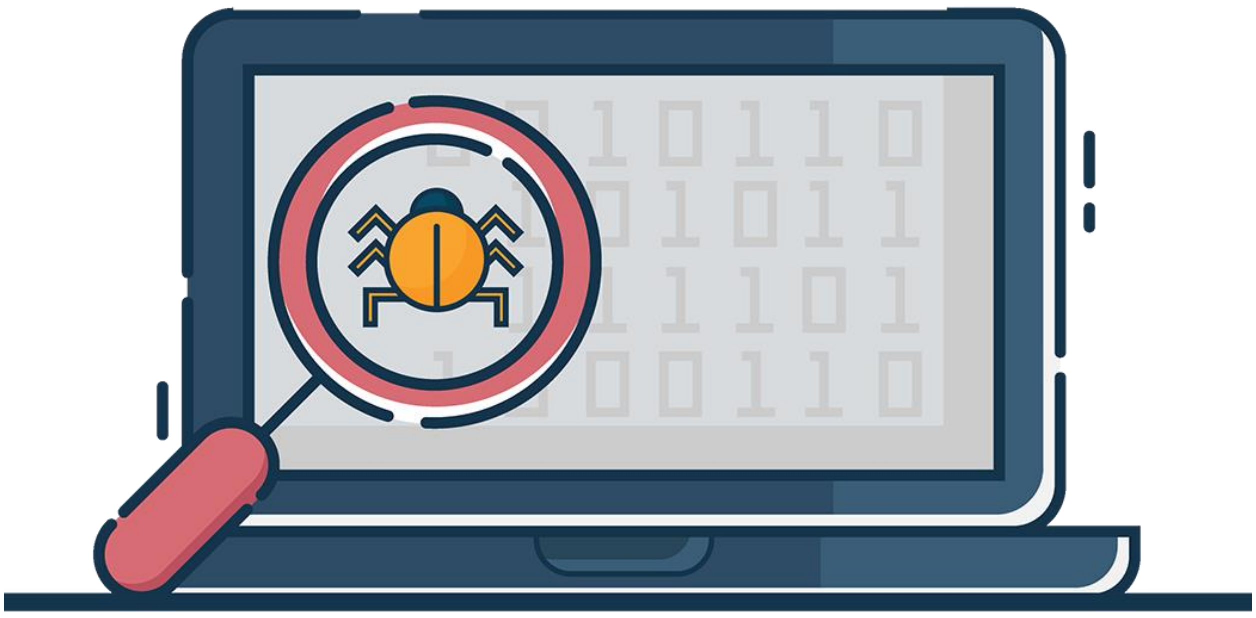
Beginners Guide: How to Use Javascript In BugBounty.

Kathan Patel

Jun 30 2020·



Let us take a look at what JavaScript is and why dev’s use them in a web app, before looking into how we can use them to find bugs.

What is JavaScript And Why It Is Used?

JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Where HTML and CSS are languages that give structure and style to web pages, **JavaScript** gives web pages interactive elements that engage a user.

In General, you can say JS is important to make a website more interactive for Users and make a web app dynamic.

Now, let’s see how we can use it for finding bugs while hunting on a target program, and side by side we will try to automate it with the help of some tools.

Things To Do.

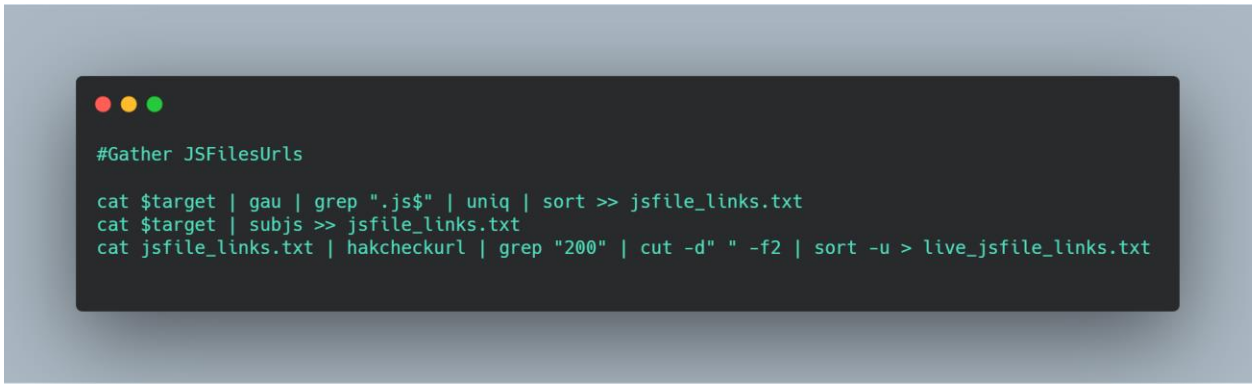
1. Gather JSFile links from the target.
2. Finding Endpoints from the JSFiles.
3. Finding Secrets in the JSFiles.
4. Get JSFiles locally for manual analysis.

We will be building our automation script based on this checklist.

1. Gathering JSFile links from a target.

There are a bunch of tools out there that can help you with this part. Some of them are ***subjs*** (https://github.com/lc/subjs), ***getJS*** (https://github.com/003random/getJS), ***gau*** (https://github.com/lc/gau).

I prefer ***subjs*** and ***gau*** both in combo made by the same person Corben Leo (https://github.com/lc). So, our code will look like this.



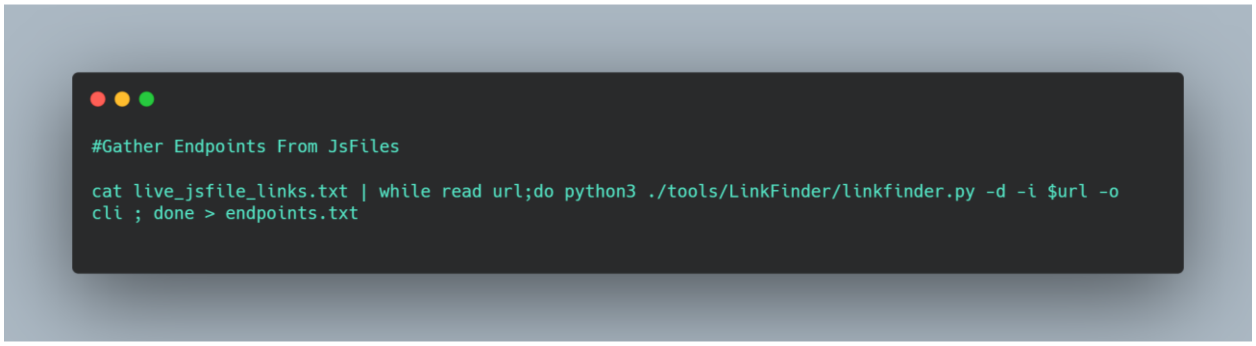
```
#Gather JSFilesUrls

cat $target | gau | grep ".js$" | uniq | sort >> jsfile_links.txt
cat $target | subjs >> jsfile_links.txt
cat jsfile_links.txt | hakcheckurl | grep "200" | cut -d" " -f2 | sort -u > live_jsfile_links.txt
```

Here, **$target** is our file containing target domains or subdomains. Using ***gau*** we gathered links of all the domains and grep them out for just JS file using regex **".js$".** Then we used subjs to actively gather JS files from domains. Now there might be a question of "Why we used these two tools here?", Answer is both tools work differently ***gau*** can fetch links from a different third-party source passively and ***subjs*** fetches JSfile link actively from the target. After running this code, we will have two files

- `jsfile_links.txt -` Which will contain gathered jsfilelinks
- `live_jsfile_links.txt -` Which will contain live links filtered using hakcheckurls.

2. Finding Endpoints in JSFiles

Once you have a bunch of JSfile links now is the time to get started for real, so the first thing we can do is gather some endpoint which we can accomplish using the tool ***LinkFinder*** (https://github.com/GerbenJavado/LinkFinder).



```
#Gather Endpoints From JsFiles

cat live_jsfile_links.txt | while read url;do python3 ./tools/LinkFinder/linkfinder.py -d -i $url -o
cli ; done > endpoints.txt
```

Here, we passed our *`live_jsfile_links.txt`* in a while loop to ***LinkFinder*** for stdout on cli and store it in *`endpoints.txt`*. You use ***interlace*** (https://github.com/codingo/Interlace) by codingo to make the process fast using multi-threading.

"Why do we do this?"

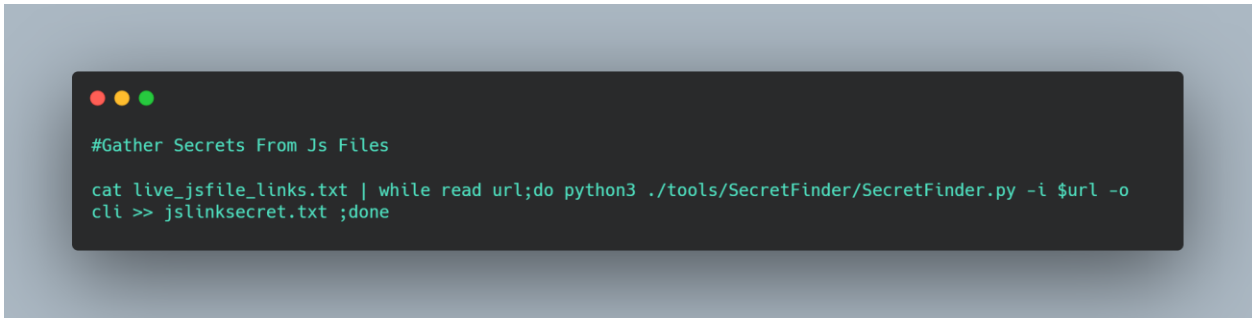
To find endpoints that are not available to the general user, but might be used by devs to perform various tasks. For Example, an endpoint that might be used by the developer to delete a user or an endpoint that is only available to an admin user.

After completion of this process, use grep to find those sensitive endpoints.

```
cat endpoints.txt | grep "admin"
```

3. Finding Secrets in the JSFiles.

Just by reading the heading you might have got the idea of what we will be doing in this part, for automation, we can use a great tool ***SecretFinder*** (https://github.com/m4ll0k/SecretFinder) made by *m4ll0k* and you can also use your regex with this tool. So, our code should look like…



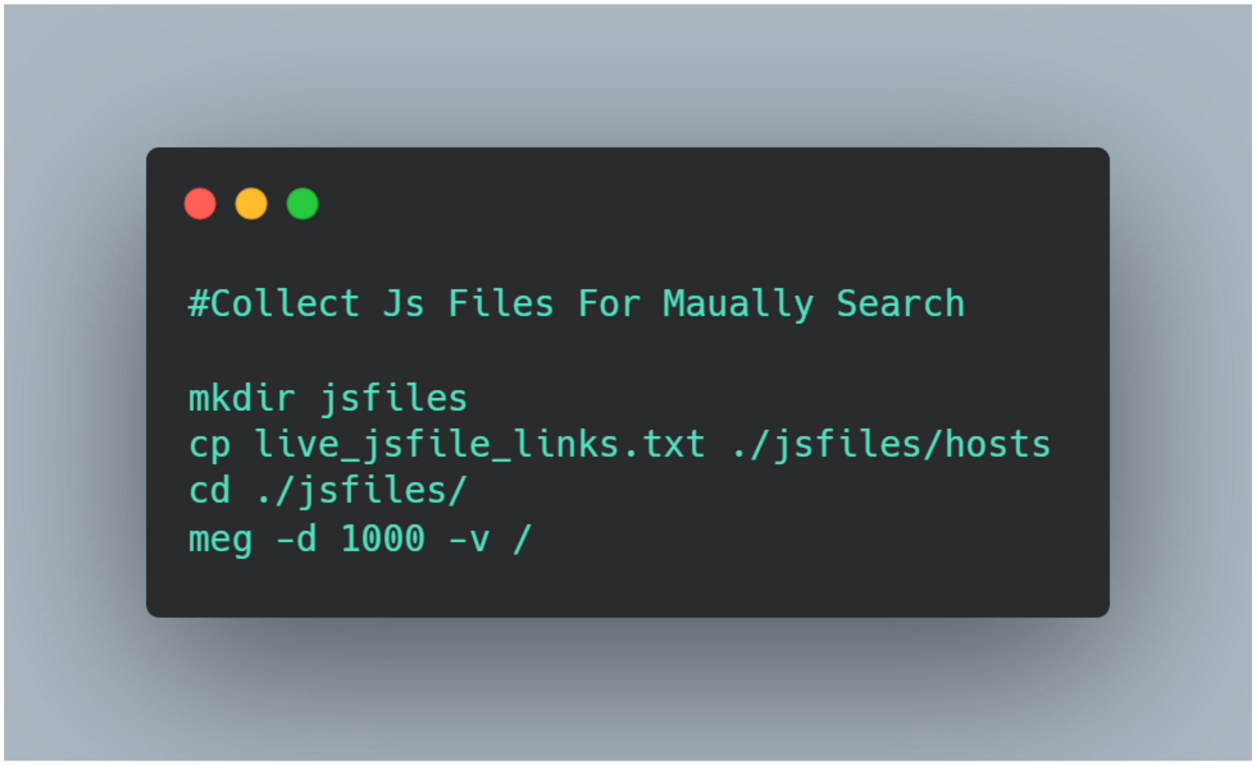
```
#Gather Secrets From Js Files

cat live_jsfile_links.txt | while read url;do python3 ./tools/SecretFinder/SecretFinder.py -i $url -o
cli >> jslinksecret.txt ;done
```

As you can see *`live_jsfile_links.txt`* is the file of live JSlinks which we passed in while loop to ***SecretFinder*** tool and stored all the stdout to *`jslinksecret.txt`*. Again, we can use interlace here to make process fast and multi-threaded.

When you look in `jslinksecrets.txt` file you will see a lot of results which are most of false-positive so it for the best to test them and then report it.

4. Get JSFiles locally for manual analysis.

So far, we used a lot of tools for finding endpoint and finding secrets but the best way is the manual analysis of the js files, we can use ***meg*** or ***wget+jsbeautifier*** or you can also use the online site like (https://codebeautify.org/jsviewer) for this purpose, but I prefer ***meg+gf*** combo as it easy to work with ***gf (***https://github.com/tomnomnom/gf) and there is a lot of patterns to ease your work.



```
#Collect Js Files For Maually Search

mkdir jsfiles
cp live_jsfile_links.txt ./jsfiles/hosts
cd ./jsfiles/
meg -d 1000 -v /
```

meg will fetch the js files locally and then you can use ***gf*** to file all sorts of things. Particularly you can use (https://github.com/dwisiswant0/gf-secrets) these patterns, also don't forget to check comment in js files which might contain hard-coded credentials.

Conclusion

We covered some basic things in this blog, which are essential while hunting on a target and when it comes to js files, there are a lot of things you can do with js files like looking for *DOM-based vulnerability*, Also don’t forget to check out this video on HackerOne youtube channel.

Thank you for reading this blog it was my first blog, so might be a lot of mistakes. You can get the Script that I made while writing this blog on Github.

https://github.com/KathanP19/JSFScan.sh

I have made a lot of improvement’s in the script so don’t forget to check.