Analyzing javascript files

What is JS?

Javascript is a client side object oriented scripting language. In essence this has several meanings:

- Client side, it runs on the computer of the client (victim)
- Object oriented (Programming term)
- Scripting language, this means cross site scripting is also possible

Developers have used this over the years to make static websites a bit more interactive and beautiful with things like javascript image carrousels but also XHR requests and AJAX requests to the backend server to automatically fill in a page. Javascript can do many things and for this reason it's of interest to us.

We can either analyze a javascript file statically (not running it) and dynamically (debugging or running it). We will mostly focus on static analysis here.

What does a JS file contain?

Besides the regular cross site scripting sinks (locations where our XSS attack vector is reflected in the JS) we can also find several other juicy secrets in there that we can use.

These secrets can contain but are not limited to:

- New endpoints, one time i found a whole list of endpoints in the comments
- Hidden parameters
- API keys, sometimes they are supposed to be public though, so be careful with these. Verify the impact before you report!
  https://github.com/streaak/keyhacks

- Business logic, which we might be able to abuse like client side calculations of prizes
- Secrets/passwords
- Potentially dangerous areas in the javascript code such as eval() or setinnerhtml(). These are DOM sinks and can lead to DOM XSS

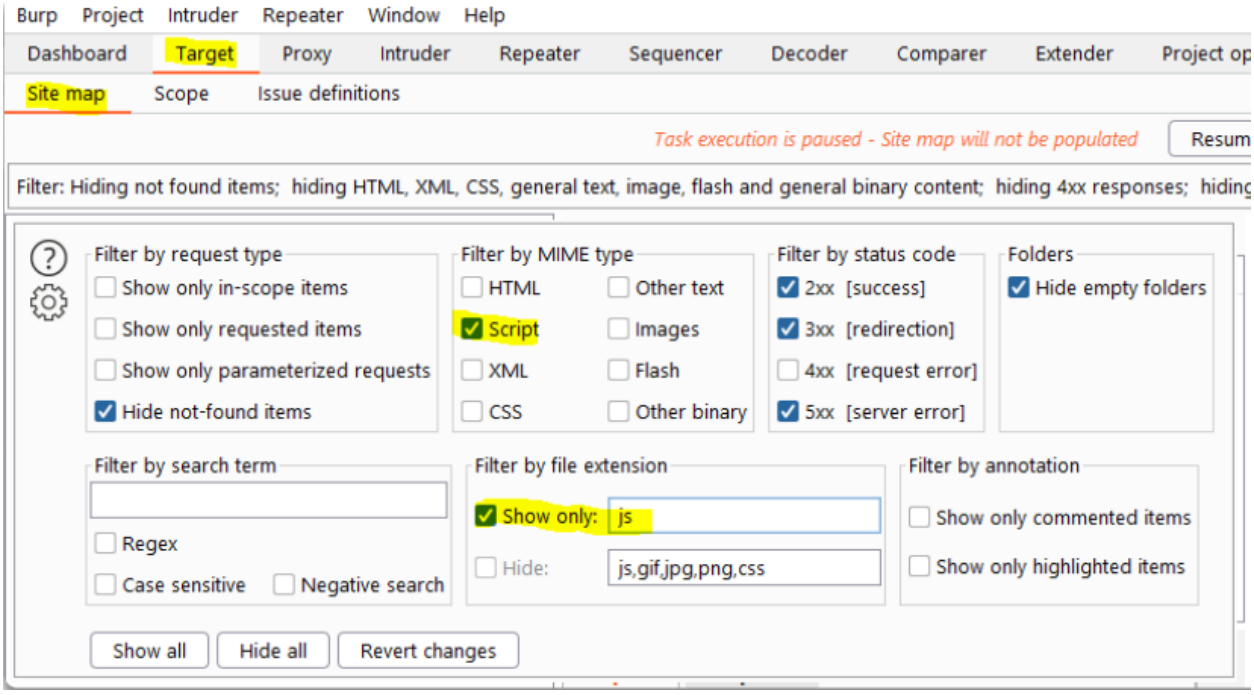
Attack strategy

For our attack stragey we first need to gather all the javascript files from a website. We have several options to do this automatically for us or we can look in the HTML source code manually but this will not catch all the JS files as some files might be called nested (a JS file called from inside another JS file), these would not show up in our initial manual scan.

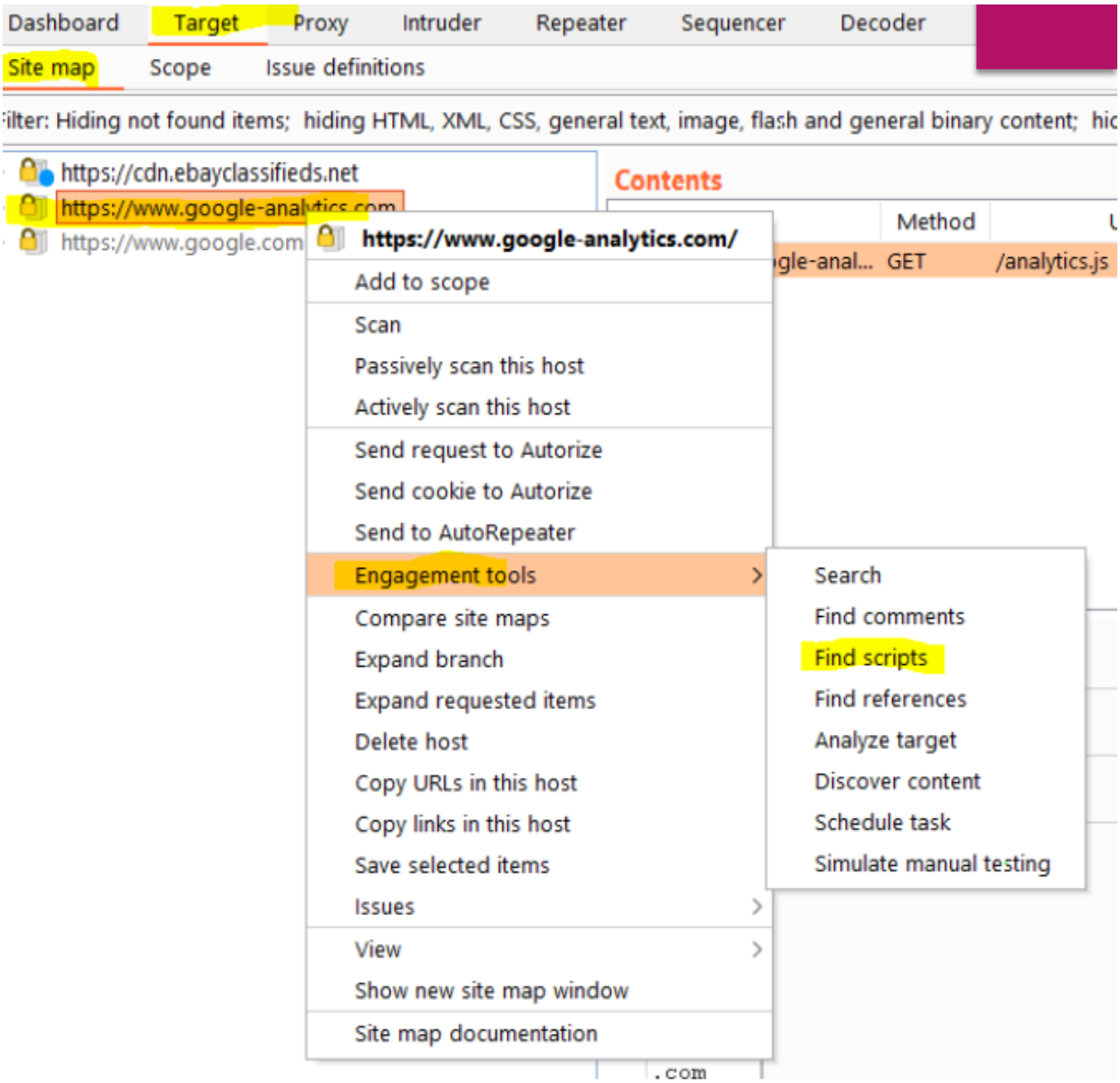
Using BURP SUITE

For our automatic scan we will want to use burp filters later on to explore all of our javascript files. To do this:

- Open burp
- Set your scope right
- Explore the site manually by clicking around
- Open the burp site map tab
- Click on the "Filter" Box
- Click on the "Script" checkbox and make sure it's the only one active under "Mime type"
- Under "Filter by file extension" , click "Show only" and fill in JS in the box



If you have burp suite pro, you can also right click on your target in the site map and under the engagement tools you will the option to "Find scripts". This will effectively do the same after exploring your target manually but the results will be displayed prettier.



Using waybackurls

Install waybackurls, using this tool we can also grep for any JS files that might not be linked anymore but still online.

```
go get github.com/tomnomnom/waybackurls
```

```
waybackurls google.com | grep "\.js" | uniq | sort
```

Defense mechanisms

Developers use a range of defense mechanisms to hold us off but that's okay. We can get around those by being dilligent and making sure that we take our time.

JS Obfuscation

- This is where developers will make it intenionally hard to read the code for humans but machines don't have any problem reading this code. This is harder to decipher but with some dilligence it can be done.
- https://stackoverflow.com/questions/194397/how-can-i-obfuscate-protect-javascript
- https://www.dcode.fr/javascript-unobfuscator (doesn't seem to work well)

JS Chunking

- This is where the developers chops up the JS into little pieces that all reference eachother. Very annoying to get arround and it's just hard work puzzling together the code

If we are trying to defeat these mechanisms it might help to set up a replica of you targets environment and to run the code statically.

Analysing JS files

So now that we have a ton of JS files, we can analyse them manually or we can run some tools on them. The cool thing is that these tools don't always need to have the JS files downloaded. It is possible for tools like linkfinder to crawl a domain for JS files.

We basically have a few tools in our toolbelt but today i want to focus on linkfinder and secretfinder.

Linkfinder

https://github.com/GerbenJavado/LinkFinder

Installing linkfinder is super simple

```
git clone https://github.com/GerbenJavado/LinkFinder.git
cd LinkFinder
python setup.py install
```

We then need to install the dependencies

```
$ pip3 install -r requirements.txt
```

We can then use linkfinder in a range of different modes.

```
python linkfinder.py -i https://example.com/1.js -o results.html
```

```
python linkfinder.py -i https://example.com -d
```

The results will consist a TON of new links that we can investigate and either dig deeper into manually or automatically scan them if the target allows it.

Secretfinder

Secretfinder builds on linkfinder but focusses on analyzing the JS for things like API keys.

https://github.com/GerbenJavado/LinkFinder

Installation is just as simple as with linkfinder

```
git clone https://github.com/m4ll0k/SecretFinder.git secretfinder
cd secretfinder
python -m pip install -r requirements.txt or pip install -r requirements.txt
python3 SecretFinder.py
```

Then we can start using it in the same way as linkfinder

```
python3 SecretFinder.py -i https://example.com/1.js -o results.html
```

```
python3 SecretFinder.py -i https://example.com -d
```

The results will consist of a list of sensitive data. The nature of this sensitive data can vary from API keys to litteral passwords. It's highly situational on how we can use these and sometimes they don't even have a use at all or are supposed to be public so judge carefully.