利用，请参阅此信息 )，并且该程序正在使用websockets。 因此，我开始研究它。

Conditions for CSWH is that the websockets should be only communicating using cookies, like any other CSRF attack.

CSWH的条件是，网络套接字只能像其他CSRF攻击一样使用cookie进行通信。

I practiced onto BURP suite labs to refresh what I had learnt 6 months back. This website also had similar conditions and only cookie was required to get the websocket communication up and running.

我练习了BURP套件实验室，以刷新6个月前学到的知识。 这个网站也有类似的条件，只需要cookie就可以启动和运行websocket通信。


PortSwigger

Products | Solutions | Research | Aca

Web Security Academy » WebSockets » CSWSH

Cross-site WebSocket hijacking

In this section, we'll explain cross-site WebSocket hijacking (CSWSH), describe the impact of a compromise, and spell out how to perform a cross-site WebSocket hijacking attack.

What is cross-site WebSocket hijacking?

Cross-site WebSocket hijacking (also known as cross-origin WebSocket hijacking) involves a cross-site request forgery (CSRF) vulnerability on a WebSocket handshake. It arises when the WebSocket handshake request relies solely on HTTP cookies for session handling and does not contain any CSRF tokens or other unpredictable values.

An attacker can create a malicious web page on their own domain which establishes a cross-site WebSocket connection to the vulnerable application. The application will handle the connection in the context of the victim user's session with the application.

The attacker's page can then send arbitrary messages to the server via the connection and read the contents of messages that are received back from the server. This means that, unlike regular CSRF, the attacker gains two-way interaction with the compromised application.

CSRF but on weboscket requests

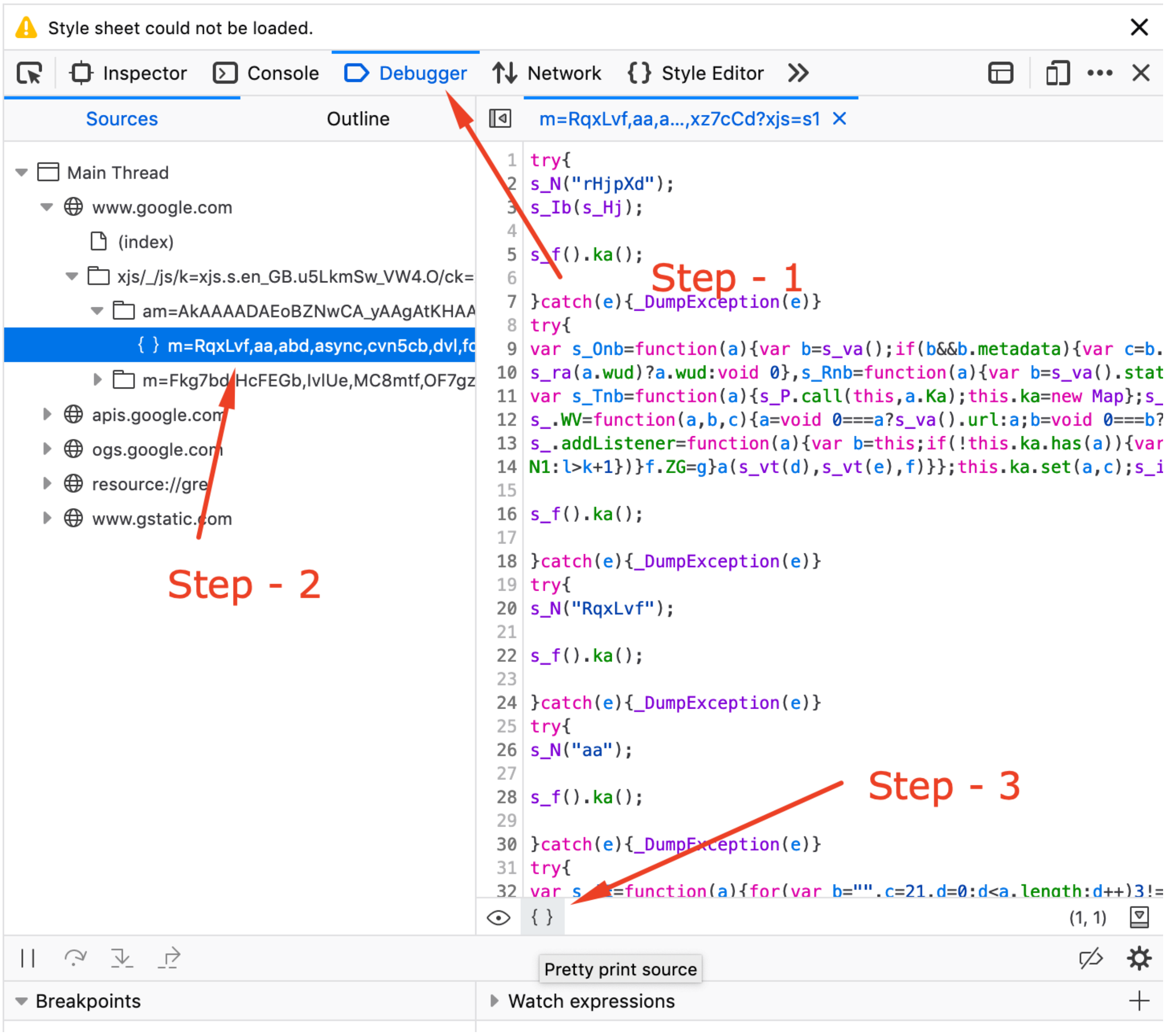
CSRF，但根据weboscket请求

So I tried leaking the websocket messages but couldn't get that. On closer inspection I found that it was also using a `nonce` and that nonce was given by the server to the client in a different request. So basically that acted as a CSRF token, which is usually used to thwart CSRF attacks. However I found some other bugs in their CSRF usage while looking for CSWH. Reported these on the same day. More on that in another post.

因此，我尝试泄漏websocket消息，但无法获取。 在仔细检查后，我发现它也在使用`nonce`并且该随机数是服务器在另一个请求中向客户端提供的。 因此，基本上，它充当CSRF令牌，通常用于阻止CSRF攻击。 但是，我在寻找CSWH时发现了CSRF使用中的其他错误。 在同一天报告了这些。 有关更多内容，请参见另一篇文章。

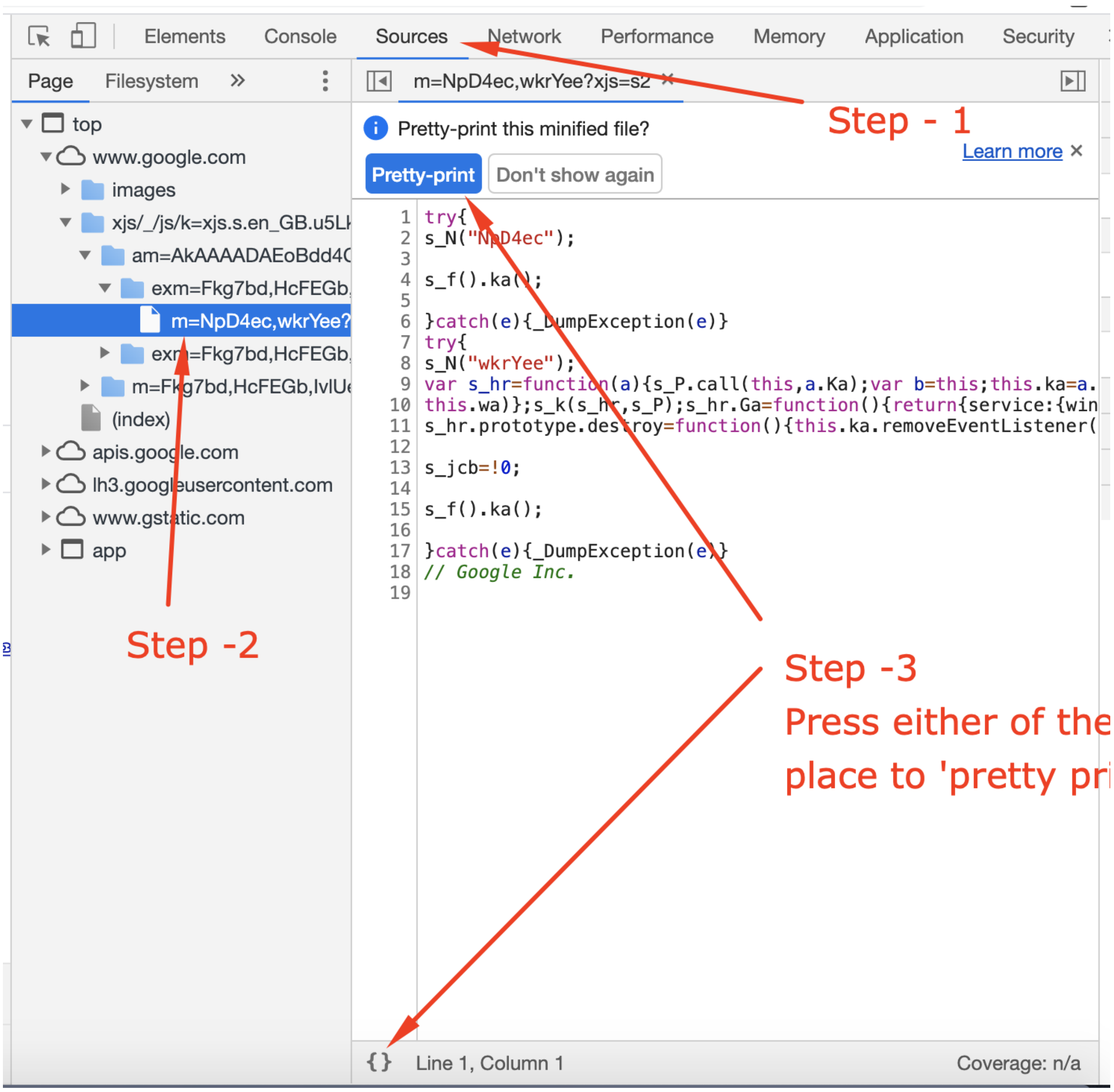
Friday — Day 3 — After spending around two hours and not finding anything I thought of looking back at the `NPM_TOKEN` again. I found the JS file and *un-uglified* it in the browser in the :

星期五-第3天-在花费了大约两个小时之后，没有找到任何东西，我想到了再次回顾`NPM_TOKEN` 。 我找到了JS文件，并在浏览器的以下位置*取消了丑化*：



```
try{
s_N("rHjpXd");
s_Ib(s_Hj);

s_f().ka();

}catch(e){_DumpException(e)}
try{
var s_Onb=function(a){var b=s_va();if(b&&b.metadata){var c=b.
s_ra(a.wud)?a.wud:void 0},s_Rnb=function(a){var b=s_va().stat
var s_Tnb=function(a){s_P.call(this,a.Ka);this.ka=new Map};s_
s_.WV=function(a,b,c){a=void 0===a?s_va().url:a;b=void 0===b?
s_.addListener=function(a){var b=this;if(!this.ka.has(a)){var
N1:l>k+1})}f.ZG=g}a(s_vt(d),s_vt(e),f)}};this.ka.set(a,c);s_i

s_f().ka();

}catch(e){_DumpException(e)}
try{
s_N("RqxLvf");

s_f().ka();

}catch(e){_DumpException(e)}
try{
s_N("aa");

s_f().ka();

}catch(e){_DumpException(e)}
try{
var s_[illegible]=function(a){for(var b="",c=21,d=0;d<a.length;d++)3!=
```

Firefox pretty print
Firefox漂亮的打印



```
try{
s_N("NpD4ec");

s_f().ka();

}catch(e){_DumpException(e)}
try{
s_N("wkrYee");
var s_hr=function(a){s_P.call(this,a.Ka);var b=this;this.ka=a.
this.wa)};s_k(s_hr,s_P);s_hr.Ga=function(){return{service:{win
s_hr.prototype.destroy=function(){this.ka.removeEventListener(

s_jcb=!0;

s_f().ka();

}catch(e){_DumpException(e)}
// Google Inc.
```

Chromium based browsers Pretty Print

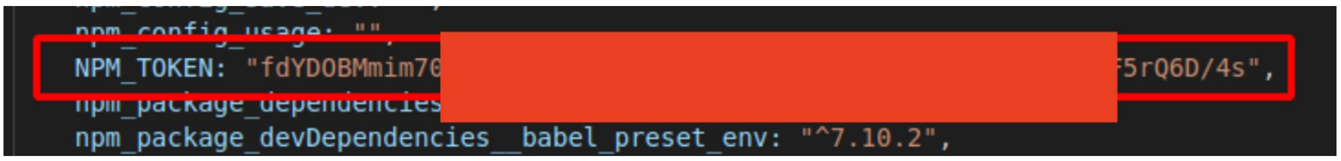
基于Chromium的浏览器Pretty Print

You could do it online as well : https://unminify.com/

您也可以在线进行： https ： //unminify.com/

There were around 17k lines but since I had some experience looking at these so I knew most of these were webpack generated code. So I skimmed through the code to find something interesting, mostly near where I found the `NPM_TOKEN` value.

大约有17,000行，但是由于我有一些经验，因此我知道其中大多数都是webpack生成的代码。 因此，我浏览了一下代码以查找一些有趣的东西，主要是在找到`NPM_TOKEN`值的位置附近。



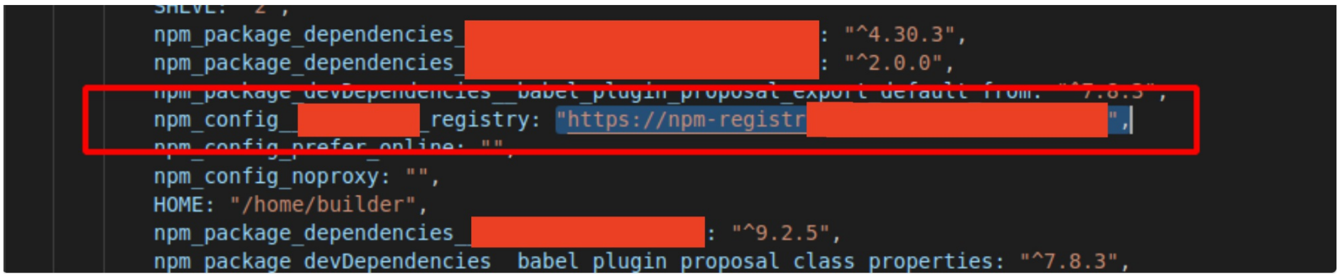
```
npm_config_usage: "",
NPM_TOKEN: "fdYDOBMmim70[redacted]5rQ6D/4s",
npm_package_dependencies
npm_package_devDependencies__babel_preset_env: "^7.10.2",
```

NPM_TOKEN

NPM_TOKEN

There I found a private registry link, then I realized, why didn't I think of this □.

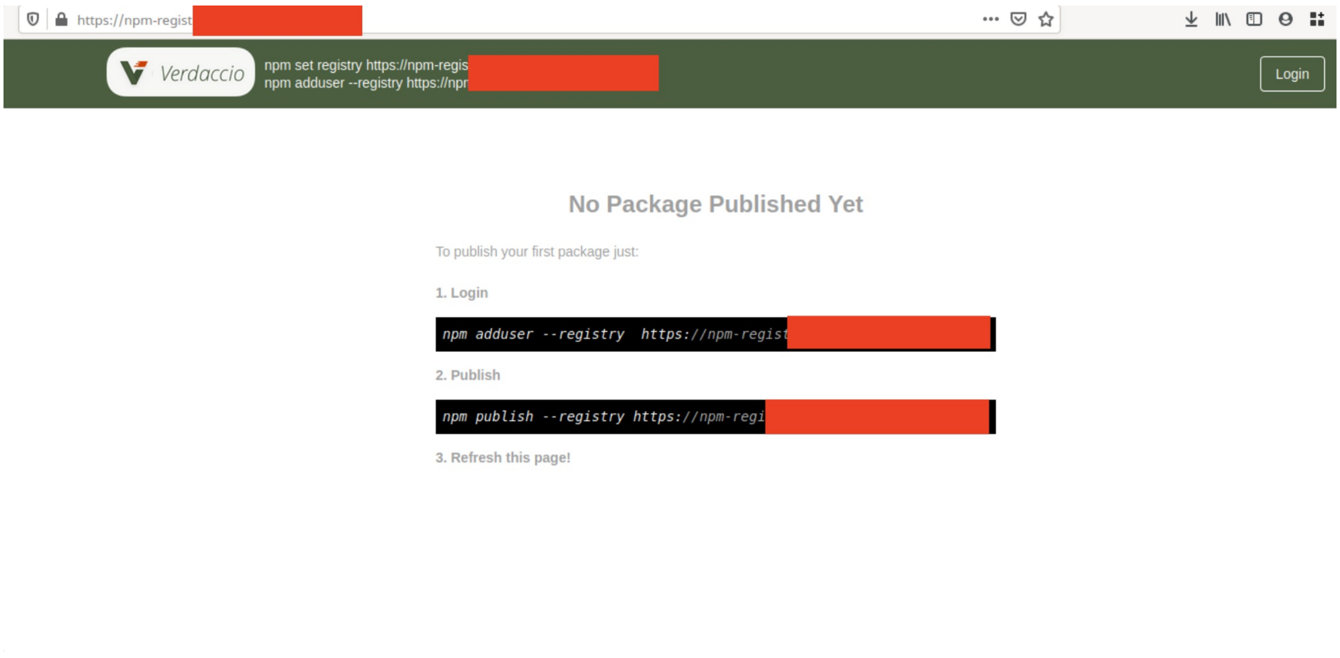
在那里，我找到了一个私人注册表链接，然后我意识到，为什么我没有想到这个□。



```
SHLVL: "2",
npm_package_dependencies_[redacted]: "^4.30.3",
npm_package_dependencies_: "^2.0.0",
npm_package_devDependencies__babel_plugin_proposal_export_default_from: "^7.8.3",
npm_config_[redacted]_registry: "https://npm-registr[redacted]",
npm_config_prefer_online: "",
npm_config_noproxy: "",
HOME: "/home/builder",
npm_package_dependencies_[redacted]: "^9.2.5",
npm_package_devDependencies__babel_plugin_proposal_class_properties: "^7.8.3",
```

Private Repo Link

私人回购链接



https://npm-regist[redacted]

npm set registry https://npm-regis[redacted]
npm adduser --registry https://npr

No Package Published Yet

To publish your first package just:

1. Login

```
npm adduser --registry  https://npm-regist[redacted]
```

2. Publish

```
npm publish --registry https://npm-regi[redacted]
```

3. Refresh this page!

Private Registry's Frontpage

私人注册表的首页

I quickly replace the value of registry in the `.npmrc` file to this :

我Swift将`.npmrc`文件中注册表的值替换为：

```
registry=https://private_registry_link_here
//private_registry_link_here/:_authToken=auth_token_here
```

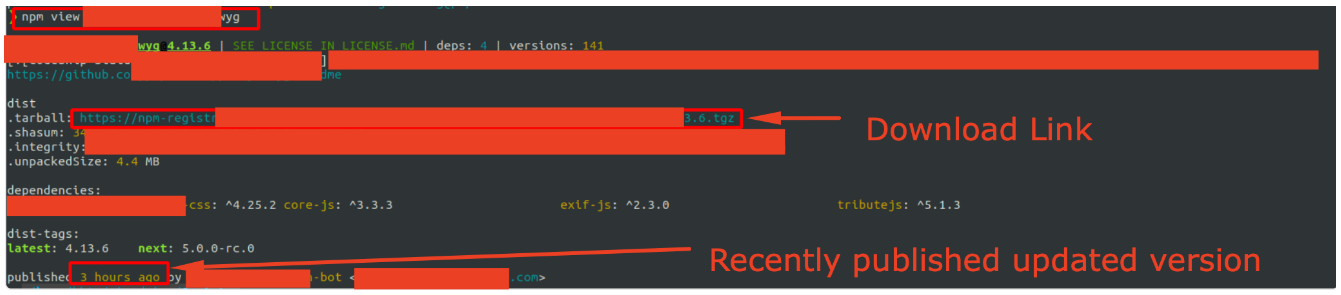
And now to my command of `npm whoami` I got a reply : `srv-npm-registry-ci`

现在我对`npm whoami`命令得到了回复： `srv-npm-registry-ci`

Then I tried fetching a list of all the packages in the private npm registry but that's not possible i.e. you can't just list all the packages on the private registy. Atleast I couldn't find a way, let me know if it's possible. So, I tried retrieving the private code of the company using the following command :

然后，我尝试获取私有npm注册表中所有软件包的列表，但这是不可能的，即您不能只列出私有域上的所有软件包。 至少我找不到办法，请告诉我是否有可能。 因此，我尝试使用以下命令检索公司的私人代码：

```
npm view private_repo
npm get private_repo
```



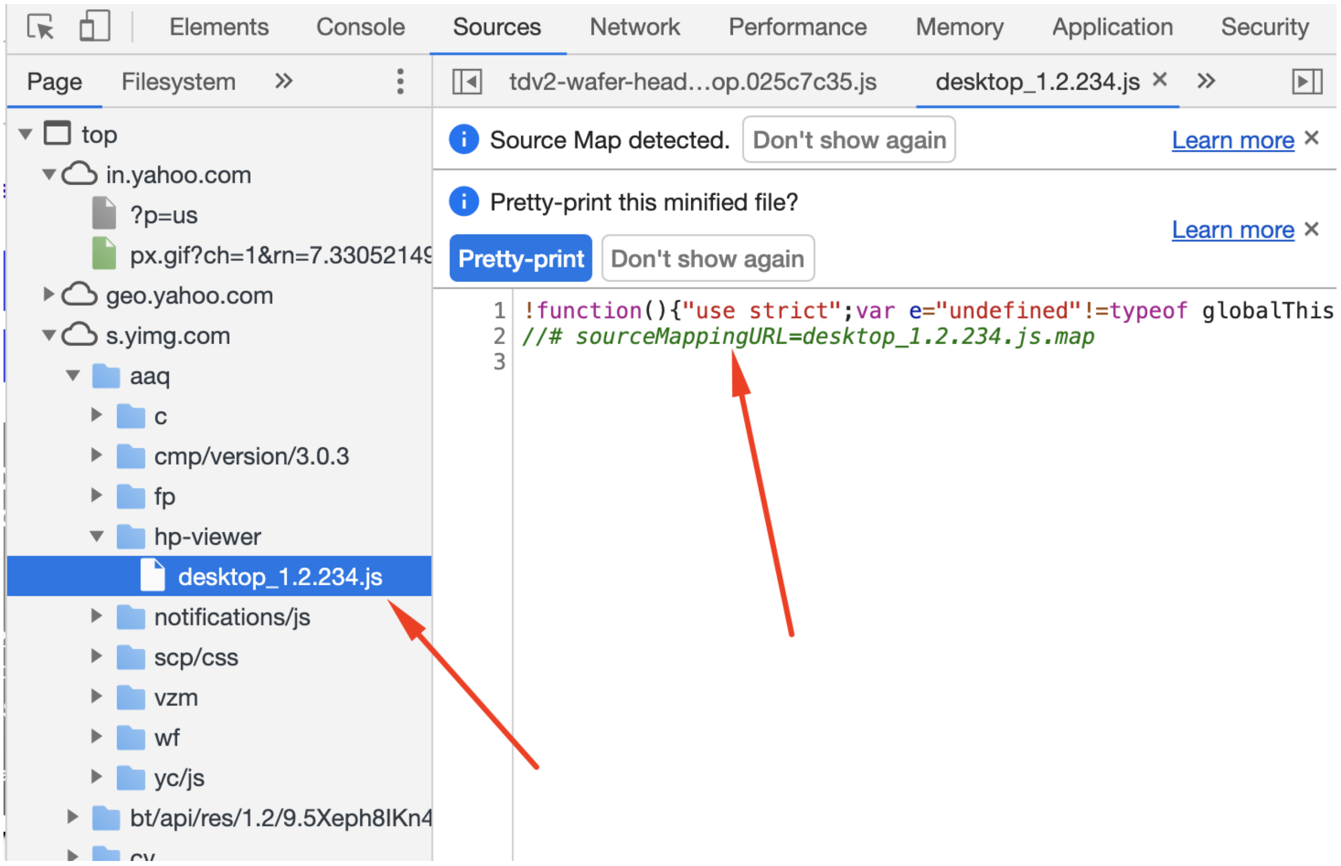
```
npm view [redacted]wyg

[redacted]wyg@4.13.6 | SEE LICENSE IN LICENSE.md | deps: 4 | versions: 141
[redacted]] [redacted]
https://github.co[redacted]dme

dist
.tarball: https://npm-registr[redacted]3.6.tgz
.shasum: 34[redacted]
.integrity: [redacted]
.unpackedSize: 4.4 MB

dependencies:
[redacted]css: ^4.25.2 core-js: ^3.3.3          exif-js: ^2.3.0          tributejs: ^5.1.3

dist-tags:
latest: 4.13.6    next: 5.0.0-rc.0

publishec 3 hours ago by [redacted]-bot <[redacted].com>
```

Output of npm view command

npm view命令输出

Now, you might ask as to how I got to know the names of private repos. Well the repos were compiled into one js file which if the source map is available then any browser can easily decouple them in the sources tab in inspect element like this :

现在，您可能会问我如何知道私有存储库的名称。 好吧，回购协议被编译到一个js文件中，如果源映射可用，那么任何浏览器都可以轻松地在inspect元素的source标签中将它们解耦，如下所示：



```
!function(){"use strict";var e="undefined"!=typeof globalThis
//# sourceMappingURL=desktop_1.2.234.js.map
```

If source map available — Then breakdown into separate components is possible

如果有源映射可用，则可以分解为单独的组件

So from there I got some unminified source code of the private js files and in those were some other included which got me access to those and so on, I could've downloaded almost all of their source code.

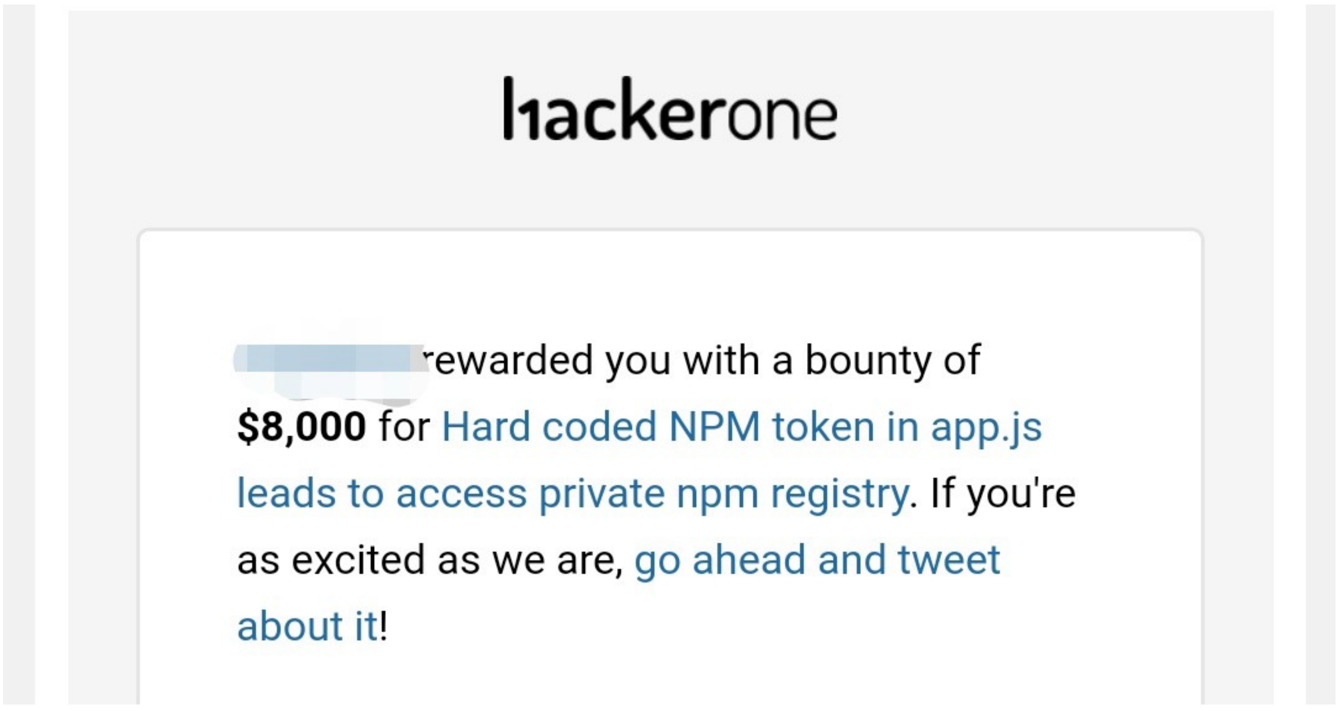
因此，从那里我获得了私人js文件的一些未压缩的源代码，并且其中包括一些其他文件，这些文件使我可以访问这些文件，依此类推，我几乎可以下载其所有源代码。

I didn't check whether the key had publish access or not as it would require me to publish a package on their private registry, which I thought wouldn't be wise. Moreover it shouldn't have publish rights as it was a CI key but you never know, people don't generally follow the practice of least privilege.

我没有检查密钥是否具有发布访问权限，因为这将要求我在其私有注册表中发布程序包，我认为这样做不明智。 此外，它不应该具有发布权限，因为它是CI密钥，但您永远不知道，人们通常不会遵循最小特权的做法。

The report was triaged in 17 hours and rewarded in another 7 days. The company handled it very professionally. I've asked them as to how these got leaked in the js file, still waiting on their reply.

该报告在17小时内进行了分类，并在另外7天内得到了回报。 该公司非常专业地处理它。 我问他们关于这些文件如何泄漏到js文件中的问题，仍在等待他们的答复。


hackerone

[redacted] rewarded you with a bounty of **$8,000** for Hard coded NPM token in app.js leads to access private npm registry. If you're as excited as we are, go ahead and tweet about it!

Highest bounty was $4000 but they awarded a bonus for this
最高奖金为4000美元，但他们为此奖励了奖金

## 重要要点： (Key Takeaways :)

1. Look for secrets in JS files — Try building automation around it
   在JS文件中寻找秘密-尝试围绕它构建自动化

2. Learn using your browser's dev tools : They alone will help in a lot of ways
   学习使用浏览器的开发工具：它们本身将在很多方面提供帮助

3. Stay updated with the tools — Even if you aren't doing bug bounties, might help in some of your other work.
   随时了解这些工具-即使您没有获得赏金，也可能对您的其他工作有所帮助。

4. Persistence is the key :)
   坚持是关键:)

*P.S.*

*聚苯乙烯*

There's a video writeup which talks about this writeup alongwith some BURP automation to find these secret tokens too here : https://youtu.be/9LBl-uFiYUE

有一个视频文章，讨论该文章以及一些BURP自动化，也可以在这里找到这些秘密令牌： https : //youtu.be/9LBl-uFiYUE

I started to find something else, template injection, and ended up finding 6 other bugs out of which 3 have been duplicate. All in all it had been a wonderful learning experience regarding websockets, will soon post a writeup of that too.

我开始寻找其他东西，模板注入，最后发现了6个其他错误，其中3个已经重复。 总而言之，这对于websockets来说是一次很棒的学习经历，很快也会发布有关它的文章。

*Hope this was worth your time, do checkout my youtube channel : HackingSimplified , I post* **videos every weekend**.

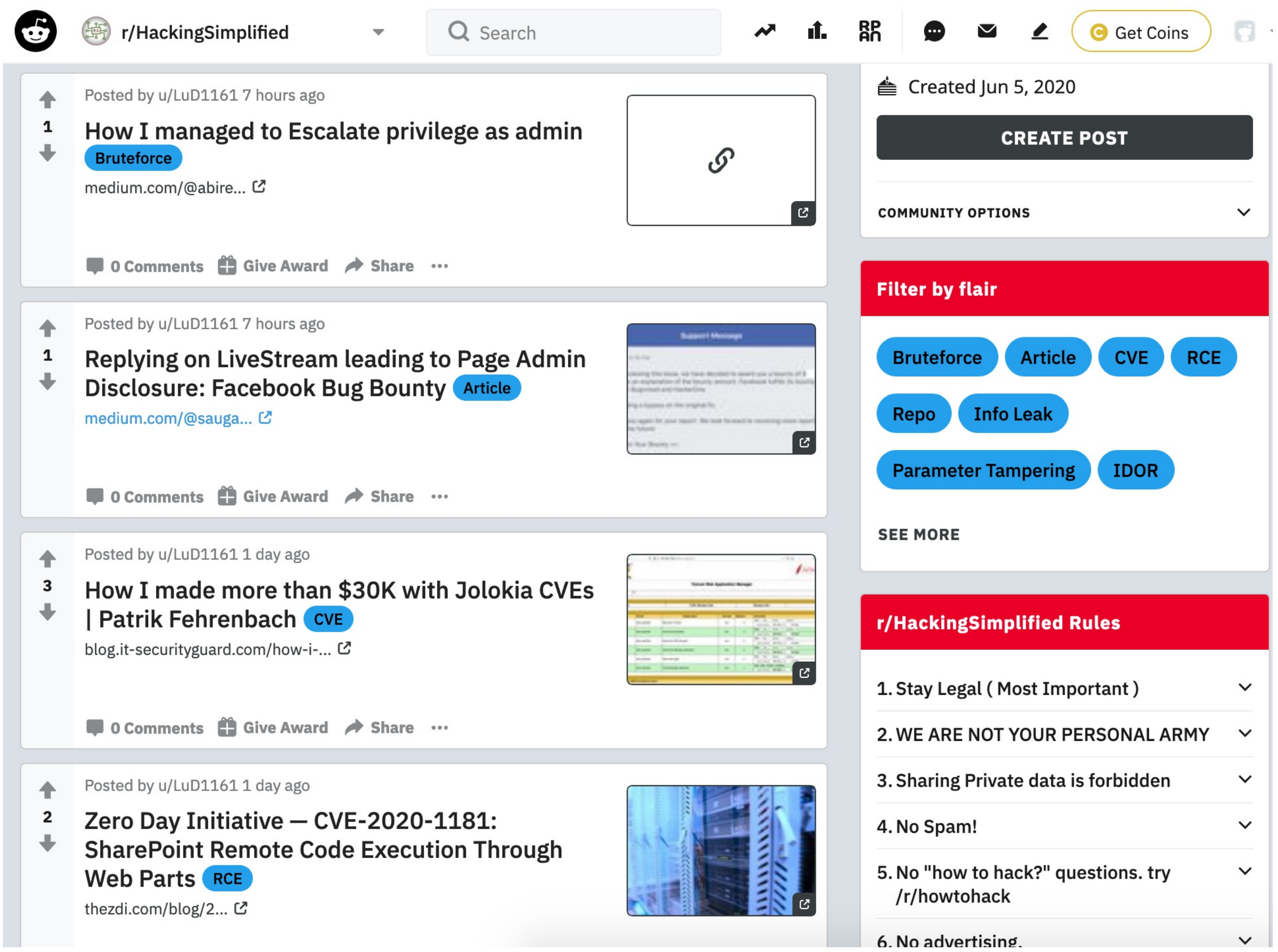
*希望这值得您花时间，请查看我的YouTube频道：HackingSimplified，我* **每个周末都** *发布* **视频** 。

YouTube channel : HackingSimplified

YouTube频道： HackingSimplified

Join the community, share, discuss, learn and grow. I post 3–4 article related to bug bounty and general cybersecurity daily here.

加入社区，分享，讨论，学习和成长。 我每天在这里发布3-4条有关漏洞赏金和一般网络安全的文章。


r/HackingSimplified

Posted by u/LuD1161 7 hours ago
How I managed to Escalate privilege as admin — Bruteforce
medium.com/@abire...
0 Comments · Give Award · Share

Posted by u/LuD1161 7 hours ago
Replying on LiveStream leading to Page Admin Disclosure: Facebook Bug Bounty — Article
medium.com/@sauga...
0 Comments · Give Award · Share

Posted by u/LuD1161 1 day ago
How I made more than $30K with Jolokia CVEs | Patrik Fehrenbach — CVE
blog.it-securityguard.com/how-i-...
0 Comments · Give Award · Share

Posted by u/LuD1161 1 day ago
Zero Day Initiative — CVE-2020-1181: SharePoint Remote Code Execution Through Web Parts — RCE
thezdi.com/blog/2...

Created Jun 5, 2020

Filter by flair: Bruteforce, Article, CVE, RCE, Repo, Info Leak, Parameter Tampering, IDOR

r/HackingSimplified Rules
1. Stay Legal ( Most Important )
2. WE ARE NOT YOUR PERSONAL ARMY
3. Sharing Private data is forbidden
4. No Spam!
5. No "how to hack?" questions. try /r/howtohack
6. No advertising.

HackingSimplified Subreddit

HackingSimplified Subreddit

Join the subreddit here : HackingSimplified

在此处加入subreddit： HackingSimplified

Telegram here : HackingSimplified

这里的电报： HackingSimplified

Twitter : @AseemShrey

推特： @AseemShrey

Thanks for reading :)

谢谢阅读 ：)

## 更新： (Update :)

I got a nice suggestion from @darthvader_htb about downloading js files.You could use another of tomnomnom's tool to do this: https://github.com/tomnomnom/fff

我从@darthvader_htb得到了一个关于下载js文件的很好的建议。 您可以使用tomnomnom的另一个工具来执行此操作： https : //github.com/tomnomnom/fff

```
cat urls.txt | fff
```

I don't suggest installing a plethora of tools and not using the built in tools. However, usage may vary.

我建议您不要安装过多的工具，也不要使用内置工具。 但是，用法可能会有所不同。

> 翻译自: https://medium.com/bugbountywriteup/one-token-to-leak-them-all-the-story-of-a-8000-npm-token-79b13af182a3

所代币代币