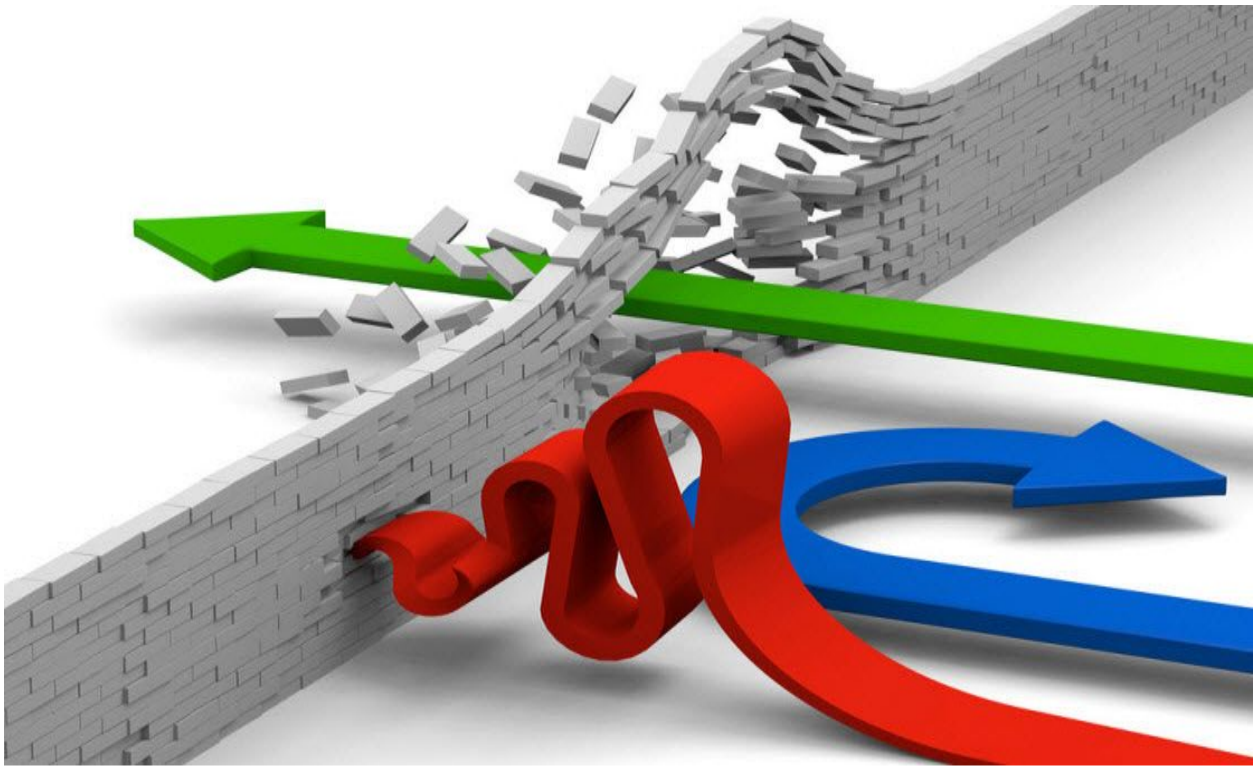


Penetration Testing Report

04.02.2019

Group csye6225-spring2019-xswl

Ang Li	001820694	li.ang2@husky.neu.edu
Changsi Liu	001831955	liu.changsi@husky.neu.edu
Xiaohan Zhao	001825212	zhao.xiaoh@husky.neu.edu
Junjie He	001893762	he.jun@husky.neu.edu

Overview

A penetration test, also known as a pen test, is a simulated cyber attack against your computer system to check for exploitable vulnerabilities. In the context of web application security, penetration testing is commonly used to augment a web application firewall (WAF).[1]

Goals

According to the requirements of assignment8[2], this documentation is built to our findings using penetration tests. The requirements are as followings:

- Identify and test your application against at least 3 attack vectors that do not exploit UI vulnerabilities. - You will document your findings in a PDF (Google doc exported as PDF) and commit it to your Github repository. - Your report should be as detailed as possible.
- You will document attacks on your own web application with and without the AWS WAF in place.

Your report should provide details on following along with screenshots:

1. Attack Vector
2. Result
3. Why did you choose this specific attack vectors

Specifications and Structures

Basic Concepts

It is very important to know some basic concepts before analysis the penetration testing: AWS WAF and attack vectors:

AWS WAF

AWS WAF is a web application firewall that helps protect your web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources. AWS WAF gives you control over which traffic to allow or block to your web applications by defining customizable web security rules. You can use AWS WAF to create custom rules that block common attack patterns, such as SQL injection or cross-site scripting, and rules that are designed for your specific application. New rules can be deployed within minutes, letting you respond quickly to changing traffic patterns. Also, AWS WAF includes a full-featured API that you can use to automate the creation, deployment, and maintenance of web security rules.[3]

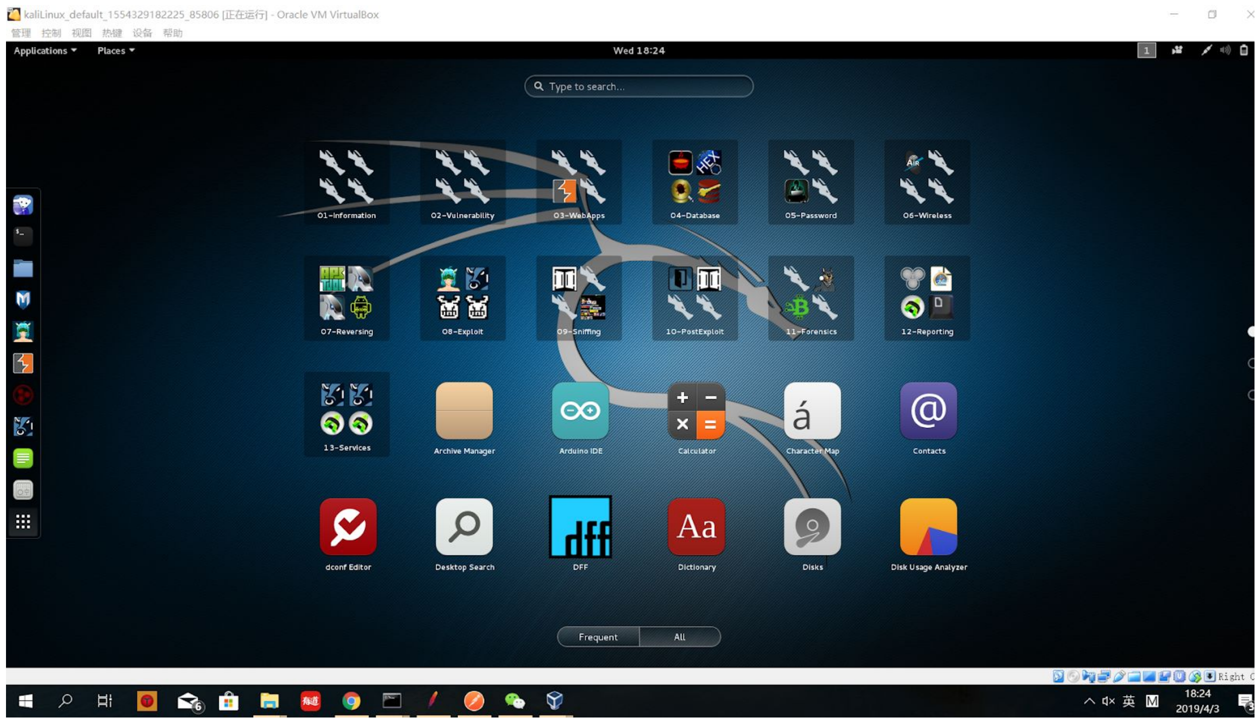
Attack Vectors

An attack vector is a path or means by which a hacker (or cracker) can gain access to a computer or network server in order to deliver a payload or malicious outcome. Attack vectors enable hackers to exploit system vulnerabilities, including the human element.[4]

Kali Linux

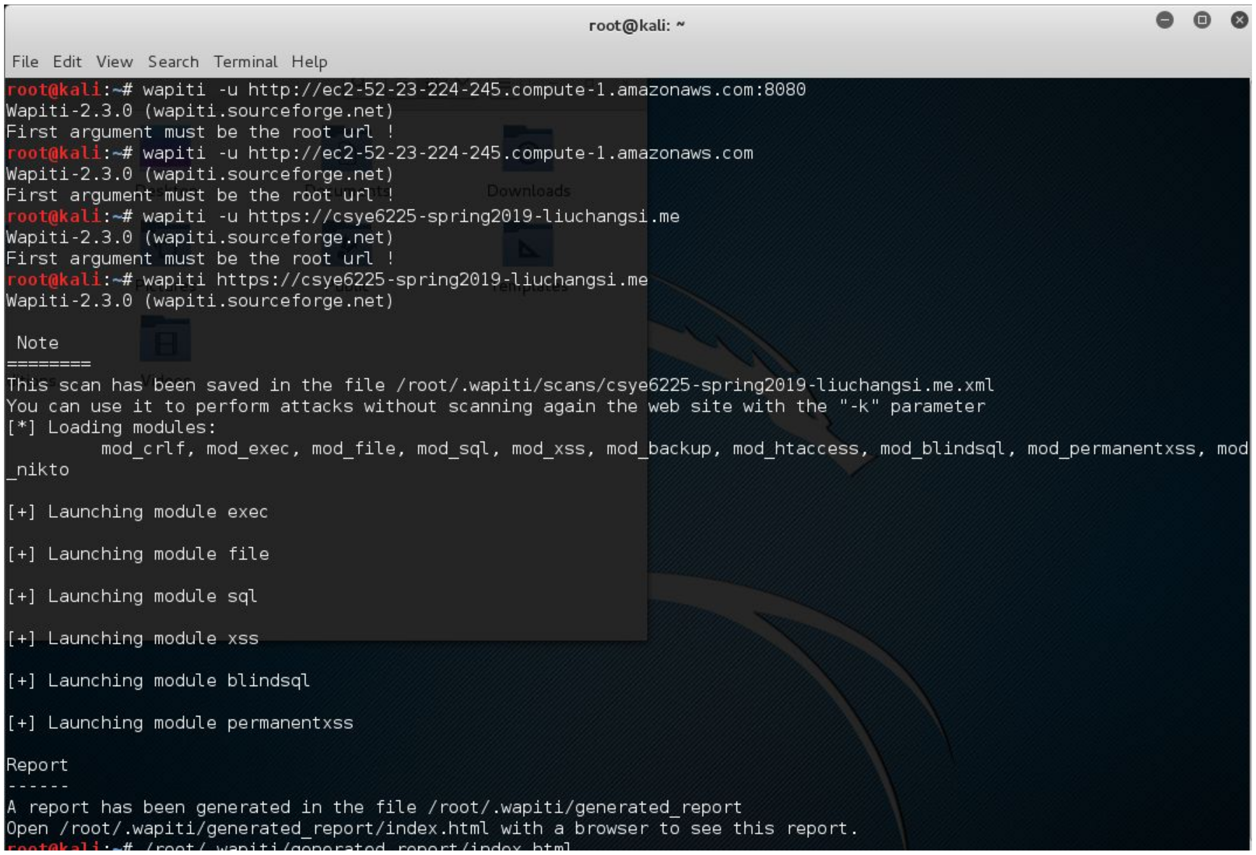
To attack web application more professionally and gracefully, we have used the most efficient and powerful attack operation system: kali linux to do the attack.

Kali Linux is a Debian-derived Linux distribution designed for digital forensics and penetration testing. It is maintained and funded by Offensive Security Ltd. It was developed by Mati Aharoni and Devon Kearns of Offensive Security through the rewrite of BackTrack, their previous information security testing Linux distribution based on Knoppix. The third core developer Raphaël Hertzog joined them as a Debian expert.[11]

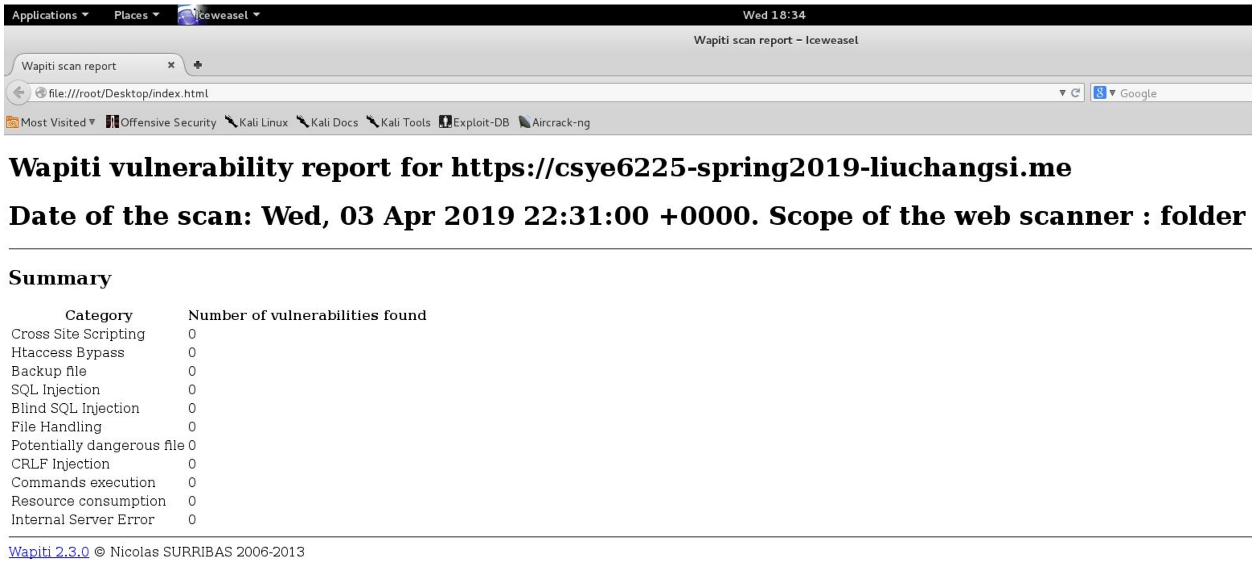


In fact, by generating a report from wapiti in kali linux, we found our web application is very secure without any possible vulnerabilities.

Screenshots are shown as followings:



```
root@kali:~# wapiti -u http://ec2-52-23-224-245.compute-1.amazonaws.com:8080
Wapiti-2.3.0 (wapiti.sourceforge.net)
First argument must be the root url !
root@kali:~# wapiti -u http://ec2-52-23-224-245.compute-1.amazonaws.com
Wapiti-2.3.0 (wapiti.sourceforge.net)
First argument must be the root url !
root@kali:~# wapiti -u https://csye6225-spring2019-liuchangsi.me
Wapiti-2.3.0 (wapiti.sourceforge.net)
First argument must be the root url !
root@kali:~# wapiti https://csye6225-spring2019-liuchangsi.me
Wapiti-2.3.0 (wapiti.sourceforge.net)

 Note
========
This scan has been saved in the file /root/.wapiti/scans/csye6225-spring2019-liuchangsi.me.xml
You can use it to perform attacks without scanning again the web site with the "-k" parameter
[*] Loading modules:
         mod_crlf, mod_exec, mod_file, mod_sql, mod_xss, mod_backup, mod_htaccess, mod_blindsql, mod_permanentxss, mod
_nikto

[+] Launching module exec

[+] Launching module file

[+] Launching module sql

[+] Launching module xss

[+] Launching module blindsql

[+] Launching module permanentxss

Report
------
A report has been generated in the file /root/.wapiti/generated_report
Open /root/.wapiti/generated_report/index.html with a browser to see this report.
```



Wapiti vulnerability report for https://csye6225-spring2019-liuchangsi.me

Date of the scan: Wed, 03 Apr 2019 22:31:00 +0000. Scope of the web scanner : folder

Summary

Category	Number of vulnerabilities found
Cross Site Scripting	0
Htaccess Bypass	0
Backup file	0
SQL Injection	0
Blind SQL Injection	0
File Handling	0
Potentially dangerous file	0
CRLF Injection	0
Commands execution	0
Resource consumption	0
Internal Server Error	0

Wapiti 2.3.0 © Nicolas SURRIBAS 2006-2013

Structures

In this documentation, we have also chosen three different attack vectors to test the penetrations of our project as follows:

- Packet Capture
- SQL Injection
- Cross-site Scripting

Besides the situations, we will test the penetrations in two different situations:

- With AWS WAF
- Without AWS WAF

And for each attack vector, we will post the screenshots of attack vectors, results, and reasons. And detail descriptions, findings, and the conclusion will also be posted as requirements.

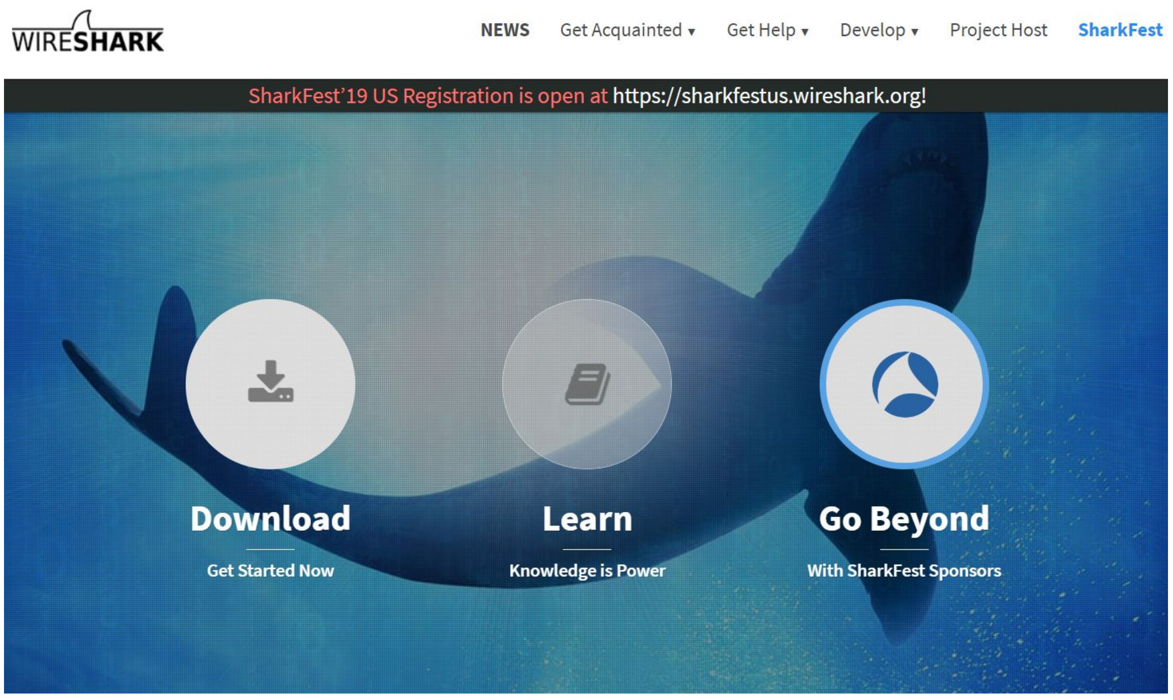
Packet Capture

Attack Vactor

Packet Capture is a networking term for intercepting a data packet that is crossing a specific point in a data network. Once a packet is captured in real-time, it is stored for a period of time so that it can be analyzed, and then either archived or discarded. Packets are captured and examined to help diagnose and solve network problems.[5]

In this assignment, we used Wireshark as the tool to do the packet capture.

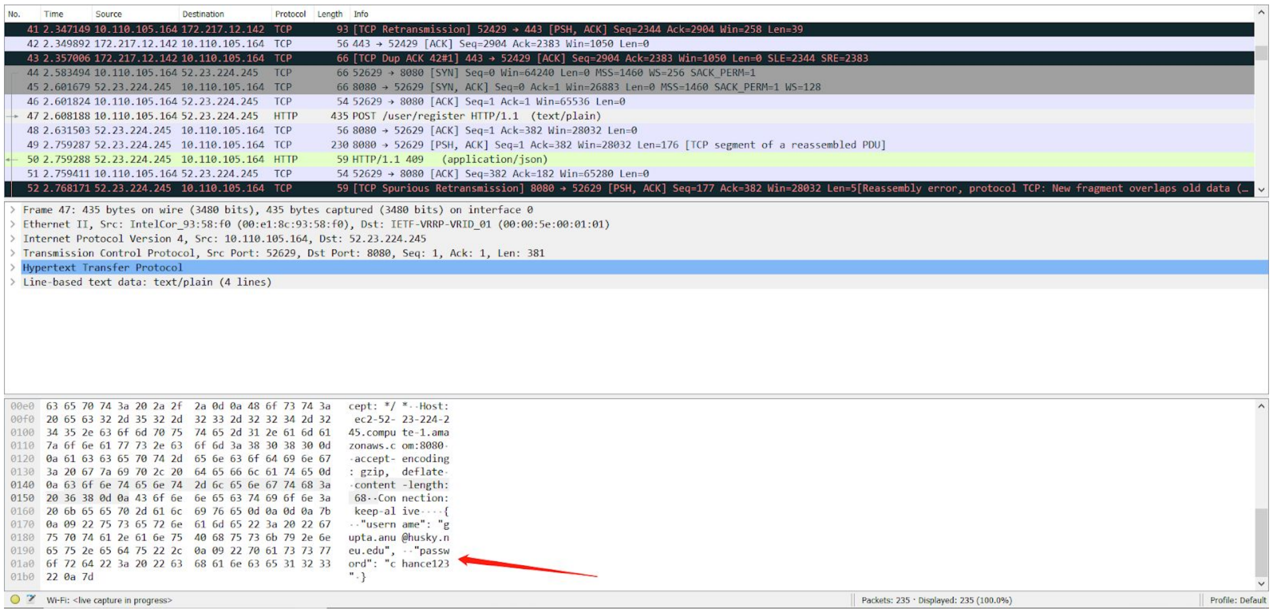
- *Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998.*[6]



To simulate the hacker attack in the real world, we used 2 computers to simulate. One computer as the route computer and published wifi while the other computer connected to the wifi generated by the first one. By this situation, a attacker can publish wifi without any private key to be connected by others. And if someone uses the wifi to visit our website, the hacker can use Wireshark or some other packet capture tools to capture the information.

Result

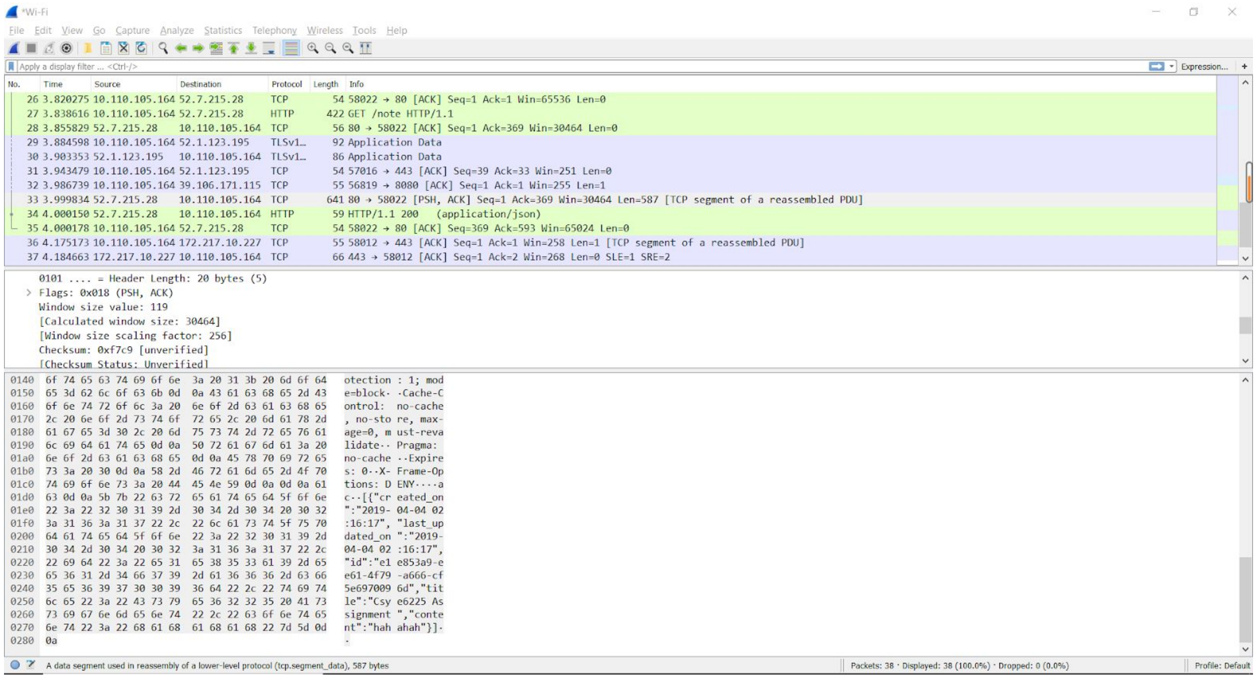
1. Without AWS WAF



```
No.  Time     Source          Destination     Protocol Length Info
  41 2.347149 10.110.105.164 172.217.12.142  TCP        93 [TCP Retransmission] 52429 → 443 [PSH, ACK] Seq=2344 Ack=2904 Win=258 Len=39
  42 2.349892 172.217.12.142 10.110.105.164  TCP        56 443 → 52429 [ACK] Seq=2904 Ack=2383 Win=1050 Len=0
  43 2.357006 172.217.12.142 10.110.105.164  TCP        66 [TCP Dup ACK 42#1] 443 → 52429 [ACK] Seq=2904 Ack=2383 Win=1050 Len=0 SLE=2344 SRE=2383
  44 2.583494 10.110.105.164 52.23.224.245   TCP        66 52629 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
  45 2.601679 52.23.224.245  10.110.105.164  TCP        66 8080 → 52629 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460 SACK_PERM=1 WS=128
  46 2.601824 10.110.105.164 52.23.224.245   TCP        54 52629 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0
  47 2.608188 10.110.105.164 52.23.224.245   HTTP      435 POST /user/register HTTP/1.1  (text/plain)
  48 2.631503 52.23.224.245  10.110.105.164  TCP        56 8080 → 52629 [ACK] Seq=1 Ack=382 Win=28032 Len=0
  49 2.759287 52.23.224.245  10.110.105.164  TCP       230 8080 → 52629 [PSH, ACK] Seq=1 Ack=382 Win=28032 Len=176 [TCP segment of a reassembled PDU]
  50 2.759288 52.23.224.245  10.110.105.164  HTTP       59 HTTP/1.1 409   (application/json)
  51 2.759411 10.110.105.164 52.23.224.245   TCP        54 52629 → 8080 [ACK] Seq=382 Ack=182 Win=65280 Len=0
  52 2.768171 52.23.224.245  10.110.105.164  TCP        59 [TCP Spurious Retransmission] 8080 → 52629 [PSH, ACK] Seq=177 Ack=382 Win=28032 Len=5[Reassembly error, protocol TCP: New fragment overlaps old data (…

> Frame 47: 435 bytes on wire (3480 bits), 435 bytes captured (3480 bits) on interface 0
> Ethernet II, Src: IntelCor_93:58:f0 (00:e1:8c:93:58:f0), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
> Internet Protocol Version 4, Src: 10.110.105.164, Dst: 52.23.224.245
> Transmission Control Protocol, Src Port: 52629, Dst Port: 8080, Seq: 1, Ack: 1, Len: 381
> Hypertext Transfer Protocol
> Line-based text data: text/plain (4 lines)

00e0  63 65 70 74 3a 20 2a 2f  2a 0d 0a 48 6f 73 74 3a   cept: */ *··Host:
00f0  20 65 63 32 2d 35 32 2d  32 33 2d 32 32 34 2d 32    ec2-52- 23-224-2
0100  34 35 2e 63 6f 6d 70 75  74 65 2d 31 2e 61 6d 61   45.compu te-1.ama
0110  7a 6f 6e 61 77 73 2e 63  6f 6d 3a 38 30 38 30 0d   zonaws.c om:8080·
0120  0a 61 63 63 65 70 74 2d  65 6e 63 6f 64 69 6e 67   ·accept- encoding
0130  3a 20 67 7a 69 70 2c 20  64 65 66 6c 61 74 65 0d   : gzip,  deflate·
0140  0a 63 6f 6e 74 65 6e 74  2d 6c 65 6e 67 74 68 3a   ·content -length:
0150  20 36 38 0d 0a 43 6f 6e  6e 65 63 74 69 6f 6e 3a    68··Con nection:
0160  20 6b 65 65 70 2d 61 6c  69 76 65 0d 0a 0d 0a 7b    keep-al ive····{
0170  0a 09 22 75 73 65 72 6e  61 6d 65 22 3a 20 22 67   ··"usern ame": "g
0180  75 70 74 61 2e 61 6e 75  40 68 75 73 6b 79 2e 6e   upta.anu @husky.n
0190  65 75 2e 65 64 75 22 2c  0a 09 22 70 61 73 73 77   eu.edu", ··"passw
01a0  6f 72 64 22 3a 20 22 63  68 61 6e 63 65 31 32 33   ord": "c hance123
01b0  22 0a 7d                                           "·}

Wi-Fi: <live capture in progress>          Packets: 235 · Displayed: 235 (100.0%)          Profile: Default
```

According to the screenshot, it can be seen that it is very simple for some hacker to get the capture information by using the packet capture attack without AWS WAF. And all the information from this packet is absolutely clear text, attackers can get all the information from this request.

2. With AWS WAF


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time     Source          Destination     Protocol Length Info
  26 3.820275 10.110.105.164 52.7.215.28     TCP        54 58022 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
  27 3.838616 10.110.105.164 52.7.215.28     HTTP      422 GET /note HTTP/1.1
  28 3.855829 52.7.215.28    10.110.105.164  TCP        56 80 → 58022 [ACK] Seq=1 Ack=369 Win=30464 Len=0
  29 3.884598 10.110.105.164 52.1.123.195    TLSv1…     92 Application Data
  30 3.903353 52.1.123.195   10.110.105.164  TLSv1…     86 Application Data
  31 3.943479 10.110.105.164 52.1.123.195    TCP        54 57016 → 443 [ACK] Seq=39 Ack=33 Win=251 Len=0
  32 3.986739 10.110.105.164 39.106.171.115  TCP        55 56819 → 8080 [ACK] Seq=1 Ack=1 Win=255 Len=1
  33 3.999834 52.7.215.28    10.110.105.164  TCP       641 80 → 58022 [PSH, ACK] Seq=1 Ack=369 Win=30464 Len=587 [TCP segment of a reassembled PDU]
  34 4.000150 52.7.215.28    10.110.105.164  HTTP       59 HTTP/1.1 200   (application/json)
  35 4.000178 10.110.105.164 52.7.215.28     TCP        54 58022 → 80 [ACK] Seq=369 Ack=593 Win=65024 Len=0
  36 4.175173 10.110.105.164 172.217.10.227  TCP        55 58012 → 443 [ACK] Seq=1 Ack=1 Win=258 Len=1 [TCP segment of a reassembled PDU]
  37 4.184663 172.217.10.227 10.110.105.164  TCP        66 443 → 58012 [ACK] Seq=1 Ack=2 Win=268 Len=0 SLE=1 SRE=2
```

```
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 119
    [Calculated window size: 30464]
    [Window size scaling factor: 256]
    Checksum: 0xf7c9 [unverified]
    [Checksum Status: Unverified]
```

```
0140  6f 74 65 63 74 69 6f 6e  3a 20 31 3b 20 6d 6f 64   otection : 1; mod
0150  65 3d 62 6c 6f 63 6b 0d  0a 43 61 63 68 65 2d 43   e=block· ·Cache-C
0160  6f 6e 74 72 6f 6c 3a 20  6e 6f 2d 63 61 63 68 65   ontrol:  no-cache
0170  2c 20 6e 6f 2d 73 74 6f  72 65 2c 20 6d 61 78 2d   , no-sto re, max-
0180  61 67 65 3d 30 2c 20 6d  75 73 74 2d 72 65 76 61   age=0, m ust-reva
0190  6c 69 64 61 74 65 0d 0a  50 72 61 67 6d 61 3a 20   lidate·· Pragma: 
01a0  6e 6f 2d 63 61 63 68 65  0d 0a 45 78 70 69 72 65   no-cache ··Expire
01b0  73 3a 20 30 0d 0a 58 2d  46 72 61 6d 65 2d 4f 70   s: 0··X- Frame-Op
01c0  74 69 6f 6e 73 3a 20 44  45 4e 59 0d 0a 0d 0a 61   tions: D ENY····a
01d0  63 0d 0a 5b 7b 22 63 72  65 61 74 65 64 5f 6f 6e   c··[{"cr eated_on
01e0  22 3a 22 32 30 31 39 2d  30 34 2d 30 34 20 30 32   ":"2019- 04-04 02
01f0  3a 31 36 3a 31 37 22 2c  22 6c 61 73 74 5f 75 70   :16:17", "last_up
0200  64 61 74 65 64 5f 6f 6e  22 3a 22 32 30 31 39 2d   dated_on ":"2019-
0210  30 34 2d 30 34 20 30 32  3a 31 36 3a 31 37 22 2c   04-04 02 :16:17",
0220  22 69 64 22 3a 22 65 31  65 38 35 33 61 39 2d 65   "id":"e1 e853a9-e
0230  65 36 31 2d 34 66 37 39  2d 61 36 36 36 2d 63 66   e61-4f79 -a666-cf
0240  35 65 36 39 37 30 30 39  36 64 22 2c 22 74 69 74   5e697009 6d","tit
0250  6c 65 22 3a 22 43 73 79  65 36 32 32 35 20 41 73   le":"Csy e6225 As
0260  73 69 67 6e 6d 65 6e 74  22 2c 22 63 6f 6e 74 65   signment ","conte
0270  6e 74 22 3a 22 68 61 68  61 68 61 68 22 7d 5d 0d   nt":"hah ahah"}]·
0280  0a                                                 ·
```

```
A data segment used in reassembly of a lower-level protocol (tcp.segment_data), 587 bytes    Packets: 38 · Displayed: 38 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

According to the screenshot, it can be seen that it is still doable for some hacker to get the capture information by using the packet capture attack with AWS WAF. And all the information from this packet is absolutely clear text, attackers can get all the information from this request.

Why did you choose this specific attack vectors

1. Reasons

Packet capture can be performed in-line or using a copy of the traffic that is sent by network switching devices to a packet capture device.

Entire packets or specific portions of a packet can be captured. A full packet includes two things: a payload and a header. The payload is the actual contents of the packet, while the header contains metadata, including the packet's source and destination address.

Analysis of packet capture data typically requires significant technical skills and often is performed with tools such as Wireshark.[5]

2. Conclusion

According to the packet capture attack testing, it can be known that our web application is weak at this situation, which means we cannot prevent any packet capture attack both with AWS WAF and without AWS WAF.

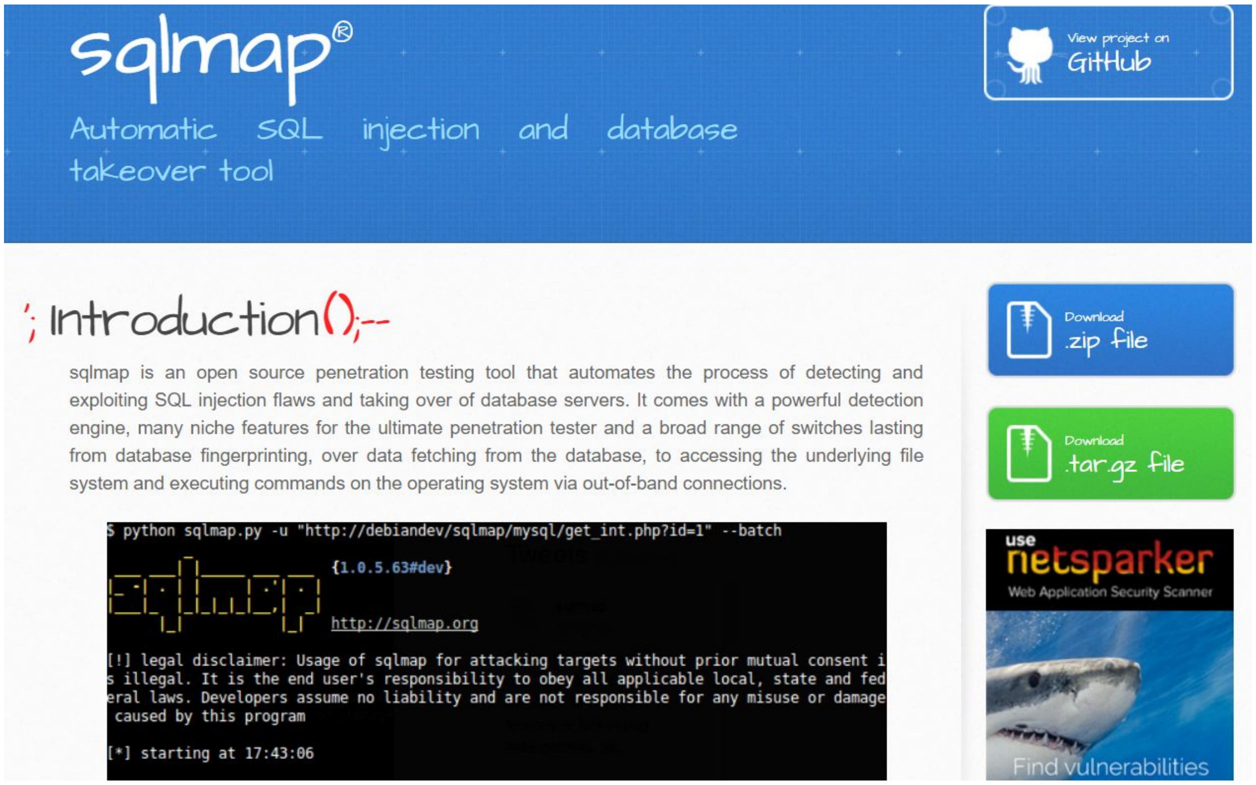
SQL Injection

Attack Vactor

SQL injection is a code injection technique, used to attack data-driven applications, in which diabolical SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.[7]

In this assignment, we used sqlmap as the tool to do the SQL injection.

- *sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.*[8]



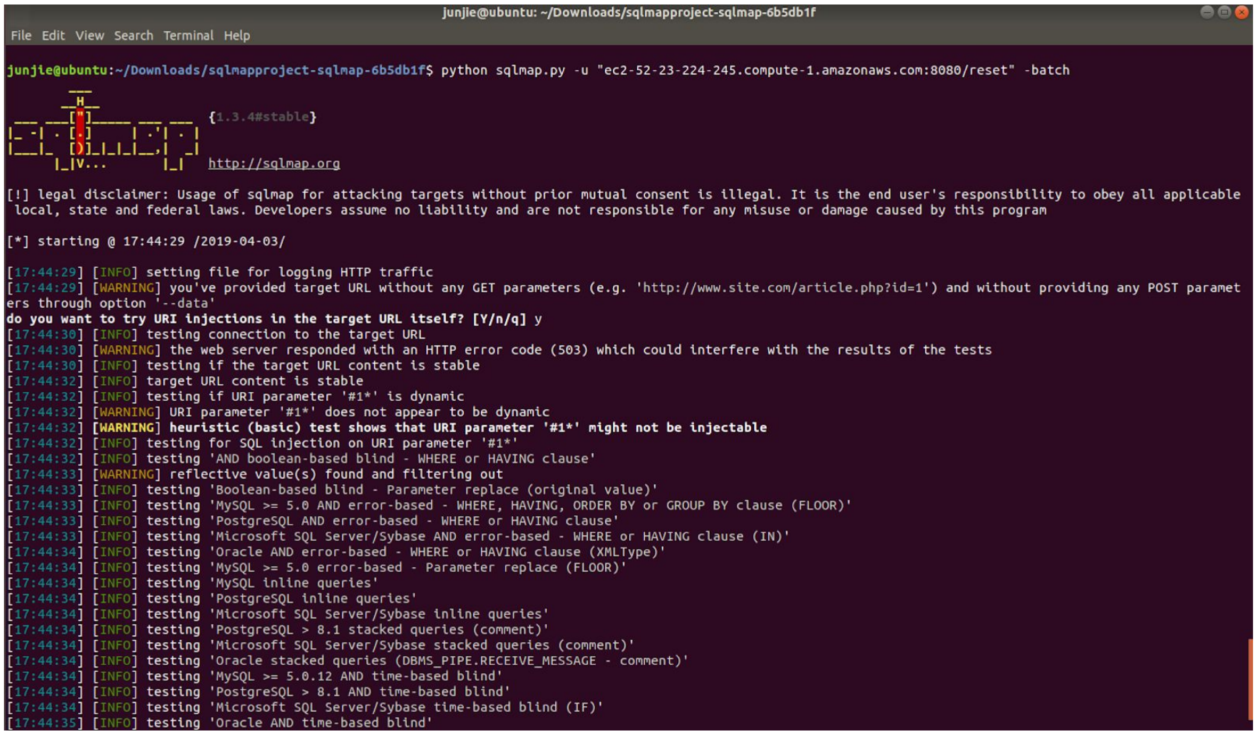
sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
                          {1.0.5.63#dev}

                          http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent i
s illegal. It is the end user's responsibility to obey all applicable local, state and fed
eral laws. Developers assume no liability and are not responsible for any misuse or damage
 caused by this program

[*] starting at 17:43:06
```

To simulate the hacker attack in the real world, we used sqlmap to try to attack our web application. And to attack more efficiently, we used --level and --risk model to do the attack.

Result

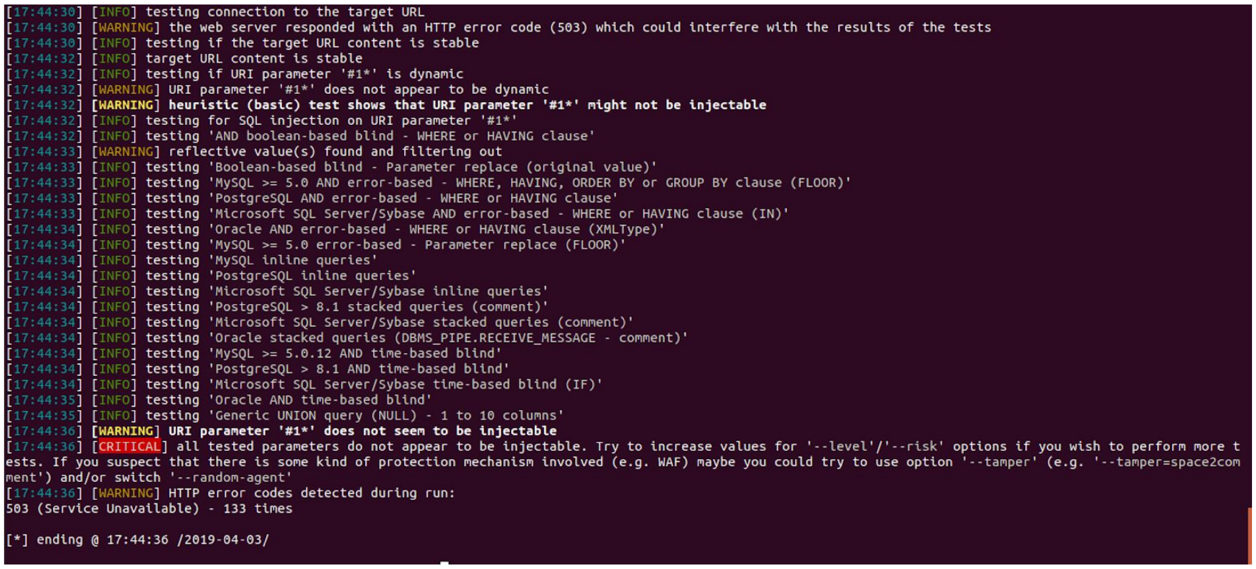
1. Without AWS WAF



junjie@ubuntu: ~/Downloads/sqlmapproject-sqlmap-6b5db1f

File Edit View Search Terminal Help

```
junjie@ubuntu:~/Downloads/sqlmapproject-sqlmap-6b5db1f$ python sqlmap.py -u "ec2-52-23-224-245.compute-1.amazonaws.com:8080/reset" -batch
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.3.4#stable}
|_ -| . [.]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable
 local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 17:44:29 /2019-04-03/

[17:44:29] [INFO] setting file for logging HTTP traffic
[17:44:29] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST paramet
ers through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] y
[17:44:30] [INFO] testing connection to the target URL
[17:44:30] [WARNING] the web server responded with an HTTP error code (503) which could interfere with the results of the tests
[17:44:30] [INFO] testing if the target URL content is stable
[17:44:32] [INFO] target URL content is stable
[17:44:32] [INFO] testing if URI parameter '#1*' is dynamic
[17:44:32] [WARNING] URI parameter '#1*' does not appear to be dynamic
[17:44:32] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[17:44:32] [INFO] testing for SQL injection on URI parameter '#1*'
[17:44:32] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[17:44:33] [WARNING] reflective value(s) found and filtering out
[17:44:33] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[17:44:33] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[17:44:33] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[17:44:33] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[17:44:34] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[17:44:34] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[17:44:34] [INFO] testing 'MySQL inline queries'
[17:44:34] [INFO] testing 'PostgreSQL inline queries'
[17:44:34] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[17:44:34] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[17:44:34] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[17:44:34] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[17:44:34] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[17:44:34] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[17:44:34] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[17:44:35] [INFO] testing 'Oracle AND time-based blind'
```



```
[17:44:30] [INFO] testing connection to the target URL
[17:44:30] [WARNING] the web server responded with an HTTP error code (503) which could interfere with the results of the tests
[17:44:30] [INFO] testing if the target URL content is stable
[17:44:32] [INFO] target URL content is stable
[17:44:32] [INFO] testing if URI parameter '#1*' is dynamic
[17:44:32] [WARNING] URI parameter '#1*' does not appear to be dynamic
[17:44:32] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[17:44:32] [INFO] testing for SQL injection on URI parameter '#1*'
[17:44:32] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[17:44:33] [WARNING] reflective value(s) found and filtering out
[17:44:33] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[17:44:33] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[17:44:33] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[17:44:33] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[17:44:34] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[17:44:34] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[17:44:34] [INFO] testing 'MySQL inline queries'
[17:44:34] [INFO] testing 'PostgreSQL inline queries'
[17:44:34] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[17:44:34] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[17:44:34] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[17:44:34] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[17:44:34] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[17:44:34] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[17:44:34] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[17:44:35] [INFO] testing 'Oracle AND time-based blind'
[17:44:35] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[17:44:36] [WARNING] URI parameter '#1*' does not seem to be injectable
[17:44:36] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more t
ests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2com
ment') and/or switch '--random-agent'
[17:44:36] [WARNING] HTTP error codes detected during run:
503 (Service Unavailable) - 133 times

[*] ending @ 17:44:36 /2019-04-03/
```

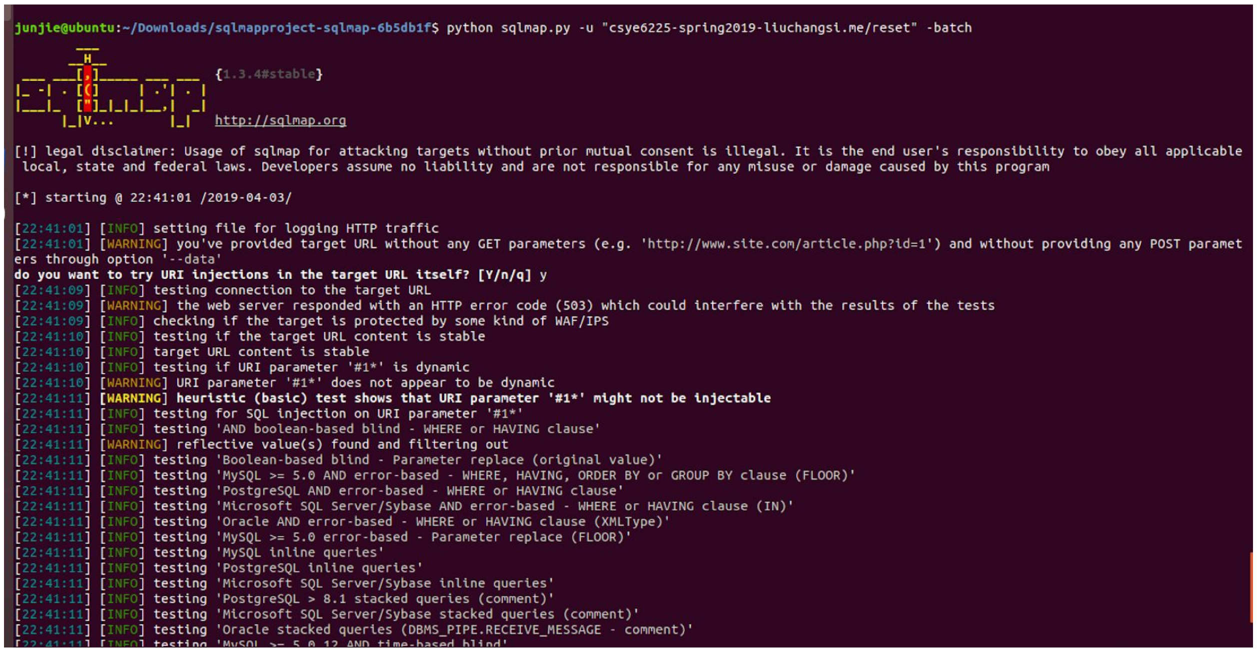
It can be seen from the screenshot that our web application can not be attacked from the SQL injection vector. This is because we used a very powerful database connection pool: [Druid from alibaba]

8. How to configure against SQL injection attacks

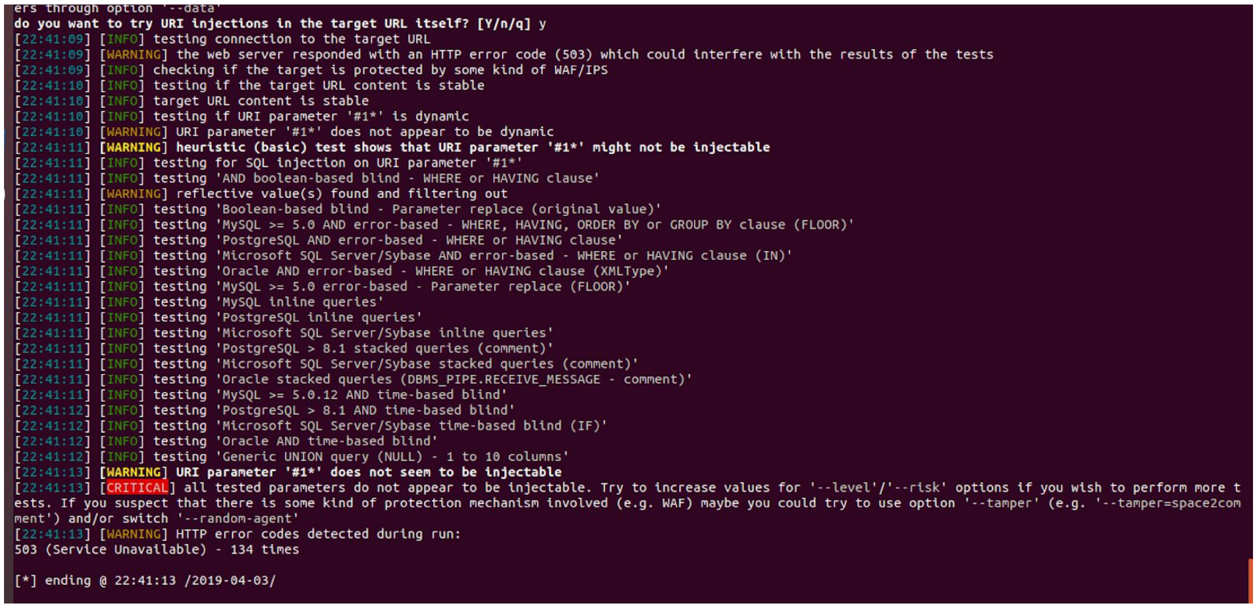
Druid provides WallFilter, it is based on the SQL semantic analysis to defense SQL injection attacks, reference : https://github.com/alibaba/druid/wiki/%E9%85%8D%E7%BD%AE-wallfilter

So even without AWS WAF, our web application can prevent SQL injection attacks using sqlmap.

2. With AWS WAF



```
junjie@ubuntu:~/Downloads/sqlmapproject-sqlmap-6b5db1f$ python sqlmap.py -u "csye6225-spring2019-liuchangsi.me/reset" -batch
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3.4#stable}
|_ -| . [(]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable
 local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 22:41:01 /2019-04-03/

[22:41:01] [INFO] setting file for logging HTTP traffic
[22:41:01] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST paramet
ers through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] y
[22:41:09] [INFO] testing connection to the target URL
[22:41:09] [WARNING] the web server responded with an HTTP error code (503) which could interfere with the results of the tests
[22:41:09] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:41:10] [INFO] testing if the target URL content is stable
[22:41:10] [INFO] target URL content is stable
[22:41:10] [INFO] testing if URI parameter '#1*' is dynamic
[22:41:10] [WARNING] URI parameter '#1*' does not appear to be dynamic
[22:41:11] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[22:41:11] [INFO] testing for SQL injection on URI parameter '#1*'
[22:41:11] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:41:11] [WARNING] reflective value(s) found and filtering out
[22:41:11] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[22:41:11] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[22:41:11] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[22:41:11] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[22:41:11] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[22:41:11] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[22:41:11] [INFO] testing 'MySQL inline queries'
[22:41:11] [INFO] testing 'PostgreSQL inline queries'
[22:41:11] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[22:41:11] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[22:41:11] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[22:41:11] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[22:41:11] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
```



```
ers through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] y
[22:41:09] [INFO] testing connection to the target URL
[22:41:09] [WARNING] the web server responded with an HTTP error code (503) which could interfere with the results of the tests
[22:41:09] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:41:10] [INFO] testing if the target URL content is stable
[22:41:10] [INFO] target URL content is stable
[22:41:10] [INFO] testing if URI parameter '#1*' is dynamic
[22:41:10] [WARNING] URI parameter '#1*' does not appear to be dynamic
[22:41:11] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[22:41:11] [INFO] testing for SQL injection on URI parameter '#1*'
[22:41:11] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:41:11] [WARNING] reflective value(s) found and filtering out
[22:41:11] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[22:41:11] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[22:41:11] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[22:41:11] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[22:41:11] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[22:41:11] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[22:41:11] [INFO] testing 'MySQL inline queries'
[22:41:11] [INFO] testing 'PostgreSQL inline queries'
[22:41:11] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[22:41:11] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[22:41:11] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[22:41:11] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[22:41:11] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[22:41:12] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[22:41:12] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[22:41:12] [INFO] testing 'Oracle AND time-based blind'
[22:41:12] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[22:41:13] [WARNING] URI parameter '#1*' does not seem to be injectable
[22:41:13] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more t
ests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2com
ment') and/or switch '--random-agent'
[22:41:13] [WARNING] HTTP error codes detected during run:
503 (Service Unavailable) - 134 times

[*] ending @ 22:41:13 /2019-04-03/
```

You can see it is still prevented using SQL injection, the reason is the same as the strategy without AWS WAF.

Why did you choose this specific attack vectors

1. Reasons

- Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.
- SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.
- SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.
- You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.
- In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.[9]

2. Conclusion

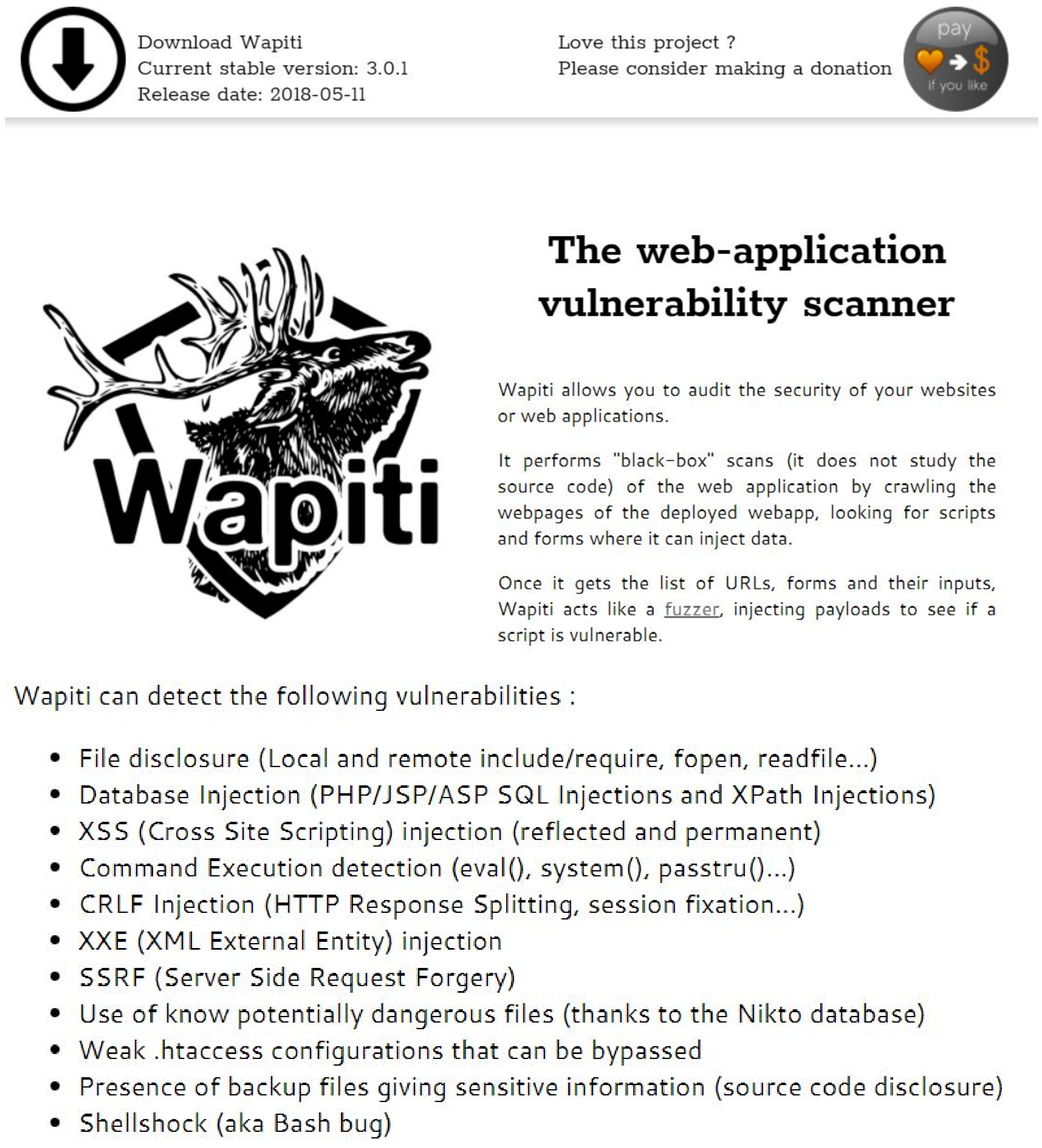
According to the SQL injection attack testing, it can be known that our web application is strong at this situation even using risk or high level model in sqlmap, which means we can easily prevent typical SQL injection attack both with and without AWS WAF. In fact, the sqlmap can not even figure out which SQL is used in our web application.

Cross-site Scripting

Attack Vactor

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007. In 2017, XSS is still considered a major threat vector.[2] XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.[10]

In this assignment, even we have already used wapiti to generate a report including the XSS attack, we will still test the XSS attack via browsers.



Download Wapiti
Current stable version: 3.0.1
Release date: 2018-05-11

Love this project ?
Please consider making a donation

The web-application vulnerability scanner

Wapiti allows you to audit the security of your websites or web applications.

It performs "black-box" scans (it does not study the source code) of the web application by crawling the webpages of the deployed webapp, looking for scripts and forms where it can inject data.

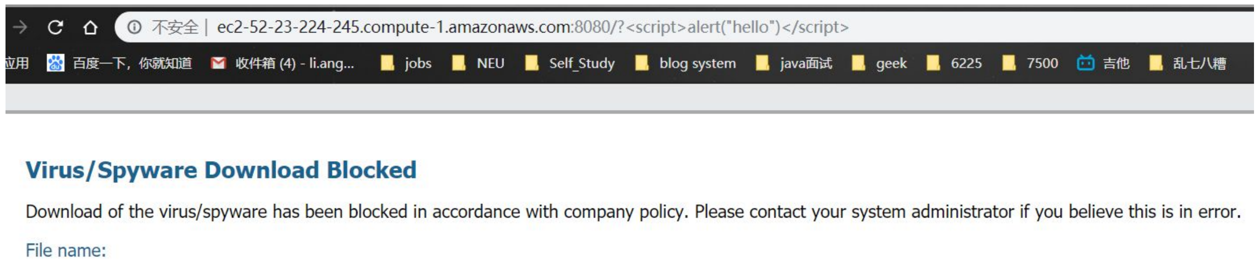
Once it gets the list of URLs, forms and their inputs, Wapiti acts like a fuzzer, injecting payloads to see if a script is vulnerable.

Wapiti can detect the following vulnerabilities :

- File disclosure (Local and remote include/require, fopen, readfile...)
- Database Injection (PHP/JSP/ASP SQL Injections and XPath Injections)
- XSS (Cross Site Scripting) injection (reflected and permanent)
- Command Execution detection (eval(), system(), passtru()...)
- CRLF Injection (HTTP Response Splitting, session fixation...)
- XXE (XML External Entity) injection
- SSRF (Server Side Request Forgery)
- Use of know potentially dangerous files (thanks to the Nikto database)
- Weak .htaccess configurations that can be bypassed
- Presence of backup files giving sensitive information (source code disclosure)
- Shellshock (aka Bash bug)

Result

1. Without AWS WAF



不安全 | ec2-52-23-224-245.compute-1.amazonaws.com:8080/?<script>alert("hello")</script>

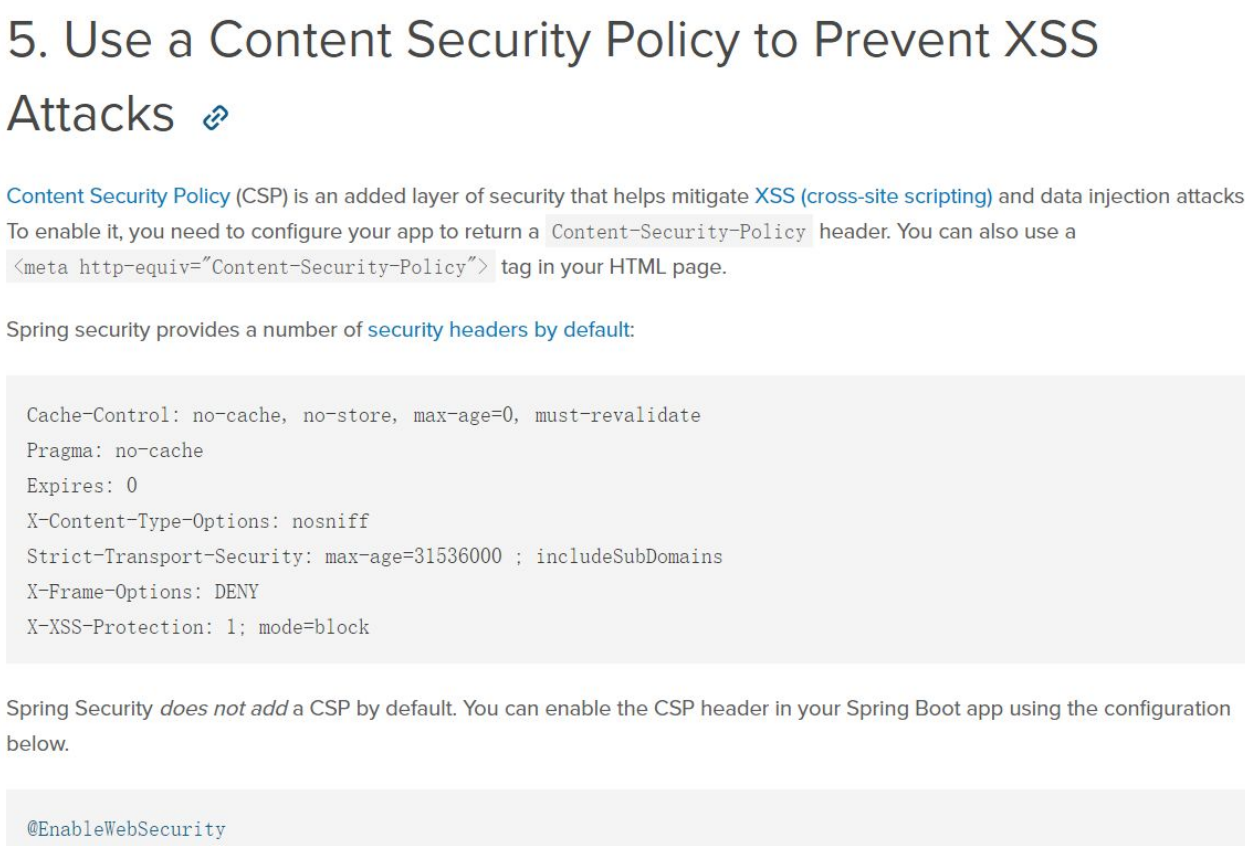
Virus/Spyware Download Blocked

Download of the virus/spyware has been blocked in accordance with company policy. Please contact your system administrator if you believe this is in error.

File name:

According to the screenshot, it can be seen that it is already rejected if some hacker tries to attack the web application via cross-site scripting vector. The screenshot page is generated by spring-boot.

Here is a website talking about what spring deal with XSS attack:[12]



5. Use a Content Security Policy to Prevent XSS Attacks

Content Security Policy (CSP) is an added layer of security that helps mitigate XSS (cross-site scripting) and data injection attacks. To enable it, you need to configure your app to return a `Content-Security-Policy` header. You can also use a `<meta http-equiv="Content-Security-Policy">` tag in your HTML page.

Spring security provides a number of security headers by default:

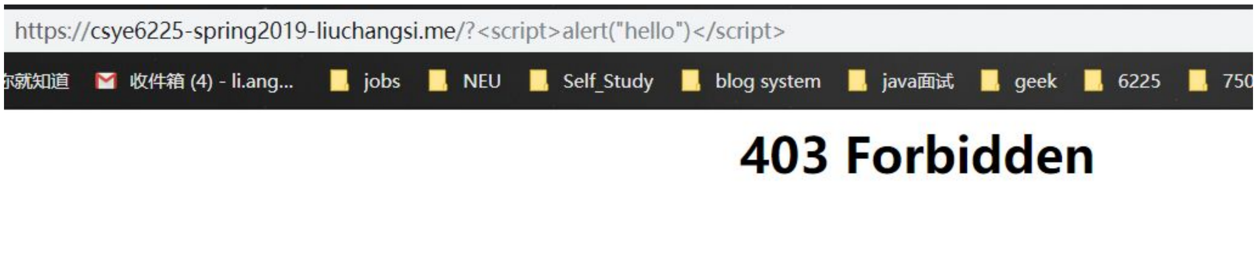
```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

Spring Security *does not add* a CSP by default. You can enable the CSP header in your Spring Boot app using the configuration below.

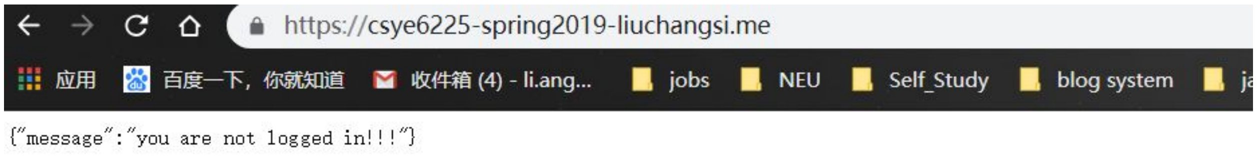
```
@EnableWebSecurity
```

So we can see that spring-boot can already protect the application from attacking with XSS.

2. With AWS WAF



403 Forbidden



```
{"message":"you are not logged in!!!"}
```

From the 2 screenshots, we can see that if you want to attack the application with XSS attack, the AWS WAF will forbid the request. You can only visit the website via URL without any XSS attempts.

This is different from the way without AWS WAF. The AWS WAF will block the request before arriving at the spring-boot level. So the request is handled by AWS WAF instead of spring-boot.

Why did you choose this specific attack vectors

1. Reasons

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. For more details on the different types of XSS flaws, see Types of Cross-Site Scripting.[13]

2. Conclusion

According to the XSS tests we did both with AWS WAF and without AWS WAF, we can conclude that when using AWS WAF, AWS WAF can prevent from XSS before attackers arrive at web level. If not using AWS WAF, the attackers can arrive at web application level but still blocked because of spring-boot security. So:

- AWS WAF will block an XSS attack if using AWS WAF
- Spring-boot will block an XSS attack if not using AWS WAF

References

[1] https://www.incapsula.com/web-application-security/penetration-testing.html

[2] https://spring2019.csye6225.cloud/assignments/08/#penetration-testing

[3] https://aws.amazon.com/waf/

[4] https://searchsecurity.techtarget.com/definition/attack-vector

[5] https://www.thousandeyes.com/learning/glossary/packet-capture

[6] https://www.wireshark.org/

[7] https://en.wikipedia.org/wiki/SQL_injection

[8] https://github.com/sqlmapproject/sqlmap

[9] https://www.acunetix.com/websitesecurity/sql-injection/

[10] https://en.wikipedia.org/wiki/Cross-site_scripting

[11] https://en.wikipedia.org/wiki/Kali_Linux

[12] https://developer.okta.com/blog/2018/07/30/10-ways-to-secure-spring-boot

[13] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)