East West University

Department of Electronics and Communications Engineering

Research Project Report

Web and Software Security

Submitted By:

Name: Ashikin Talaha

ID: 2015-1-50-031

Name: Ayan Chowdhury

ID: 2016-1-50-016

Supervisor:

Mohammad Rafsun Islam

Lecturer, Department of ECE

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of Mohammad Rafsun Islam, Lecturer, Department of Electronics and Communications Engineering, East West University. We also declare that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned	Signature
....	....
(Mohammad Rafsun Islam)	(Ashikin Talaha)
Supervisor	(ID: 2015-1-50-031)
	Signature
	....
	(Ayan Chowdhury)
	(ID: 2016-1-50-016)

Letter of Acceptance

This thesis report entitled “Web and Software Security” submitted by Ashikin Talaha (ID: 2015-1-50-031), Ayan Chowdhury (ID: 2016-1-50-016) to the Department of Electronics and Communications Engineering, East West University is accepted by the department in partial fulfillment of requirements for the Award of the Degree of Bachelor of Science and Engineering on August, 2019.

Supervisor:

Mohammad Rafsun Islam

(Lecturer),

Department of Electronics and Communications Engineering

Dr. Mohammad Moseeur Rahman

(Chairperson and Assistant Professor),

Department of Electronics and Communications Engineering,

East West University.

Acknowledgement

As it is true for everyone, we have also arrived at this point of achieving a goal in our life through various interaction and help from various people. However, written words are often elusive and harbor diverse interpretations even in one’s mother language. Therefore, we would not like to make efforts to find best words with us in this paper. This work was carried out in the Department of Electronics and Communications Engineering at East West university, Bangladesh.

We would like to express our sincere thanks to our Thesis supervisor Lecturer Mohammad Rafsun Islam for his endeavor help and cooperation in completion of the thesis. He constantly encouraged us with his valuable and wise suggestions throughout the thesis life cycle.

We would also like to thank faculty members of our university for their kind interest at various points of fellow the Thesis life-cycle and we also wish to thank all the members of Department of Electronics and Communications Engineering at East West University.

Ashikin Talaha
December, 2019

Ayan Chowdhury
December,2019

Table of Contents

Contents

Table of Figure:

List of Tables:

Abstract:

About 200 million websites are active at present. Billions of people use web applications for transferring information, money and communicating with each other. Web applications are made by humans. So, there may exist many kinds of vulnerabilities. The main reason for the weakness is the lack of choosing the proper programming languages. There are a lot of web application attacks that are existing now such as SQL injection, Buffer overflow, security misconfiguration, cross-site scripting, etc. So, the security issues of web applications are a great concern in presents. Developers are very interested to know about any kind of attack. In this project, we have created a tool to find different types of web application vulnerabilities of particular websites. This 'D-tect' tool will check eight dangerous and critical web application attacks. They are WordPress username enumerator, sensitive file detector, sub-domain scanner, port scanner, WordPress scanner, cross-site scripting (XSS), WordPress backup grabber, SQL injection. The tool will show host address, IP address, header information, the vulnerable scopes and server of the web application. There will be also detection of WordPress as it is mentioned that some vulnerabilities may arise due to using WordPress. The tool will check 1904 ports to find out the vulnerable ports. Sub-domains may have vulnerable DNS resolver that may help the attacker to exploit a system. That will be also scanned by the tool. The WordPress backup system will be also analyzed to find whether it is vulnerable or not. So, the tool will check for particular ports and try to inject different types of attacks. Then the corresponding result will be visible. This tool is created by using python and different modules and functions of python. There are different types of modules and functions are used to create the tool. the program can be run till the user wants to stop scanning.

Chapter 1

INTRODUCTION:

A web application or in short web app is a group of servlets, html pages, classes and other scripts which can be combined and executed on various platform from multiple vendors. It is computer program which perform operations on the internet by utilizing web browsers and web technologies. Any web application basically uses a group of server-side scripts like PHP and ASP to manage the information storage and retrieval of information. The client-side scripts are JavaScript and HTML which presents the information to users which are requested. Most of the web applications have used JavaScript, HTML5 and CSS. The client-side programming languages basically utilize the languages and build an application front-end. The server-side programs are done by using Python, Java or Ruby to create scripts for web applications. When a user generates a request to the web server though web browser or the applications user interface, the web server sends the request to particular web application server. Then the server is initialized to perform the requested task like fetching information from the database, or other processing methods to generate results and send to the particular user.

A lot of functions are used for web application. the most important functions are menu functions, script functions, A5w_DeleteFromWebRespository function, enumerate function, save to web app function, all web files function, edit component style function, get from web app function, A5_HTML_LIST_ADO function etc. A website can represent multiples of web application. Every web application has two users. The first one is business user which basically indicates the web provider. This user basically defines the functions for the end users. The end user does not have the interference or selection of designing for the web application. So, the web components for any web application is business application functionality, security, email components, forum. The components are added or modified on a regular basis. The web applications can be characterized to two types which are collections of static HTML pages and online application (simple form fill-up and information provision application and fully functioned business application).

There are several types of web applications. Static web application, dynamic web application, e-commerce, portal web app, animated web application, content management web application. The static web applications are basically developed in HTML and CSS. It can be also developed by jQuery and Ajax. It displays a little content and in not very flexible. Modification of contents is not easy in static web page. The html code must be downloaded and then the modification can be done. A static web application may consist of professional portfolios or different types of curriculums. Dynamic web applications are much complex in technical level and developed with PHP and ASP languages. This type of web applications use database to load data. The data of these applications updates with the requests of user at each time. In this type of web application, content upgrading or modification is very easy and the server doesn't even need to be accessed. Online e-commerce application is a bit complicated than the dynamic web application. It is like a shop on online. It has a new feature which is electrical payment methods. These web application needs some administrative panel. Portal web app is kind of application that includes several of sections or categories by a homepage. These applications can include many different features like chat, forum, emails etc. through registration. Another type of web application is animated web application. It is associated with flash technology. It allows to present contents with animated effects. But they are not suitable for web positioning and SEO optimization aims due to search engines cannot read the information which they contain. Another type is web application with content management system. This type of web application contains content management system (CMS) which is used to implement the changes and updates from the administrator. WordPress, Joomla, Drupal etc. are some popular CMS methods. WordPress is the most popular content manager for web application. It is very easy to customize and realize how does it works. It is a free version to all. Other versions of CMS are very useful to build communities[1]. The web applications are a great advantage for our daily life. By means of web application, works can be done from anywhere at any time. It saves the time and efforts to setup a physical software by downloading, installing, updating or managing. Web applications are compatible with any kind of current devices and platforms. Mobile web application allows user to connect the software without opening the computer or laptop. Web applications are scalable and pay per use system. They are always up-to-date with new technologies. Web applications does not need to build for all operating system version and testing. Web applications can be developed on a single platform and the testing can be done. So, web application development is very cost-effective development. It can be configured with any kind of system and interface. User can access to a web application from any place within a few times. This helps people to collaborate, communicate, and do so many other things at a time. It seems like real time collaboration, working or communicating. The web applications are easily customizable. The user interface can be easily customized rather than the desktop applications. It helps to update and charm look of the application. So, all users do not need to use the same configured settings of the application. In addition, most of the web applications are supported on various types of devices like mobile, laptop, android, PDA, tablets which are connected to internet.it is easier and comfortable to installation and maintenance. A web application can be easily implemented to online based shopping cart systems. Web applications have the most adaptable feature for increased numbers of workloads. Since it is easy to customize a web application, developers can add more database or servers to increase the performance overall having increased load to web application. Web applications are secured and have flexible core functionalities which makes them very useful and user friendly.

With the growth of web application, security concerns arise. There are thousands of websites are running on world. Some is for business, some for education, some for social mediums, some for storing data, and some for organization and so on. The World Wide Web has become one of the most important and global information mediums in the world[2]. The process of securing web applications is difficult as they are naturally open to public which may include malicious users. The inputs of web applications come from the http requests that are difficult to process accurately. So, improper or missing input validation may result in security vulnerabilities in web applications[3]. The attackers can apply many different approaches to the application to extract information and do potential harm to that particular web application. Sometimes the paths may come trivial to find and exploit. The contrast can be happened[4]. The main reasons for most of the web applications vulnerability is the lack of input validation and sanitization from the input. The vulnerabilities give the attacker an opportunity to perform malicious actions and collect sensitive information from the application to gain unauthorized access[5]. There are various types of web application attacks in now-a-days. SQL injection is the oldest and most dangerous attack still now. As the web applications rely on databases, so attacker choose to attack the database. They basically use malicious code which is inserted into strings. These strings are sent to the SQL server for performing the parsing and execution. This attacking process works by terminating the string unconsciously and appending into new command line. The SQL servers executes all the syntactically valid queries which the server receives. Cross site scripting is another web application attack type. It is a client-sided code injecting attack. As the web applications are built with different scrip languages, the attacker tries to inject malicious script in web browser as the distinguish of real web page. The actual attack happens when the user clicks on the web application the code start executing malicious script codes by itself. Then the web application becomes the medium to exporting the malicious scripts to user`s browser. A web application becomes vulnerable to XSS if it uses any kind of unsanitized and unprocessed user input. This may result in malicious script execution and stealing of user's cookie information from the browser. This helps attacker to achieve unauthorized access[6]. WordPress is an open and online source to build websites which is written in PHP. It is the most powerful blogging platform with website content management system. WordPress user enumeration is the technique to find out the users of WordPress installation. There are three enumeration techniques that are very fast approach to find the list of users of WordPress installation. After knowing the valid usernames brute force can be attempted to guess the password of those accounts. The three techniques are enumeration via author, JSON API, and via the login form. This may lead to inconsistency[7]. A significant risk can be represented by uploaded files to the web application. Because the attackers build codes to run on the application. The file upload option can help the attackers to perform such actions. So, sensitive files must be detected on the application. Subdomain is the additional part of main domain name of any web application. The subdomains are formed to categories and navigate the different segments of website[8]. The subdomain can be taken over by attacker. This deletes or removes the service that the subdomain was providing. The attacker will create a service with the additional same information pointing the same subdomain. Ports are the module of electronic, software of programming related docking point by which information can flow between two end users. The web applications can have multiple ports. There are ports for FTP, SMTP, HTTP, HTTPS, and TELNET etc. The attacker tries to find out the potential open ports to perform attack. This process can be done by port scanning. There are many websites that are running on WordPress. A web application can be attacked by scanning the WordPress information. There are several things which are vulnerable in a WordPress system such as themes, plugins, hosting vulnerability etc.[9]. There are many plugins in WordPress that can help to back up the WordPress information. It saves the complete installation and push them to external back up services. There are lot of approach to attack a WordPress web application. So, the backup grabber can be very dangerous for any kind of web applications. Though web applications can be secured by various methods, vulnerability testing and penetration testing is a unique process for securing web applications. It may help to find out the system vulnerabilities too[6]. There are various reasons that motivates an attacker to attack a web application. The major reasons are stealing sensitive and personal and organizational information, spreading malware, defacement of a web application and some for unnecessarily or out of curiosity, blackmailing, spreading spam, phishing, fraud etc.[2]. Attackers can be motivated to attack a web application to gain financial access. This can be done by stealing the credit card harvesting or other online payment account number and the pin from a web application. Some attacker may try to attack a web application to preventing real users from accessing into the web app system. They may show the user threatening messages or fake messages during accessing into the web application. The attacker may make the server so busy that it will not accept the request of a user until the false requests are processed from the buffer. Attacker can also attack a well secured web application to make bad impression about the app to users. Web applications can be attacked due to popularity. Popularity means more visitor will visit the web application. So, the intention of stealing information from the users may motivate an attacker to attack. This may also a newsworthy task. So, to get popularity the attackers may tend to attack popular web applications. Besides, attacking different web applications may occur due to perform various protest against government. Religious and corporate web applications can be attacked to perform protest. The attacking can be happened due to political reasons. The exposure of political confidential data may give advantage to some organization along with the attacker. These kinds of attack can be done by some group that has particular demands. All the attackers may not come from outside. Most of the attacks are done by the insider of any organization or government employees. They disclose the sensitive and confidential information about the web application system that helps the main attacker to gain the access for exploiting[2].

Our tool ‘D-TECT’ will run penetration testing to web applications to find out potential vulnerabilities. This will also show the related information to any web application. In our project, we have worked to identify recent most 8 web application vulnerabilities. The tool ‘D-TECT’ will sequentially perform different scanning and attempt to find out the vulnerable segments of any particular web application and website. The detection of expected scanning must be inputted along with the web application link. Then the tool ‘D-TECT’ will start performing the scanning and finding any vulnerable areas. The tool always shows the header type of the particular web application. After finishing first scan, the tool will ask for the second option and if no other scanning is required then the exit option is also included to the tool. The tool shows the server system that the web application is using. The tool ‘D-TECT’ is one of the best options to find out the vulnerabilities of a web application. This may help to strengthen the security of a web application and make them more secure for the users.

Chapter 2

BACKGROUND

2.1 Web Applications with Common Vulnerabilities:

This is a web application Vulnerability Testing app. So far, I think we all have known something about web application and its vulnerabilities.

Here is a table showing Vulnerable web applications

Name	URL	Technology	Credential
Acunetix Acuforum	http://testasp.vulnweb.com/	IIS, ASP, Microsoft SQL Server	unknown
Acunetix Acublog	http://testaspnet.vulnweb.com/	IIS, ASP.NET, Microsoft SQL Server	unknown
Acunetix SecurityTweets	http://testhtml5.vulnweb.com/	nginx, Python, Flask, CouchDB	admin:admin:1234
Acunetix Acuart	http://testphp.vulnweb.com/	Apache, PHP, MySQL	unknown
bWAPP	http://bwapp.ywnxs.com/	Ubuntu, Nginx, PHP	user: bee:bug

Table 1: Vulnerable web applications

These are some dummy sites that can be used for practicing or testing how they are vulnerable. These websites have some common vulnerability like the First one contains ‘SQL-injection’, the Second one contains ‘XSS or Cross Site Scripting’, third one has ‘Word-press Vulnerability’. The fourth one contains some backup file links in its word-press, And the last one has sensitive files [10].

All these Vulnerabilities are very common and easy to find and also the oldest. But it’s very sad to say that these vulnerabilities still do exist in some of the web applications we use.

2.2 Necessity:

Nowadays there is a lot of entrepreneurs who want to develop their own business. Most of them do not have that much technical knowledge. But whether you have the technical knowledge or not you must have some digital existence of your company like an app or a simple web site. But being new in the market they don’t want to spend that much money on this, so they contact noob developers for this job. Normally most of them build this kind of static apps or website by only downloading a simple template. They don’t even know properly what this template’s code contains. This makes them vulnerable.[11]

2.3 Vulnerabilities

There are 8 common vulnerabilities that can be found on a website or app. And for our project, we have also worked on these 8 vulnerabilities. Now let’s see them in detail and how they are vulnerable.

2.3.1 WORDPRESS USERNAME ENUMERATOR:

Web applications usually use an authentication mechanism to prevent unauthorized/anonymous users to access the application’s protected resources and functionalities. Attackers always try to find weaknesses in the authentication mechanism to get into the protected resources and functionalities.

Username enumeration is one of the most popular attacks that are performed on the authentication mechanism to identify the valid usernames on the system.

In many WordPress installations, it is possible to enumerate WordPress usernames through the author archives, including the admin username. To access the author archives, we just need to add author=n (where n equals any integer) as a parameter to the WordPress home page like the following:

http://example.com/?author=1

The request automatically will be redirected by WordPress to its counterparts:
http://example.com/author/admin/
Using this method, we will able to identify all the usernames by fuzzing the author parameter.

Word-press username can also be accessed by an error message.

2.3.2 SENSITIVE FILE DETECTOR:

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. One must validate the metadata extremely carefully before using it.

The targeted files are often configuration files that contain critical information for the operation of the application or website. The attacker can access this type of files if the server is not properly configured. For example, if access to indexes has not been blocked, the attacker can easily access to the sensitive files.

2.3.3 SUB-DOMAIN SCANNER:

Most hackers' senses start tingling at this point. 404 page indicates that no content is being served under the top-level directory and that we should attempt to add this subdomain to our personal

GitHub repository. Please note that this does not indicate that a takeover is possible on all applications. Some application types require you to check both HTTP and HTTPS responses for takeovers and others may not be vulnerable at all.

Second-Order Subdomain:

Second-order subdomain takeovers, what I like to refer to as "broken link hijacking"[12], are vulnerable subdomains that do not necessarily belong to the target but are used to serve content on the target's website. This means that a resource is being imported on the target page, for example, via a blob of JavaScript and the hacker can claim the subdomain from which the resource is being imported. Hijacking a host that is used somewhere on the page can ultimately lead to stored cross-site scripting since the adversary can load arbitrary client-side code on the target page. The reason why I wanted to list this issue in this guide, is to highlight the fact that, as a hacker, I do not want to only restrict myself to subdomains on the target host. You can easily expand your scope by inspecting source code and mapping out all the hosts that the target relies on.

Now that we have a high-level overview of what it takes to serve content on a misconfigured subdomain, the next step is to grasp the huge variety of techniques, tricks, and tools used to find vulnerable subdomains.

Before diving right in, we must first differentiate between scraping and brute-forcing, as both of these processes can help you discover subdomains, but can have different results. Scraping is a passive reconnaissance technique whereby one uses external services and sources to gather subdomains belonging to a specific host. Some services, such as DNS Dumpster and VirusTotal, index subdomains that have been crawled in the past allowing you to collect and sort the results quickly without much effort

And for brute-forcing is a direct attacking technique where the hacker directly tries the break the system by continuously attacking the server with a different combination of passwords.

2.3.4 NECESSITY of PORT SCANNING

We all know there are total of 65535 ports on a computer. But some common ports get usually hacked like TCP or UDP ports.

Port number	**Protocol's name**
21	FTP (File Transfer Protocol)
22	SSH (Secure Shell)
23	Telnet
25	SMTP (Simple Mail Transfer Protocol)
53	DNS (Domain Name System)
443	HTTP (Hyper Text Transfer Protocol)
110	POP3 (Post Office Protocol 3)
135	Windows RPC
137-139	Windows NetBIOS over TCP/IP
1433 & 1434	MSS (Microsoft SQL Server)

Table 2: Different Port numbers and Name of ports.

These ports get usually hacked because some misconfiguration or careless attitude of the user may keep the ports open. If it's possible to find an open port then it's easy to run an exploit on that specific port. Mostly they are considered as payloads.

Most people working in an office or organization might not know what ports are usually is. But they just use them. Most of the ports get opened automatically when a service is called in and it gets closed automatically with the service being closed. These are several apps or services that need to run in the background. For a non-technical guy, it is quite hard to find out which services are running in the background. So usually they kept that services open. This makes a port vulnerable to a hacker for taking advantage.

In web application security port scanning covers a major part. That's why there are several tools dedicated to port scanning's like Nmap or Nikto app in Kali Linux.

Besides, there are many other ports that can be vulnerable. SSH or telnet are used for establishing a remote connection. A hacker can create a honeypot attack and make these ports vulnerable [13].

2.3.5 WORDPRESS SCAN

Nowadays all of the websites are made on word-press themes. And the developments of the application are done using Word-press plugin.

There can be 5 main word-press vulnerabilities.

POOR HOSTING

All web hosts are not created equally. Some web-hosts are cannot separate user account properly.

WORDPRESS LOGIN

Word-press login systems are the most vulnerable system that hacker attacks. Word-press can be vulnerable to misconfiguration. Word-Press uses SQL for the database. In SQL all data is saved under a rooted tree. So, if one root dumped then all the other information under that root will also be dumped easily [14].

OUTDATED SOFTWARE

Most of the services on a web-application are word-press themes

Which may bring some special vulnerabilities with it.

2.3.6 CROSS SITE SCRIPTING

Of all the attacks we talked about so far, Cross-Site Scripting is the newest one at which still 60% of the websites are vulnerable

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.

There are mainly two types of XSS

1. Sorted XSS

2. Reflected XSS

Stored XSS Attacks

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS.

Reflected XSS Attacks

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other website. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user’s browser. The browser then executes the code because it came from a "trusted" server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-II XSS [15].

In this project, we have used 3 injectable commands as payloads the commands are

1: 'd4rk();"\'\\/}{d4rk',

2: 'd4rk</script><script>alert(1)</script>d4rk',

3: '<d4rk>'

2.3.7 WORDPRESS BACKUP GRABBER

Most of the word press sites Use word-press plugins for database or any related services. For example, backing up the user data and Use them whenever needed. That means a lot of work. Making that data visible in real-time.

For these purposes, 80% of the web-applications use word-press Theme plugins.

Almost all of them are very cheap so they offer very limited services. And some backdated plugins or software for free.

These plugins are not well maintained so it has some configuration file hidden in it which makes them vulnerable. In this project, we have tried to find out if there are any such files by creating a dictionary and assigning the names configuration file in the dictionary.

The names are

'wp-config.php~','wp-config.php.txt','wp-config.php.save','.wp-config.php.swp','wp-config.php.swp','wp-config.php.swo','wp-config.php_bak','wp-config.bak','wp-config.php.bak','wp-config.save','wp-config.old','wp-config.php.old','wp-config.php.orig','wp-config.orig','wp-config.php.original','wp-config.original','wp-config.txt'

2.3.8 SQL INJECTION

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual

property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities. The OWASP organization (Open Web Application Security Project) lists injections in their ‘OWASP Top 10’ 2017 document as the number one threat to web application security [16].
In this project we have used 3 injectable commands they are:

1. 105 OR 1=1
2. ""
3. '\"

2.3.9 Necessity of this Project

As all this website is made using different services so most of the time the developers don’t really know all about the underlying code. This is the biggest problem to face while this kind web-applications are used in the production level.

If any problem occurs it will take a long time to find out where is the problem. But before using any website actively the user can use this app to find out whether there is any problem with this web application.

This app is can act as a developer’s tool. They can use this tool after finishing the development to check whether their developed project is vulnerable or not.

Besides all these this app plays a vital role in reconnaissance or information gathering. There are a lot of apps who does vulnerability test for a specific domain. But they just analyze the domain and submit output only when the page is vulnerable. Also, the report contains only the information about the vulnerability. It’s limited because they don’t give much information about anything else. If anyone tries to find out any vulnerability using our app then it will give all the information related to that vulnerability. For example, if anyone want to do a port scanning then he all also see an analyzed result of the code for the particular website or web-application.

Later in the 5th chapter I have discussed more about this part.

As the user who will use this will also have the access to the code because it’s still not executable app. There are several payloads for XSS scan and checking SQL-injection. These two are the most common vulnerabilities that can be found in a modern web-application. But they might not be vulnerable for the same payload again and again. So, as we have the access of the code, we can change the payloads whenever we want. The report of this app is sorted in such a manner that any person who is not a technical guy can read through the report and understand it and if any user wants know more about the vulnerability then there are links of several documentation attached with the report.

Chapter 3

Literature Survey or Review:

With the advancement of web applications, security concerns arise. Web applications contain huge information about users and the company. But web applications may contain different types of vulnerabilities. The vulnerability means that there is a chance of getting attacked and losing information. In 2017, about 48% of web applications were at serious security risk of unauthorized access. About 91% attack was for client-side and 79% attack was for information leakage. Statistics say that about 70 types of weakness are existing in web applications in 2018. In these attacks, cross-site scripting is the most common and dangerous attack. The percentage has been increased to 88.5% in 2018 than in 2017 [17]. So different approaches are held to run an efficient web application scanning for finding vulnerabilities. This segment will describe some approaches in web application vulnerability searching that have been conducted.

In July 2009, Gencer Erdogan has proposed several techniques for testing the security of web applications. The main motivation was to develop methodologies and tools for securing web-based applications. The chosen methodologies were Agile Securing Testing, A penetration testing Approach, The OWASP Testing Framework. The OWASP testing framework is only designed for web applications. The proposed tool is Acunetix Web Vulnerability Scanner. It is an automated security testing tool with high coverage. It saves the results of crawling to fast inspection. The tool produces less false reports. This helps to minimize the time in scanning [18].

In August 2011, Frank van der Loo, has been described some web application vulnerabilities such as SQL injection, Xpath injection, XSS, CSRF (Cross-Site Request Forgery), Local file inclusion, HTTP response splitting, SSI injection, etc. To find out web application vulnerabilities the author (Frank) has suggested some commercial tools. He proposed that these tools can be used for web application vulnerability scanning. HP webinspect, Jsky, w3af, Websecurify, Wapiti, Arachni, etc. are the suggested tools. These tools were selected because they could detect the OWASP top 10 attacks easily [19].

In January 2013, Jose Enrique Charpentier had proposed for the web application security. In his paper, it is described that the security of web application is beneath in network security, the configuration of Operating system security measures and the webserver. Some doubts are shown for the firewall of the operating system. Some mentioned threats for any network are DDos, Session hijacking, spoofing, sniffing, Information gathering. The main threats that target a web server are unauthorized access, lacking in privileges setting up, denial of service, etc. During the research, some testing tools had been used to find web application vulnerabilities. The tools were hydra, SQLiX, Netsparker, and Fiddler. Hydra is used as a medium to gain an unauthorized remote connection to a web application system by login cracker. The SQLiX is a SQL injection scanning tool that is written in Perl. It crawls a web application and detects for SQL injection. It also identified the back-end of the database system and catches the functions that are running. Netsparker is a web application security scanner tool that reports false-positive status about the web application. Fiddler is a debugging proxy tool that looks for all logs of all the HTTP(S) traffic between the host computer and the internet. In this paper, the result of web application scanning is done by a pie chart. The vulnerability analysis ensures web application security. SQL injection is a complex problem. There are also some of the vulnerabilities are found in this project such as Password Transmitted Over HTTP, Cross-site Scripting, Cookie Not Marked As HttpOnly, Internal Path Leakage. The main purpose of this paper was to checking for vulnerabilities of web applications [20].

In November 2014, Zoran ĐURIĆ [5] had proposed a simulation-based dynamic analysis approach to scan web applications and the approach was performed as a black-box testing approach. The process was divided into five phases. The first phase was to web crawling. The crawling was configured that went beyond the login pages and explored all the pages of a web application. This will extract all the entry points and parse all the links to collect all the anchor parameters. If the authorization stops the crawling then phase two will be started to get some input. Phase two is the point of entry detection and extraction of input of the web application. Then the constraints are collected and the input to get the entry is done. All the constraints are not supported in all kinds of web browsers. To avoid this situation, HTTP requests are sent to web applications. In phase three, the testing had been implemented. The WAPTT used all the information of the entry points to generate valid HTTP requests for the web application that is to be tested. The WAPTT tool will check for SQL Injection attack using different types of queries such as Logically Incorrect Queries, Union Queries, PiggyBacked Queries and alco tautologies, inference attack to SQL server and detection of second-order SQL injection. The tool will also check for cross-site scripting attack (XSS parameter attack) and Buffer overflow attack. In phase 4, the analysis will be done. In this part, the tool will analyze all the HTTP response that is received by it and search for vulnerabilities. If no error status code is found after SQL injection checking, then the scanned web application can be marked as safe. If the error status exceeds more than 500 then the web application can be said as vulnerable enough to attack. The detection of XSS detection and Buffer Overflow is also done by checking the input validation and response from the server respectively. In phase five, the report will be generated automatically. The WAPTT tool was developed by using JAVA. A semi-automated web crawler was used because an automated crawler does not fulfill all the parsing all the web pages in the selected domain [5].

In 2015, Jai Narayan Goel, BM Mehtre made a plan to develop an efficient vulnerability assessment and penetration testing tool. They planned a life cycle of assessment and testing. They had chosen some techniques such as static analysis, automated testing, manual testing, fuzzing, black box testing, grey box testing, white box testing, etc. to check the vulnerabilities of a web application. Manual testing indicates that no tool is needed for the assessment of any web application scanning. The automated indicates that there should be tools for scanning and testing the web application. Fuzzing is the technique to input invalid syntax and value to crash the system of web applications. The black box testing is run from the outside network to the internal network. So, the tester does not need to worry about knowledge. In grey box testing, the tester may have some knowledge about the network architecture. In white-box testing, the tester has proper knowledge about all the systems and the network architecture. Then they have chosen penetration testing. But they did not propose any model in the conclusion rather than describing the necessity of vulnerability testing and penetration testing for cybersecurity [21].

In 2018, Muhammet Baykara in the International Journal of Computer Science and Mobile computing has analyzed some web application vulnerabilities and then scanning those vulnerabilities by different tools. Netsparker is a tool for scanning a web application. This tool is fully supported by AJAX, HTML5, and JavaScript-based web applications. It has tow versions and they are desktop and cloud. It is a built-in tool for web application scanning. The great disadvantage of this tool is that it has a fee of $5000 per year to license. Then the tool Acunetix is used. This is a scanning tool and controls the websites written in HTML and JavaScript. In this tool, the SDLC (software development lifecycle) is included that helps to integrate with the bug track system or project management. It also includes founded reports. It is very helpful to detect SQL injection and XSS vulnerabilities. WordPress vulnerabilities can be also found by using Acunetix. It has a web-based version. Vega is another free and open-source web application security analyzer tool. It is written in JAVA. It applies a blocker proxy to debug. The attacking module is written in JavaScript. Besides OWASP ZAP and Wapiti are included here. These are some tools to analyze web applications [3].

In all the segments, the researchers have talked about the web application security vulnerability analysis. There are very few works in developing a tool. most of the reports contain the basic idea of online based web application vulnerability scanning tools. These tools are very different from each other. Sometimes there is no kind of free version tool. The tools may attempt some unauthorized access to any kind of web application. The 'D-tect' tool helps to find the most recent eight web application attacks.

Chapter 4

INTERNAL ARCHITECTURE:

Before talking about the code, we must take look at the file that we have in our project root directory.

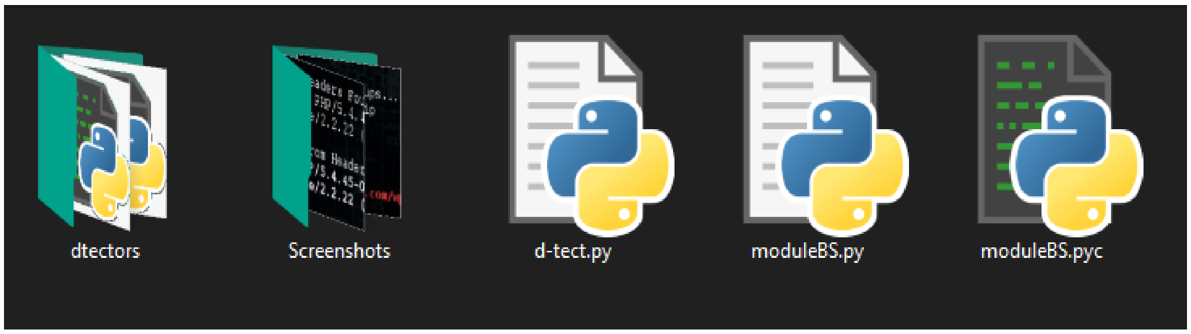


Figure 4. 1: root-directory

The above figure is showing us the orientation of the files that we have in our project. These four scripts and two folders in the root directory. The first folder is my editor's configuration. Then there is the 'detectors' folder. It does contain some important files. We will talk about that later. Then our main function 'd-tect.py'. Then 'moduleBS.py', this function plays a very important role in our project. This Script is only for the beautiful soup module. We will talk in detail about them in the later section of the project. Then there is the 'moduleBS.pyc' which is the binary executable file for the modules script. Then there is the readme.md which is the default web interface documentation of our project. Here only the basic function is mentioned. Now let's jump into the coding part.

4.1 detectors Folder

Here we can see here we have in total 12 files but only six of them are main scripts and the other

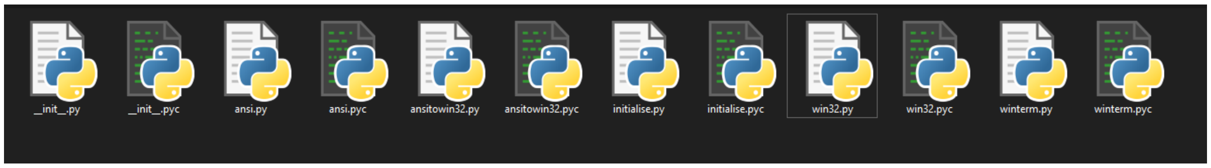
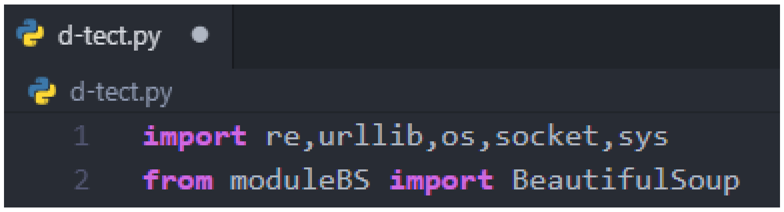


Figure 4. 2: detector Folder

6 files are the binary executable files of the main script. The first one is __init__.py. This makes the scripts of the entire project available to all other scripts.



```python
import re,urllib,os,socket,sys
from moduleBS import BeautifulSoup
```

Figure 4. 3: modules

For example, we can import ‘moduleBS’ script as a module in our main script ‘d-tect.py’. Which is shown in figure 4.3.

And the all other scripts are like ansi.py, ansiwin32.py, win32.py, winterm.py are mainly used for conversion of the data. As this is a web-application security testing tool so it will need a lot of scrapping and conversion. That’s why here we have used a python framework.

A framework is a collection of packages or module that helps to write reusable codes.

Here we have used a web-based framework. A web-based framework is a collection of packages or modules that developers to write applications or services without having to handle such low-level details as protocols, sockets or process/thread management.

Here we have used a framework named pyLons. It is an open-source web-based framework that mainly focuses on the rapid development of applications. This framework is mainly designed for incorporating some of the properties and best elements of popular languages such as Ruby, Perl, and Python [22].

Special features of cubic-web framework include

· URL Dispatch

· Routes

· Text-based templating

· URL mapping based on routes configuration via Web-Helpers

· HTML form generation and validity

· HTTP request handler

To use this framework first, we had to install it. As it is a pip module so we have used ‘pip install pylons’ for installing the framework. After executing the command, the detectors folder was automatically created to maintain the application or project.

This was all about the main architecture of the project now let’s dig into the codes.

4.2 Code Description

The coding part will be described in two main sections. One section is for the d-tect.py script and another section is for the moduleBS.py script.

d-tect.py is mainly used for doing the scan or using any other payloads where the moduleBS.py file is mainly used for scrapping.

4.2.1 d-tect.py

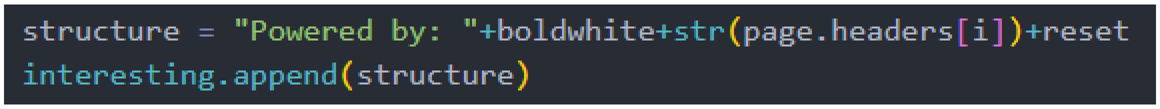
This is the main function of our project. While running we use python command and the file’s name to run or launch this app.

Here is a table showing used modules in this project.

Module’s Name	**Use**
Re	Making regular expression of certain tags for further development
urllib	For handling the URL library
Os	For OS commands
Socket	Connectivity
Sys	For using system
Module BS	web scrapping
Urlparse	Parsing web url
Detectors	For using and encoding different colors

Table 3: Modules

These are the main modules that are making this app functional. In this script there are mainly 13 functions written[23]. Now let’s see and know about the functions.

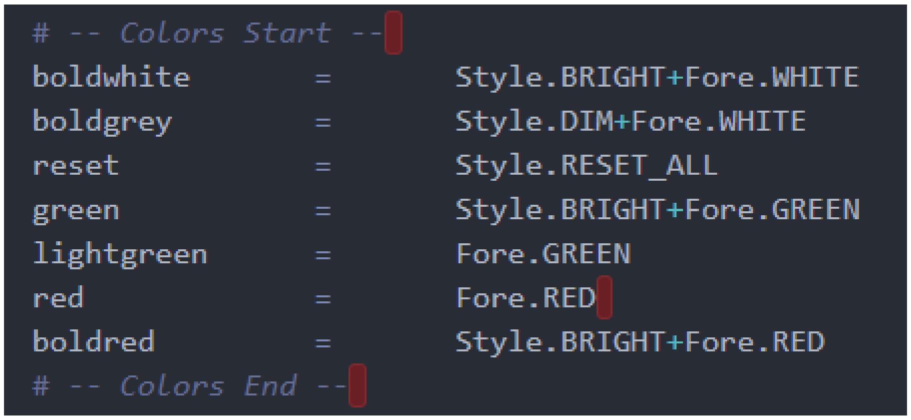


```python
structure = "Powered by: "+boldwhite+str(page.headers[i])+reset
interesting.append(structure)
```

Figure 4. 4: usage of ‘reset’

In all over the code in the formatting commands of the URLs we have used ‘reset’, ‘boldwhite’ etc. keywords. This might confuse the reader.

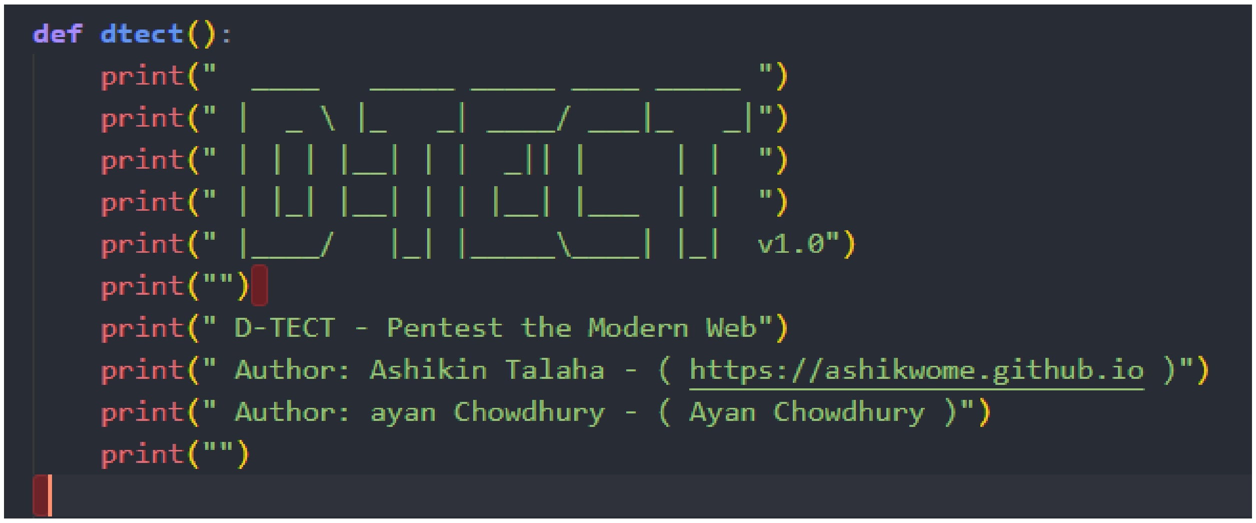
These are the color and format attribute that was made before start writing any functions



```python
# -- Colors Start --
boldwhite       =       Style.BRIGHT+Fore.WHITE
boldgrey        =       Style.DIM+Fore.WHITE
reset           =       Style.RESET_ALL
green           =       Style.BRIGHT+Fore.GREEN
lightgreen      =       Fore.GREEN
red             =       Fore.RED
boldred         =       Style.BRIGHT+Fore.RED
# -- Colors End --
```

Figure 4. 5: color Attributes

4.2.2 d-tect.py: Function dtect()



```python
def dtect():
    print("  ____   _____ _____ ____ _____ ")
    print(" |  _ \ |_   _| ____/ ___|_   _|")
    print(" | | | | |__| | |  _|| |     | |  ")
    print(" | |_| | |__| | | |__| |___  | |  ")
    print(" |____/   |_| |_____\____| |_|  v1.0")
    print("")
    print(" D-TECT - Pentest the Modern Web")
    print(" Author: Ashikin Talaha - ( https://ashikwome.github.io )")
    print(" Author: ayan Chowdhury - ( Ayan Chowdhury )")
    print("")
```

Figure 4. 6: Authors

This function contains nothing but the print functions of the code through printing the statement we showing the owners of this project, name, and version as this is our first finished project so we have declared this as version one.

4.2.3 d-tect.py: Function menu ()

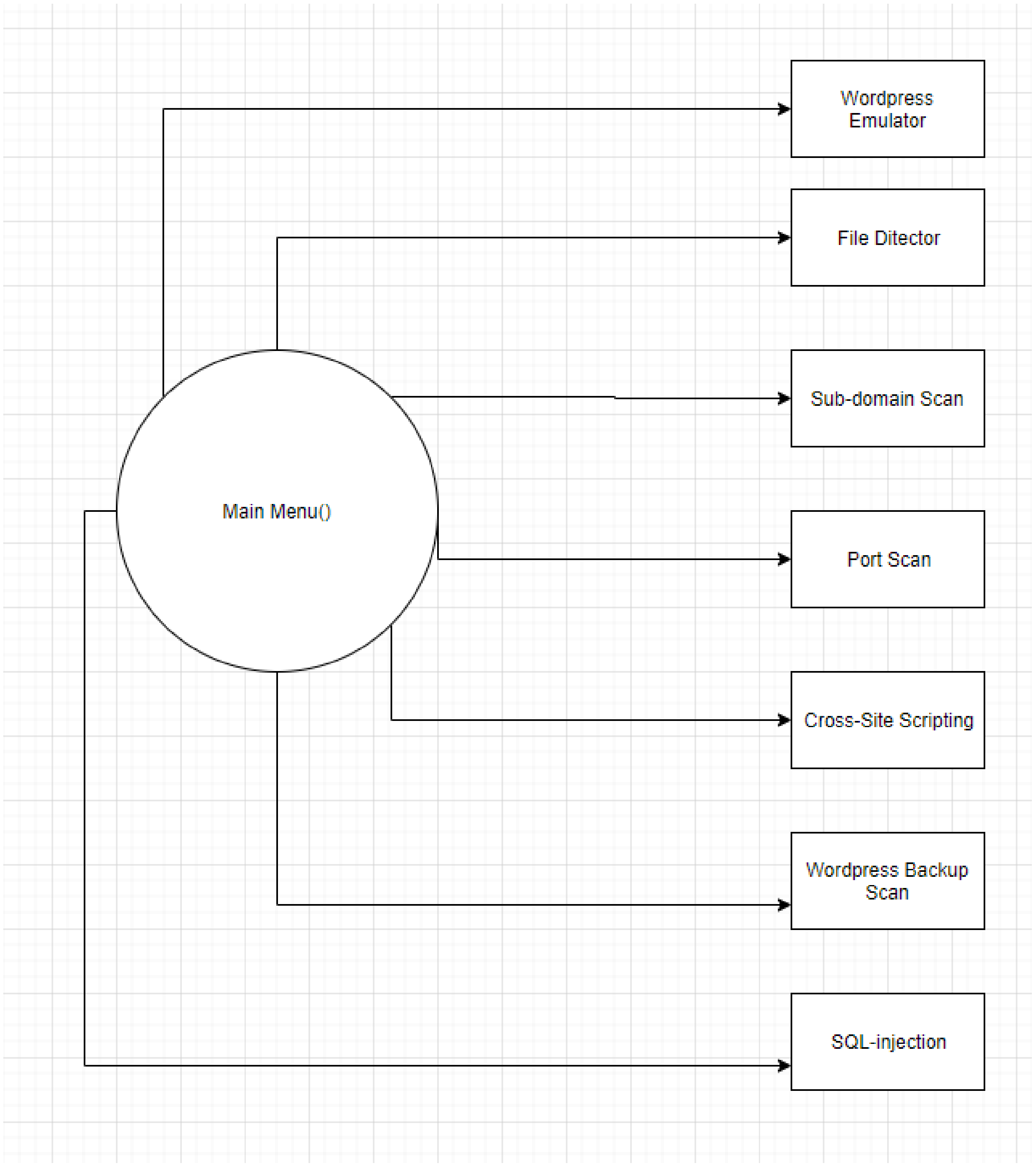
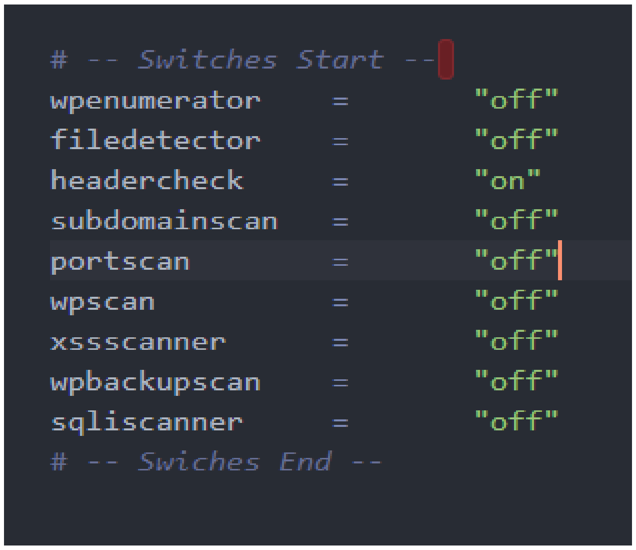


Figure 4. 7: Menu

This is the menu function. Here we are printing or showing the user the number of options he or she might get from this app. Before starting any function, we have created switches and set



```
# -- Switches Start --
wpenumerator    =       "off"
filedetector    =       "off"
headercheck     =       "on"
subdomainscan   =       "off"
portscan        =       "off"
wpscan          =       "off"
xssscanner      =       "off"
wpbackupscan    =       "off"
sqliscanner     =       "off"
# -- Swiches End --
```

Figure 4. 8: Switches

them in ‘off’ condition like shown in figure 4.4. In this function, we are taking input from the user and with the selection number, we are setting the switch to on mode. With one of condition seven elif condition and one else condition. In the else condition we have set a print request for incorrect input and called the menu function again which will start the app from the beginning again.

For example, if anyone selects option ‘1’ then it will start the word-press enumerator then in the background this app will start scrapping the given URL and will start to evaluate that scrap. We will see the proper function in the later segment of this paper [24].

4.2.4 d-tect.py: Function sock(i,secrectswitch=0)

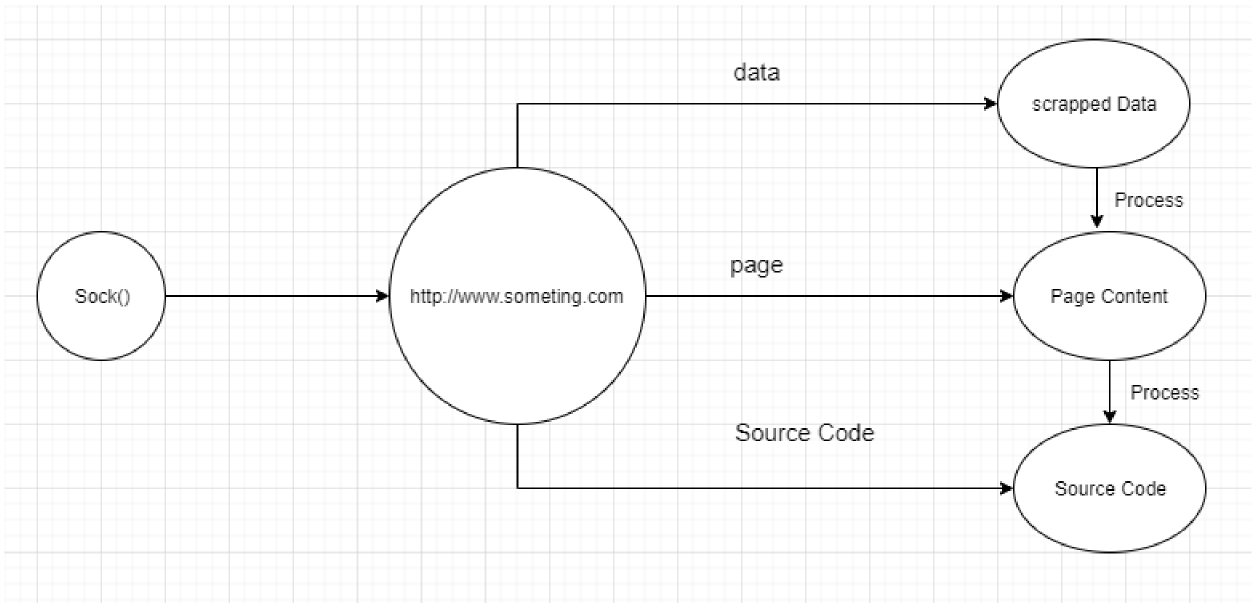


Figure 4. 9: Socket

In this function, we are passing two arguments one is the indicator and another one is a variable which value is initially set to 0. We are doing this so for using the output of the given code below.

```
def sock(i,secretswitch=0):

        secret = secretswitch

        global data,page,sourcecode

        if redirect == 1:

        data = host+i

        else:

        data = host.strip("/")+'/'+i
```

It’s a string type variable. [several data types of python: ref 4.4]

Now in the secret variable, we have the secret-switch (still null at the beginning after writing the whole code it will only contain the above-written code). In the next line there three global variables which I need as the scrapped data.

In the first, if condition “redirect == 1” means the URL is working. Then the URL will be placed in the data variable added with our indicator which was null till now.

In the else condition we are stripping the URL with the breakpoint set to ‘/’. And by manually adding ‘/’ and ‘ I ’ with it.

Either way, we are getting the data in the page variable. The content of the URL is temporarily opened for reading.

In the source-code variable, we have read the data temporarily. That means if we print this variable twice then it will print the content first time and It will print null for the second print.

Now we must have some data so our secret variable won’t be null. After checking it the function will return the source code which is our scrapped website.

4.2.5 d-tect.py: Function cloudflare()

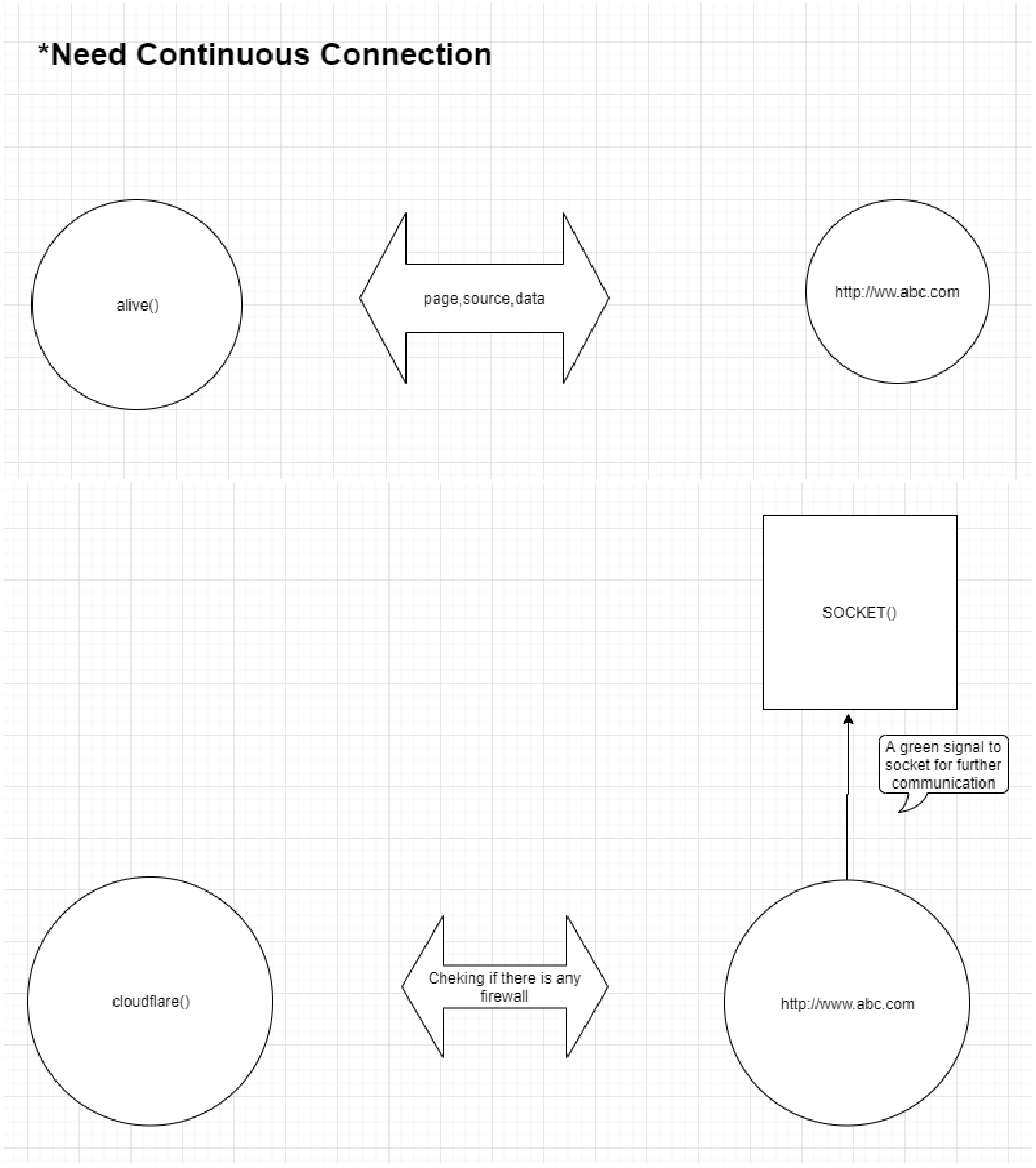

*Need Continuous Connection
alive()
page,source,data
http://ww.abc.com
SOCKET()
A green signal to socket for further communication
cloudflare()
Cheking if there is any firewall
http://www.abc.com

Figure 4. 10: Cloud-Flare

Cloud flare is a web-application security company [ref: 4.4]. In this function, we are searching, if the website is blocked by them or not. Like the above function, we are just reading the file and checking if cloud-flare is in the title to block the IP or not. If it's there then we are giving a print statement and calling the **again()[**chapter:4 – 4.2. **]** function [25].

4.2.6 d-tect.py: Function alive()

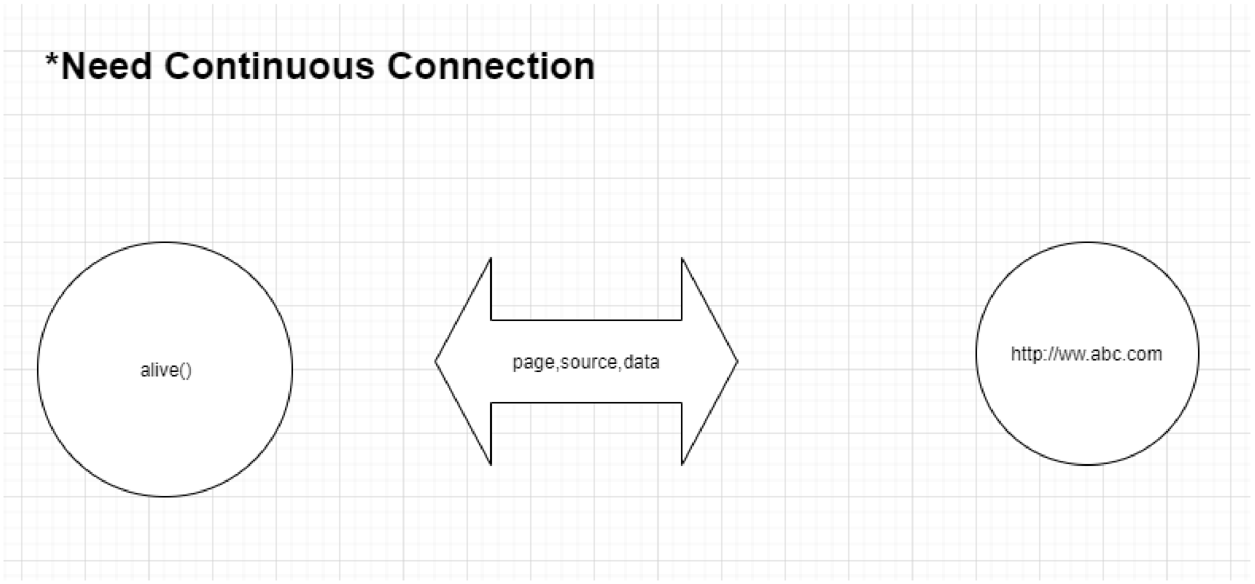


Figure 4. 11: Alive

This function has created to check if the URL or the site is alive or not. We writing the next block of code in try-except block cause sometimes there might be an error.

The possible output of the above code is shown below in figure 4.5. First, there are three global variable pages, split host and IP. In data, we have the host address and in the page variable, we have the opened page. And by page.read we are putting them in the source variable. In the split host, we are putting the host address by splitting them and the scale is ':' and "//" sign. In the IP variable, we are getting the IP of the host using a socket.

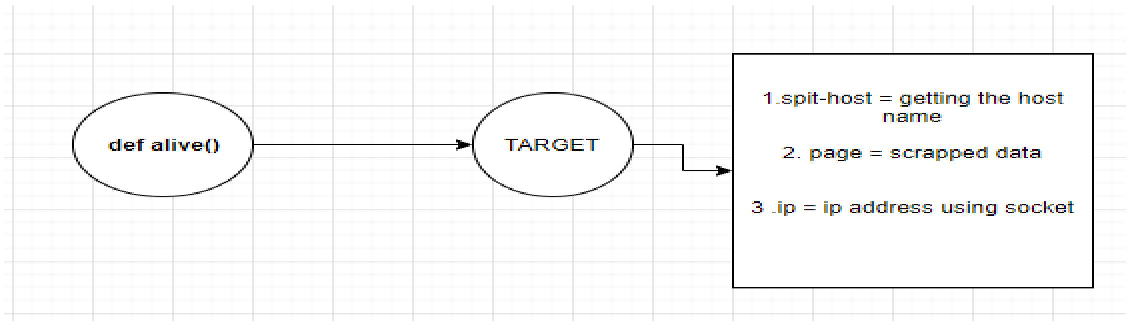

def alive()
TARGET
1.spit-host = getting the host name
2. page = scrapped data
3 .ip = ip address using socket

Figure 4. 12: alive function output

If all of the above requirements are complete then the program will give two statements one will give the green flag which means the site is up. In the next print statement, the program is showing that after being connected to the server it will first check if the site is blocked by cloud-flare it will complete the task by calling the ‘cloudflare()[4.2.2]’ function. And also there is another function call for redirecting check [26].

4.2.7 d-tect.py: Function responseheadercheck()

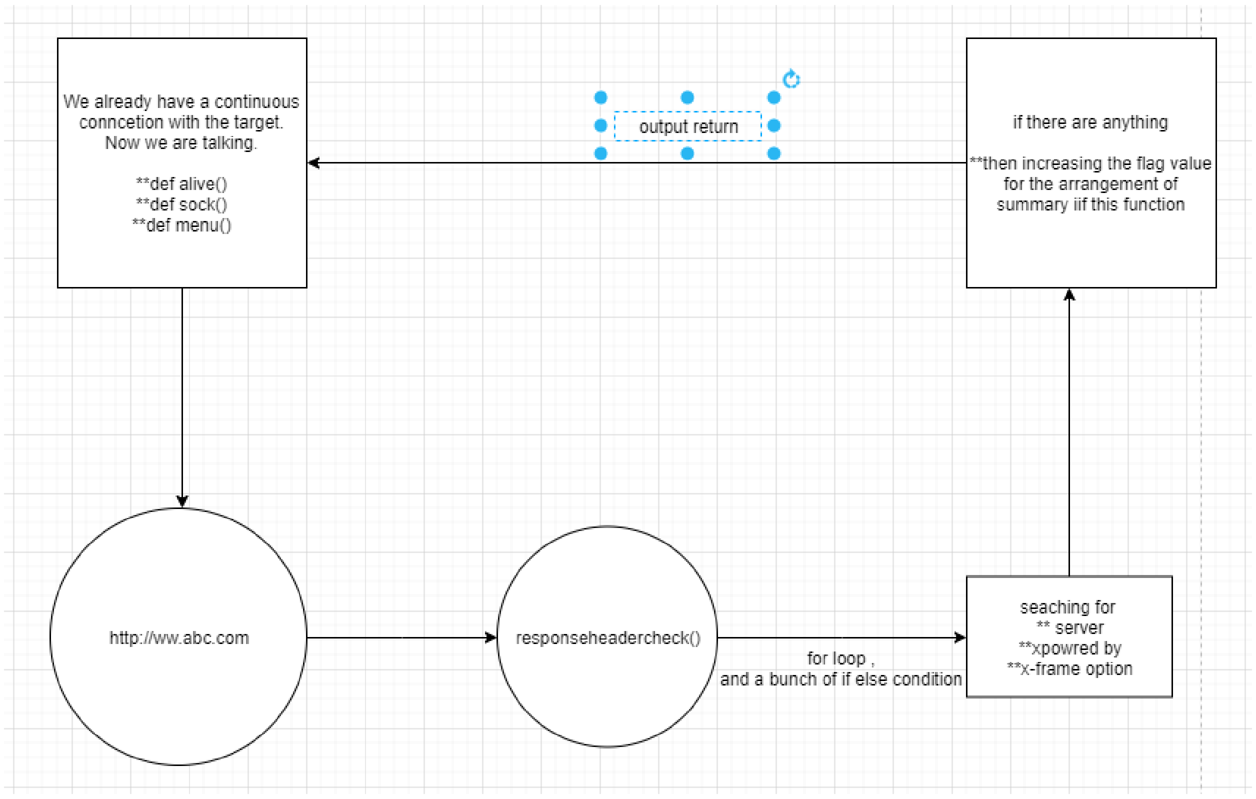

We already have a continuous conncetion with the target. Now we are talking.
**def alive()
**def sock()
**def menu()
output return
if there are anything
**then increasing the flag value for the arrangement of summary iif this function
http://ww.abc.com
responseheadercheck()
for loop ,
and a bunch of if else condition
seaching for
** server
**xpowred by
**x-frame option

Figure 4. 13: Response-Header

This is one of the big functions of the program. It’s mainly for checking the responsive headers that we getting by scrapping the given URL. First, we are printing an empty string to differentiate this function from the other functions. First, we have a self-made dictionary. Which contains some keywords. These are the names of some serious files location. If we are able to find any of these files then this website might be vulnerable. Cause these files contains some serious information that might lead to click-jacking or cross-site scripting.

Then there 3 empty string and a handler ‘cj’ which value is set to 0. In the for loop we are iterating through the page [global variable created in 4.2.2] headers if there are any lower cases in I then it will pass them. Then if I find anything named server then it will reformat it into string in structure variable. Then it will save them in the empty list ‘headersfound’ then in the next line we are putting the server and appended them in the ‘interesting’ list. Here we are not only finding the vulnerability we are also specifying them for making a report. Which makes this app really very special. In the other two else-if conditions, on the first one, we are doing the same thing as before just we are doing it for x-frame this time. In the other else if condition we are just searching for x-frame-options if any is available then we will set the value cj=1. If none of them is found then there is the else condition in which we are just reformatting the URL and putting the reformatted URL in the ‘headersfound’ variable.

At the end of the loop, there is an if condition it is checking if ‘cj=0’ then there are no x-frame in this web-application which is a weakness, so we are reporting that to a user and also providing a link where he or she can know more about this vulnerability.

After all that, in the end of the function we are just printing the interesting headers that we have founded.

4.2.8 d-tect.py: Function parameterarrange(payload)

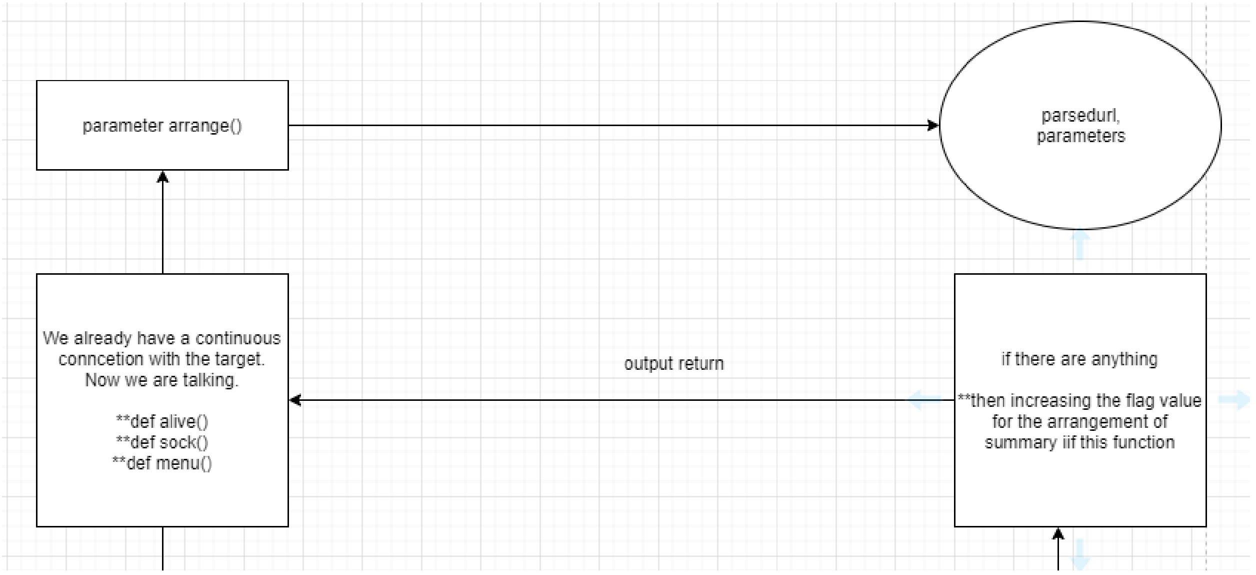

parameter arrange()
parsedurl, parameters
We already have a continuous conncetion with the target. Now we are talking.
**def alive()
**def sock()
**def menu()
output return
if there are anything
**then increasing the flag value for the arrangement of summary iif this function

Figure 4. 14: parameter-arrange

This function is for the arrangement of some parameters to use as payload. In the ‘parsedurl’ variable we have the given host and, in the parameters, we are parsing as qsl. ‘qsl’ actually Parse a query string given as a string argument (data of type *application/x-www-form-urlencoded*). Data are returned as a list of names, value pairs [27].

There are two blank lists parameter’s names and parameter’s values. In the next line there is a loop in which we are searching for ‘m’ in parameters [query]. We are appending first value in the 0 index and the second value is 1st index.

Then in the next loop we searching for ‘n’. While finding this there might throw an error so we are putting it in a try and except block.

In the ‘payload’ variable we have the index of the parameter-values. And we are returning the encoded data as a zipped dictionary. In the ‘except’ block we are just passing them.

4.2.9 d-tect.py: Function SQLIscan(site)

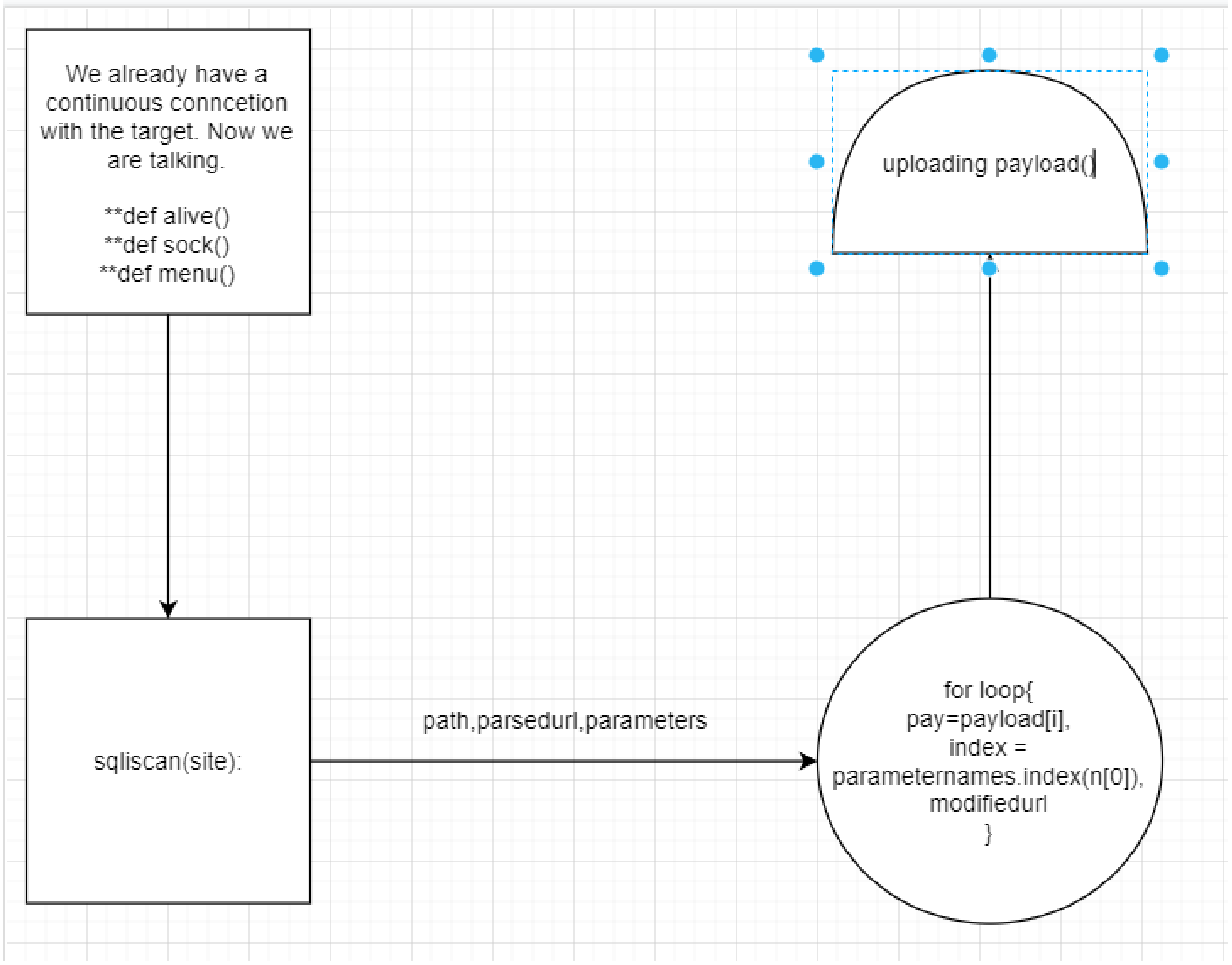

We already have a continuous conncetion with the target. Now we are talking.
**def alive()
**def sock()
**def menu()
uploading payload()
sqliscan(site):
path,parsedurl,parameters
for loop{
pay=payload[i],
index =
parameternames.index(n[0]),
modifiedurl
}

Figure 4. 15: SQL-Scan

This function is another big-function of our project. This part finds out whether the targeted website is vulnerable to SQL-injection or not. This function does this by scraping and using payloads.

First there is a list named vuln, in this, we will have the faulty parameters which we will find after scrapping the website. And we will use the list for further use. Then there are two tuples named payloads and errors.

These will the tools that we will need to make this function work. First, we have a path variable in which we getting three things ‘**SCHEME**’, ‘ **NETLOC**’, ‘**PATH**’.

Scheme:- it provides the information of the URL scheme**.** The scheme can be anything either HTTP or https. If it's on HTTP that means it's on port number 80. And if it's on https that means the port number 443.

Netloc:- it will give the information about the network location.

PATH:- Hierarchical path is the topology information.

We are storing all of these in our path variable. Now in the next line of codes, we are going to use this information against this target.

In the 'parsedurl' we are opening the host. Now there is an interesting point here. In the 'parameters' variable, we are using 'parse_qsl'.

It passes a URL as text and takes a return in a dictionary format. We are setting the blank values as True because if don't do that then qsl will automatically add a '+' sign for blank values like space.

Then we have two empty lists. Then using a for-loop we are appending values to that two lists in the first list we have the parameter's name and in the second list, we have parameter's value.

In for loop, we are iterating through the parameters variable. In this variable we have the dictionary of the URL. At first, we have a counter named found and the value is set to 0. Then we are again iterating through our payloads. Then in the 'modified' URL, we are uploading the vulnerable payload or the SQL-injection command. If it's vulnerable or not. Then in the source variable, we are reading the updated URL.

This process might occur an error so we are putting this whole code in a try-except for preventing the app from being stopped.

In for loop we are finding the vulnerability if we are able to find any then we will set the value of the found counter to 1.

Now if the found counter has the value of 1 then the website is vulnerable and we are printing the vulnerable parameters in the 'vuln' list. If the vuln parameter has a value of 0 then it means the website is not vulnerable. So we are printing in the else condition "Not Vulnerable".

4.2.10 d-tect.py: Function XSSscan(site)

In this section we are checking if the given website is vulnerable to XSS or cross-site-scripting.

This function is written in the same way as the previous function (sql-scan). Here we have a different payload in the payload's variable. We are scrapping the given website using the same method parse qsl. Then in the first loop, we are appending the parameter in the name list and the values in the values list.

In the next loop, we are adding the payloads with the scrap in the payload variable and using the u variable to upload the payloads. In the source, we have the read file of the URL. And now we are using Beautiful-soup. Because we are searching for cross-site scripting vulnerability this vulnerability could be found in the code of the website like it could be hidden in any HTML tag or a JavaScript function. So, we are now using beautiful soup so that we can find any special attribute or tag or a script.

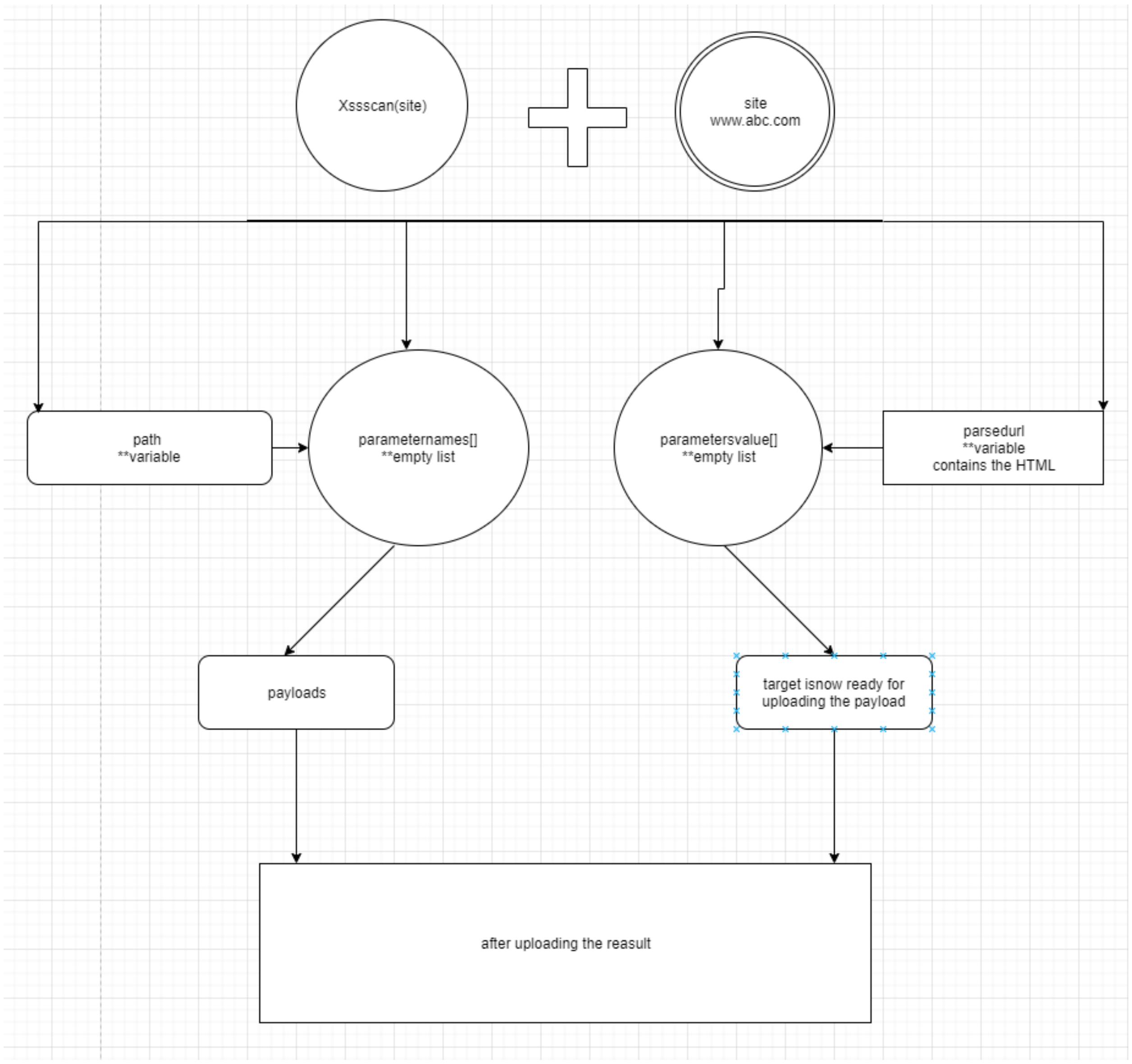


Figure 4. 16: XSS scan

Now in if condition we searching for the payloads. Here we are using str for this purpose because it will create a new string object from the given object, if any encoding or error is specified then it will expose the data buffer that will decode using the given encoding and error handler and here in our case we have specified the error handler and encoding as ‘u’ and ‘payload’ variable.

In if condition we are checking the HTML content. And if it can find any vulnerability then it will set the found counter as 1.

In the script we are searching for all the script tag because it contains the JavaScript codes. Now we are checking them and searching for a XSS vulnerability. If there is any then it will also show the vulnerable script from HTML or JS.

In chapter two [2.3.6] section we have already declared the theory of cross-site scripting. The main types of cross-site-scripting are stored and reflected. In our project, we have used reflected scripting as payloads.

Here our main target is to find out any vulnerability not to hack the site. Reflected scripting does not change anything permanently. It will start a session and will give the attacker a new view. And this view will contain some information. The items of the information will be specified in the given payload.

Here in our project we have used this given command for our XSS scan

```
 '3':'d4rk();"\'\\/}{d4rk',

'2':'d4rk</script><script>alert(1)</script>d4rk',

'1':'<d4rk>'
```

In this payload we are trying to show a message on the vulnerable page. Which is temporary. It will only appear after executing the code. ‘d4kr’ is a simple string type. Its job is to call upon a sensitive file.

This is an app, if it stops running then it will create a big problem. So, there is a bunch of if-else condition for handling that.

The whole code is in a try-except block for ignoring the errors[28].

4.2.11 d-tect.py: Function portscanner()

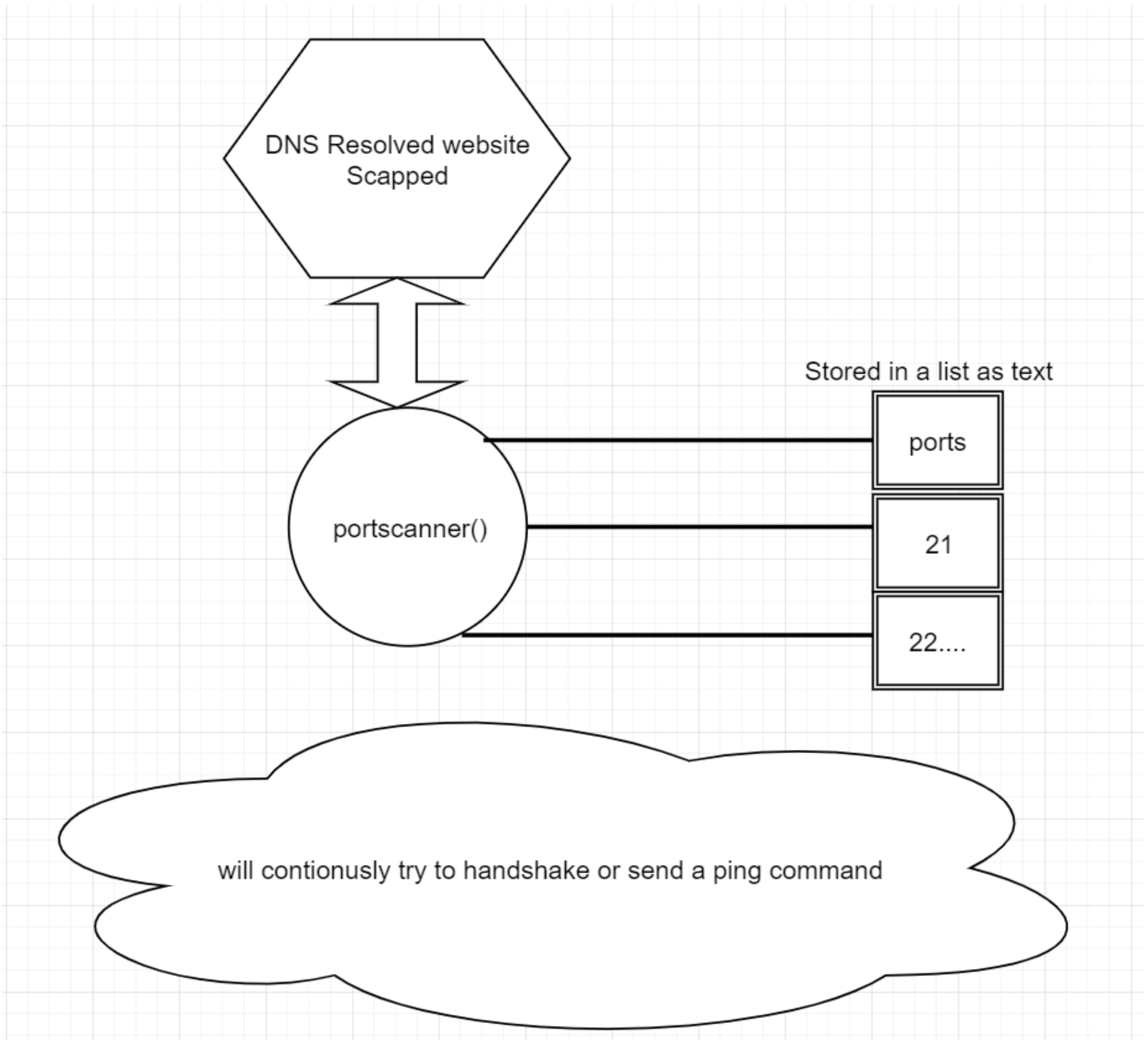

DNS Resolved website Scapped
Stored in a list as text
ports
portscanner()
21
22....
will contionusly try to handshake or send a ping command

Figure 4. 17: Port Scan-One

In this function, we will scan all the ports that are connected to the targeted host.

First, there is a print statement that will show the user how to conduct the scanning using this app. Then there are three variables ‘was multiple’ ‘was range’ ‘was one’.

The first one is for multiple users and the second one is for scanning in a range and the last one is for scanning one single port.

Here the proportion keyword is used as the user input. If there is any ‘,’ comma that means the user has given multiple port numbers for scanning. Then in if the condition it will split the given input by comma and store that in the ‘multiple-port’ variable. Now it will check if it’s a valid port number or not by checking if the given input is stored as a string object whether it has a digit or not. We are doing the same for all other scans. Like all ports or single port or a given range of ports.

Here the user has the option to select all the ports for scanning. Normally in this part, he or she will only be able to scan ports starting from 20 and ends in 5000 for more than 5000 ports the user will have to manually enter the range or the port number only. In a computer system normally there 65535 ports. But we are conducting this range of port because ports before 20 numbers are normally occupied by various services. They normally don’t have any vulnerability.

Ports are the main equipment that computers used for establishing a connection with any other host. Any port can be vulnerable or all 65535 ports can be secure. But on 21,22,23,25 or 53 etc. ports can be vulnerable. All these ports are used for the establishment of a remote connection. There can be many types of vulnerability on a port. Like broken authentication or open port or any already existed backdoor.

How a port is kept open?

```
import socket

my_sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

my_sock.connect(('www.py4inf.com',80))

my_sock.send(b'GET /code/romeo.txt HTTP/1.0\r\nHost: www.py4inf.com\r\n\r\n')


while True:

    data = my_sock.recv(1024)

    if len(data)<1:

        break

    print(data)

my_sock.close()
```

The codes given in here is a simple code for downloading an text from a web site. We have established the connection using sockets. At the end of the code there is a command for closing the connection. It means if any connection is open then it needs to be closed after finishing the job. Just stop using the service without closing keeps the port opened. This can make it a vulnerable point for an attacker to attack.

Now after checking the validity of the given input we are trying to connect to each of the port by using socket. In the ‘portconnect’ variable we have the connected socket.

Here we have only checked the open ports.

Now in response, we have ‘connect_ex’ it mainly connects to a remote address and then wrap the connection in a ‘SSL’ shell for secure connection, if app can make connection with any of the given port number then it will increase the flag ‘found’ value 1, in the same process it will iterate through all the given port numbers and will show the output using a ‘print’ statement.

4.2.12 d-tect.py: Function subdomainscanner()

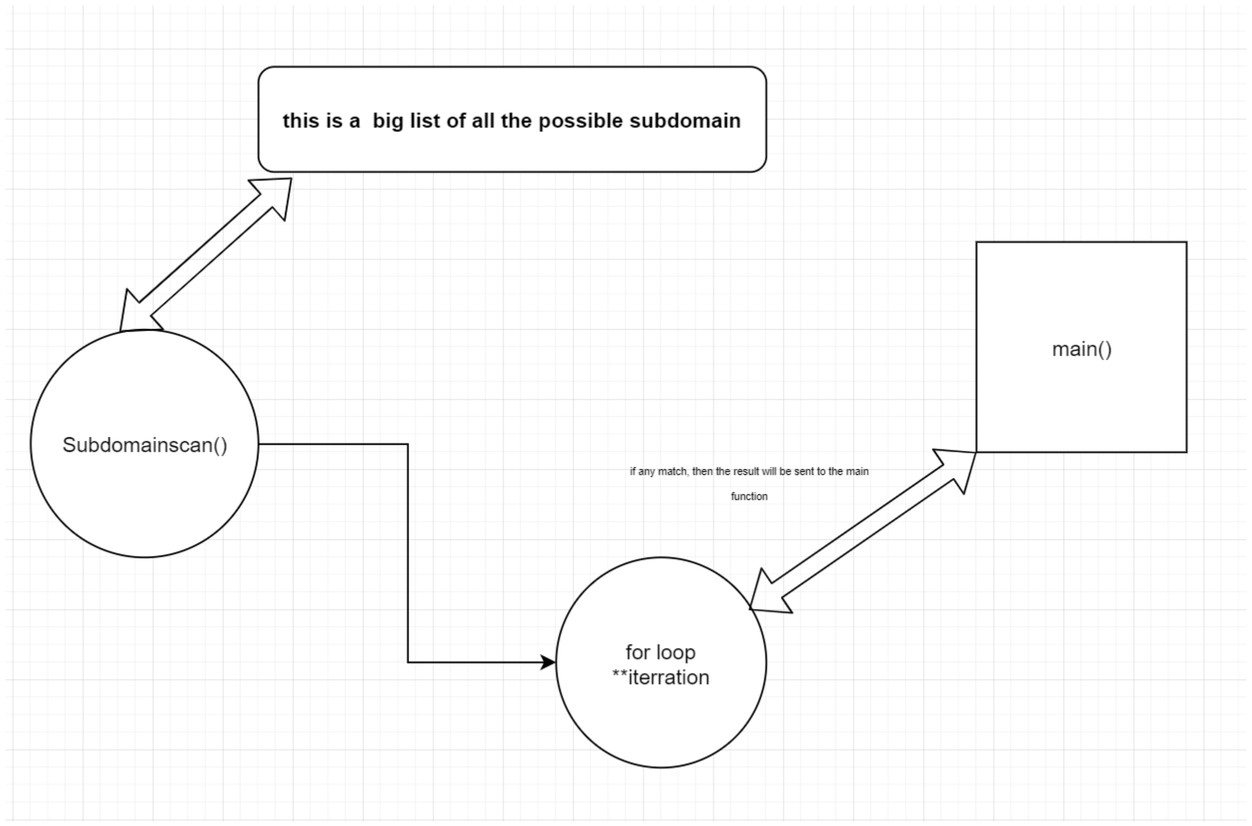

this is a big list of all the possible subdomain
main()
Subdomainscan()
if any match, then the result will be sent to the main function
for loop
**iterration

Figure 4. 18: Subdomain

Here we have a dictionary of words which contains name of the possible subdomains. Examples

"mail","localhost","blog","forum","0","01","02","03","1","10","11","12","13","14","15","16","17","18","19","2","20","3","3com","4","5","6","7","8","9","ILMI","a","a.auth-ns","a01","a02","a1","a2","abc","about","ac","academico","acceso","access","accounting","accounts","acid","activestat","ad","adam","adkit","admin","administracion","administrador","administrator","administrators","admins","ads","adserver","adsl","ae","af","affiliate","affiliates","afiliados","ag","agenda","agent","ai","aix","ajax","ak","akamai","al","alabama","alaska","albuquerque","alerts","alpha","alterwind","am","amarillo","americas","an","anaheim","analyzer","announce","announcements","antivirus","ao","ap","apache","apollo","app","app01","app1","apple","application","applications","apps","appserver","aq","ar","archie","arcsight","argentina","arizona","arkansas","arlington","as","as400","asia","asterix","at","athena","atlanta","atlas","att","au","auction"

The progress flag is set to 0. Iterating i[url] through with the word list if it can find any match then at first. Then it will spilt the host and resolved it into ip addresses.

After just resolving the subdomain and showing the IP address to the host our program will move on and start searching for the next sub-domain.

These were the main scanning functions that were used for making the scanning shell app work.

There are some other functions used for maintaining the whole working process.

Like word-press emulator, sensitive file exposer, re-direction check, etc. functions work with the output of the main function.

In the SQL-injection function, we have already scrapped a given URL and processed a lot for further use. Like while iterating through the URL for SQL-injection vulnerability we have already scrapped and read through specific.

But in python, all scrapping is done by using '.read()' can only be written once to resolve this problem after printing the final result we have returned the final output that means the processed data. And they all are global variables so we can call them anywhere we want.

Most of the vulnerabilities of web-applications can be found by scrapping them.

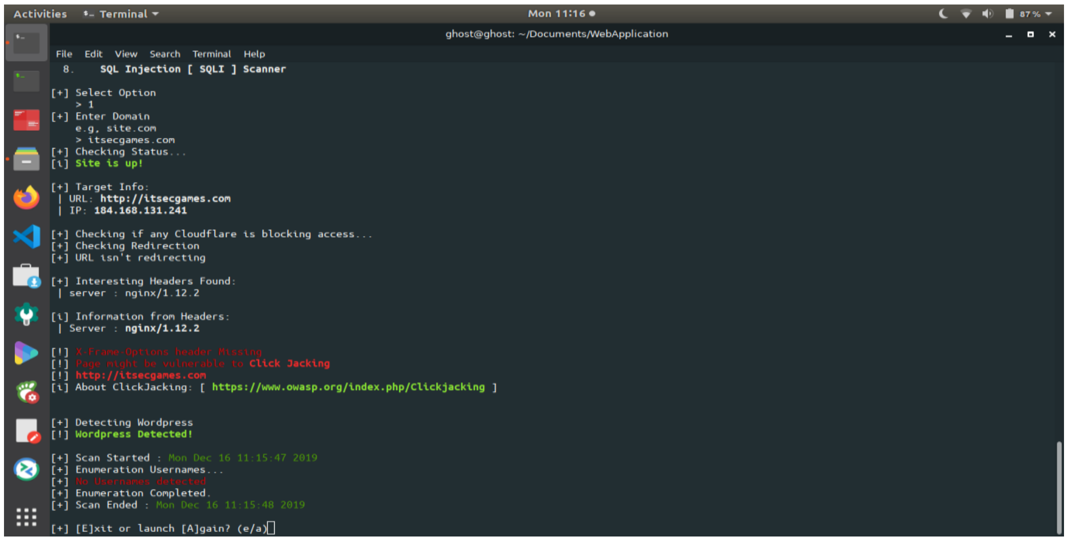
There is one another main Module 'moduleBS.py'. In this function, we have mainly used the Beautiful-soup module and regular expression [re] module. In this function, we scrapped the website on some special patters. Like In an HTML document their specific subdomain links wrapped under '<href> … </href>' tag. Sometimes they are encoded and sometimes they are not. A real-life example of such a subdomain is a forgotten password URL.

It may conation some special password or user session details. If we can get that information using our app then an attacker can create a fake session with is skills. So we have searched for them by converting them into strings and comparing them with certain other functions.

There are lots of static data used in dictionaries of several functions. This might make this app somewhat limited but hacking without data or information is impossible.

The moduleBS.py module can't be described due to privacy

4.3 Result:



```
 8.    SQL Injection [ SQLI ] Scanner

[+] Select Option
    > 1
[+] Enter Domain
    e.g, site.com
    > itsecgames.com
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://itsecgames.com
 | IP: 184.168.131.241

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[+] URL isn't redirecting

[+] Interesting Headers Found:
 | server : nginx/1.12.2

[i] Information from Headers:
 | Server : nginx/1.12.2

[!] X-Frame-Options header Missing
[!] Page might be vulnerable to Click Jacking
[!] http://itsecgames.com
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]


[+] Detecting Wordpress
[!] Wordpress Detected!

[+] Scan Started : Mon Dec 16 11:15:47 2019
[+] Enumeration Usernames...
[+] No Usernames detected
[+] Enumeration Completed.
[+] Scan Ended : Mon Dec 16 11:15:48 2019

[+] [E]xit or launch [A]gain? (e/a)
```

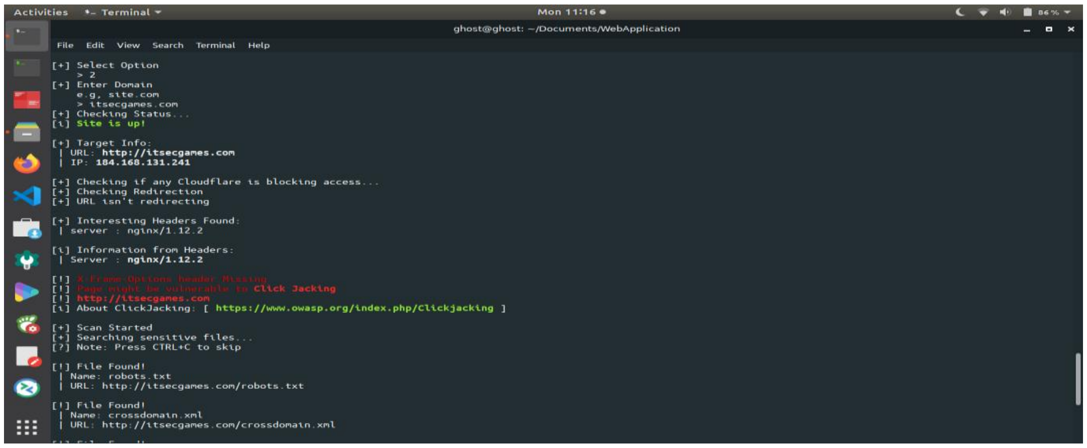
Figure 4. 19: Scan – 1

Here we are checking the header of the given website 'itsecgames.com'. This website is lab equipment for testing.

First, the URL is resolved into the IP address. From the previous theory [4.2.4] we already know about the 'cloudflare' part. These functions are found in the headers.

It also found two different servers connected to the given URL. And also detected the word-press. In which the x-frame header is missing. Which means the header files are not encrypted.

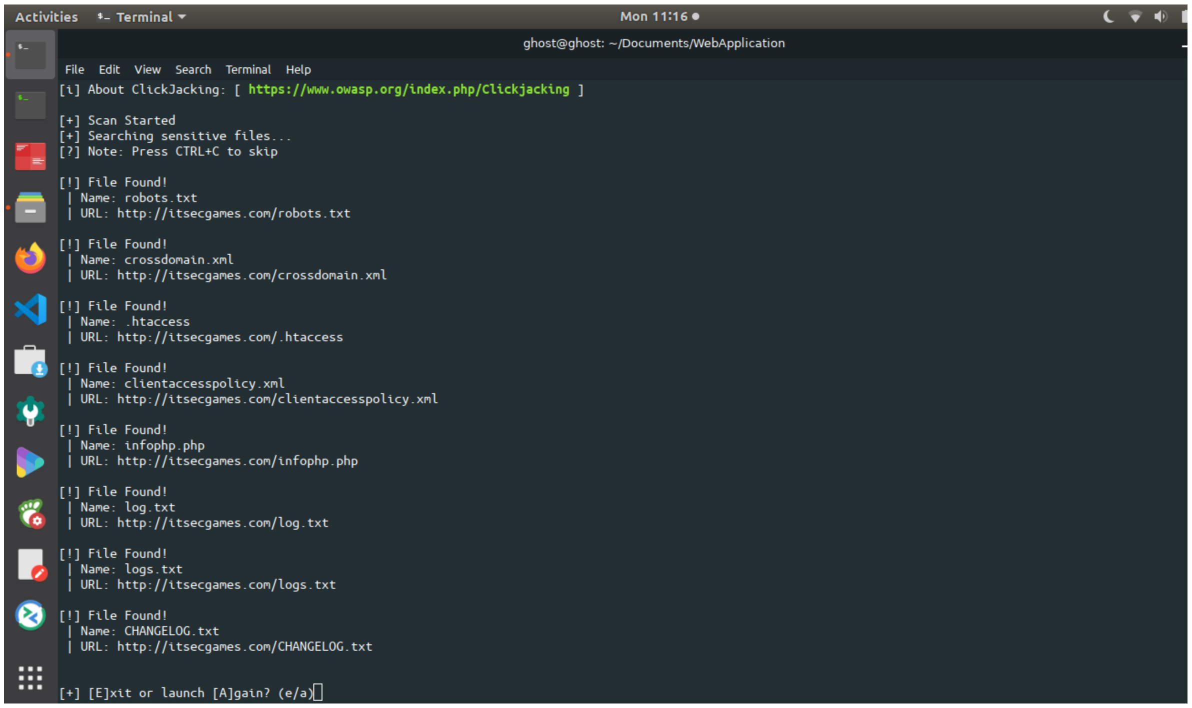
This might make the website vulnerable to click-jacking.



```
[+] Select Option
    > 2
[+] Enter Domain
    e.g, site.com
    > itsecgames.com
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://itsecgames.com
 | IP: 184.168.131.241

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[+] URL isn't redirecting

[+] Interesting Headers Found:
 | server : nginx/1.12.2

[i] Information from Headers:
 | Server : nginx/1.12.2

[!] X-Frame-Options header Missing
[!] Page might be vulnerable to Click Jacking
[!] http://itsecgames.com
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]

[+] Scan Started
[+] Searching sensitive files...
[?] Note: Press CTRL+C to skip

[!] File Found!
 | Name: robots.txt
 | URL: http://itsecgames.com/robots.txt

[!] File Found!
 | Name: crossdomain.xml
 | URL: http://itsecgames.com/crossdomain.xml
```

Figure 4. 20: Scan- 2

Now we checking if there are any sensitive files in the same URL. This time the app found two different sensitive files one if robost.txt and crossdomain.xml both of the files have serious importance. The first one contains configuration and the second one can be used to upload or delete any file from this domain using a third-party client like ‘FileZilla’


Activities
Terminal
Mon 11:16
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]

[+] Scan Started
[+] Searching sensitive files...
[?] Note: Press CTRL+C to skip

[!] File Found!
 | Name: robots.txt
 | URL: http://itsecgames.com/robots.txt

[!] File Found!
 | Name: crossdomain.xml
 | URL: http://itsecgames.com/crossdomain.xml

[!] File Found!
 | Name: .htaccess
 | URL: http://itsecgames.com/.htaccess

[!] File Found!
 | Name: clientaccesspolicy.xml
 | URL: http://itsecgames.com/clientaccesspolicy.xml

[!] File Found!
 | Name: infophp.php
 | URL: http://itsecgames.com/infophp.php

[!] File Found!
 | Name: log.txt
 | URL: http://itsecgames.com/log.txt

[!] File Found!
 | Name: logs.txt
 | URL: http://itsecgames.com/logs.txt

[!] File Found!
 | Name: CHANGELOG.txt
 | URL: http://itsecgames.com/CHANGELOG.txt

[+] [E]xit or launch [A]gain? (e/a)
```

Figure 4. 21: Scan – 2

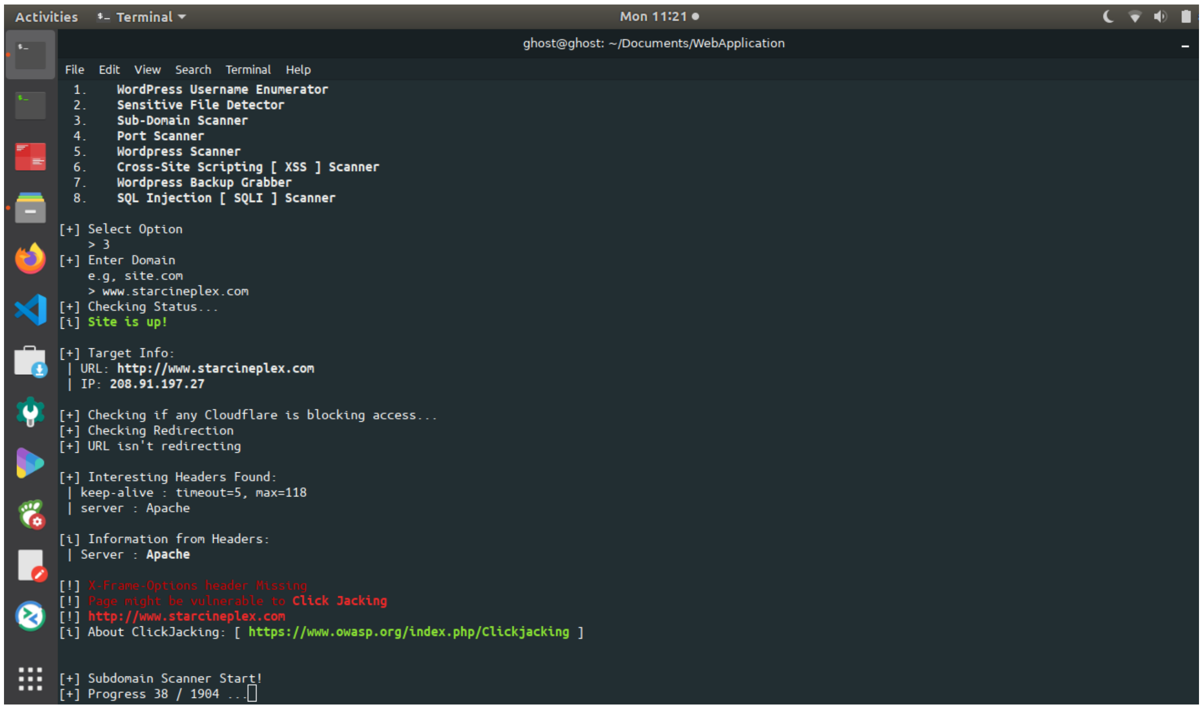
These are some other sensitive files.

.htaccess ➔ file is used to remotely connect any client with the server.

Clientaccesspolicy.xml ➔ Here we can change and modify the client access limitations.

Info.php ➔ it can be used to break the authentication policy.

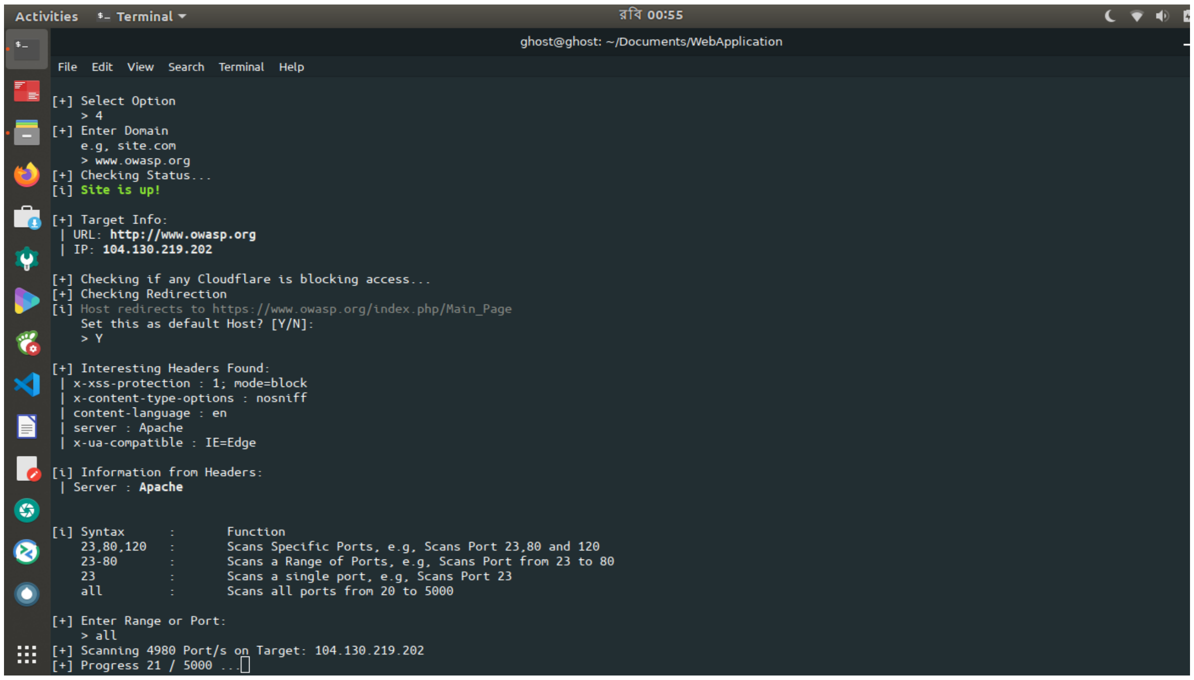
Besides log and change log have a list of changes that were made throughout the time for this website.


Activities
Terminal
Mon 11:21
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
  1.    WordPress Username Enumerator
  2.    Sensitive File Detector
  3.    Sub-Domain Scanner
  4.    Port Scanner
  5.    Wordpress Scanner
  6.    Cross-Site Scripting [ XSS ] Scanner
  7.    Wordpress Backup Grabber
  8.    SQL Injection [ SQLI ] Scanner

[+] Select Option
    > 3
[+] Enter Domain
    e.g, site.com
    > www.starcineplex.com
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://www.starcineplex.com
 | IP: 208.91.197.27

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[+] URL isn't redirecting

[+] Interesting Headers Found:
 | keep-alive : timeout=5, max=118
 | server : Apache

[i] Information from Headers:
 | Server : Apache

[!] X-Frame-Options header Missing
[!] Page might be vulnerable to Click Jacking
[!] http://www.starcineplex.com
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]


[+] Subdomain Scanner Start!
[+] Progress 38 / 1904 ...
```

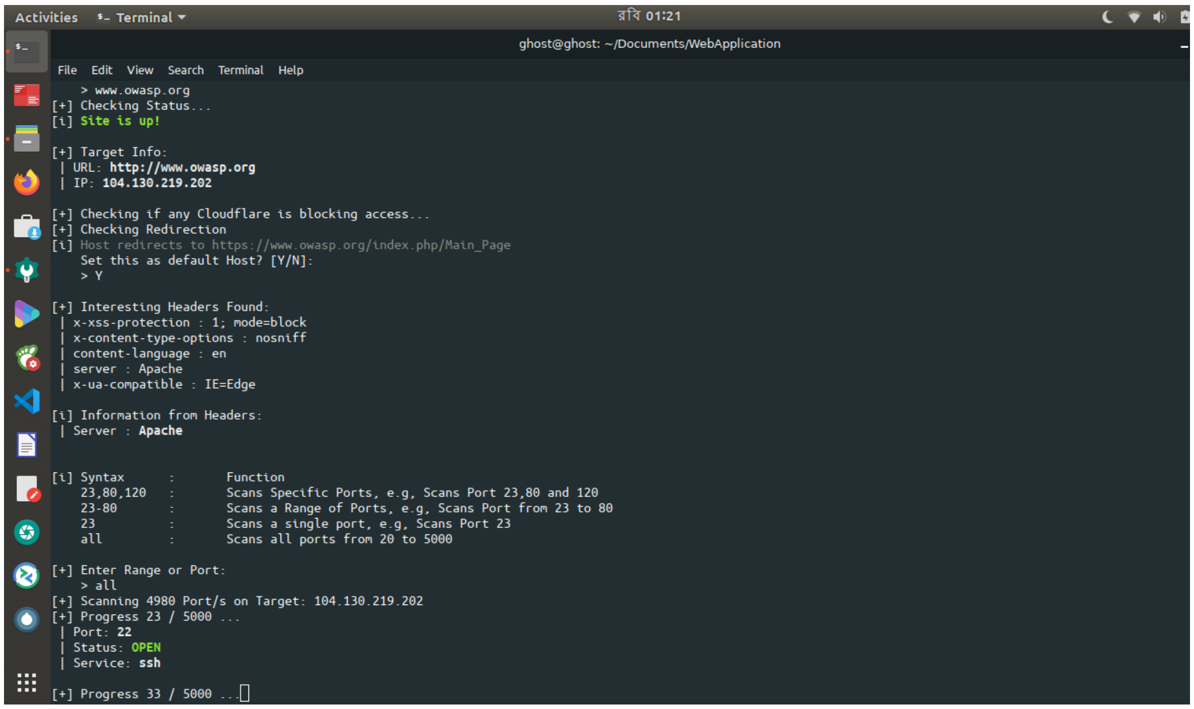
Figure 4. 22: Scan – 3

Here we have checked ‘www.cineplex.com’ as this is a static website so it won’t redirect. So, there are no subdomains still we are searching for subdomains 1904 time.


Activities
Terminal
রবি 00:55
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
[+] Select Option
    > 4
[+] Enter Domain
    e.g, site.com
    > www.owasp.org
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://www.owasp.org
 | IP: 104.130.219.202

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[i] Host redirects to https://www.owasp.org/index.php/Main_Page
    Set this as default Host? [Y/N]:
    > Y

[+] Interesting Headers Found:
 | x-xss-protection : 1; mode=block
 | x-content-type-options : nosniff
 | content-language : en
 | server : Apache
 | x-ua-compatible : IE=Edge

[i] Information from Headers:
 | Server : Apache

[i] Syntax        :      Function
    23,80,120     :      Scans Specific Ports, e.g, Scans Port 23,80 and 120
    23-80         :      Scans a Range of Ports, e.g, Scans Port from 23 to 80
    23            :      Scans a single port, e.g, Scans Port 23
    all           :      Scans all ports from 20 to 5000

[+] Enter Range or Port:
    > all
[+] Scanning 4980 Port/s on Target: 104.130.219.202
[+] Progress 21 / 5000 ...
```

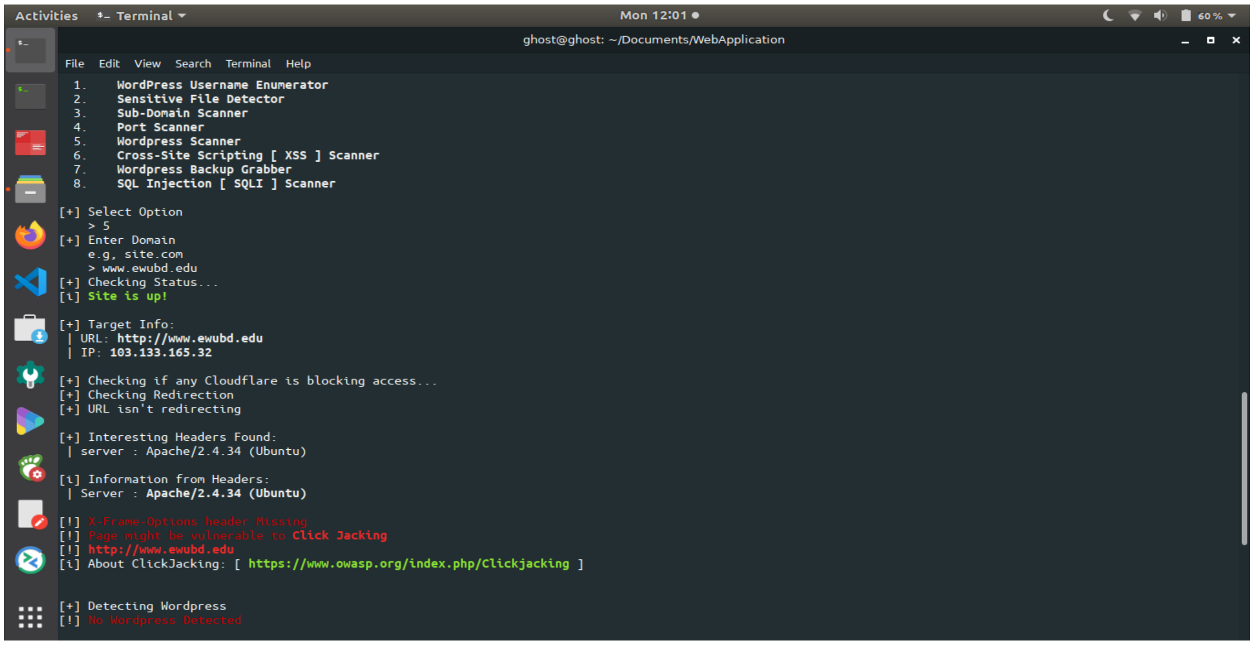
Figure 4. 23: Scan-4


Activities
Terminal
রবি 01:21
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
    > www.owasp.org
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://www.owasp.org
 | IP: 104.130.219.202

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[i] Host redirects to https://www.owasp.org/index.php/Main_Page
    Set this as default Host? [Y/N]:
    > Y

[+] Interesting Headers Found:
 | x-xss-protection : 1; mode=block
 | x-content-type-options : nosniff
 | content-language : en
 | server : Apache
 | x-ua-compatible : IE=Edge

[i] Information from Headers:
 | Server : Apache

[i] Syntax          :       Function
    23,80,120       :       Scans Specific Ports, e.g, Scans Port 23,80 and 120
    23-80           :       Scans a Range of Ports, e.g, Scans Port from 23 to 80
    23              :       Scans a single port, e.g, Scans Port 23
    all             :       Scans all ports from 20 to 5000

[+] Enter Range or Port:
    > all
[+] Scanning 4980 Port/s on Target: 104.130.219.202
[+] Progress 23 / 5000 ...
 | Port: 22
 | Status: OPEN
 | Service: ssh

[+] Progress 33 / 5000 ...
```

Figure 4. 24: Scan-4

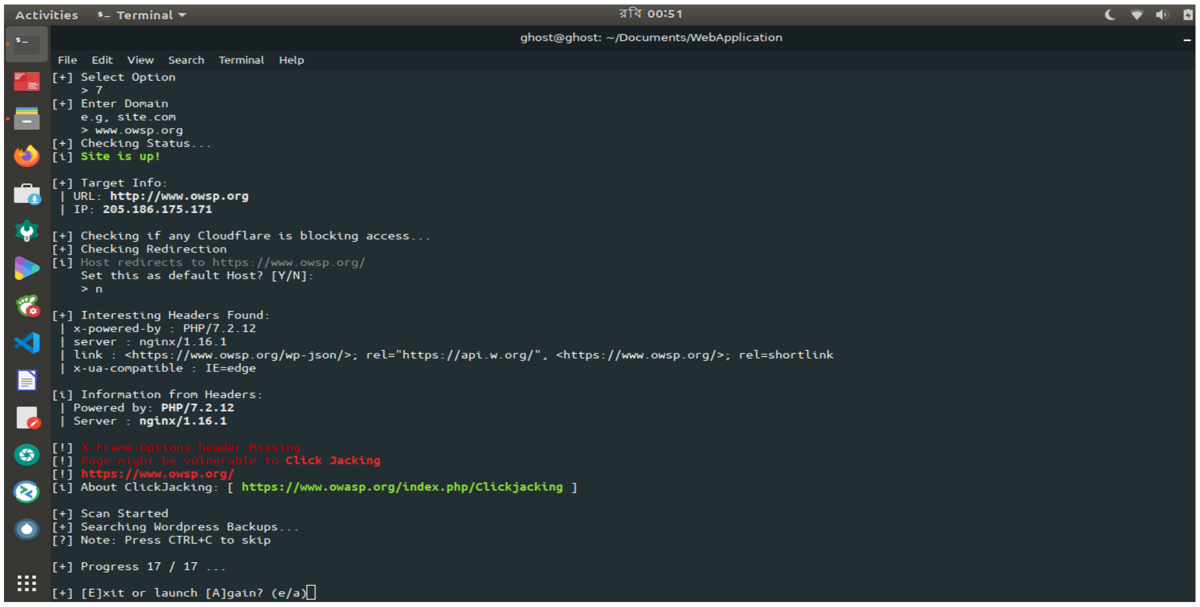
As discussed in the theory we are scanning 5000 ports. Here 22 no port is open which is the ssh port or a secure shell for remote connection. The website is ‘www.owasp.org’ which is also a lab tool.


Activities
Terminal
Mon 12:01
60 %
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
 1.    WordPress Username Enumerator
 2.    Sensitive File Detector
 3.    Sub-Domain Scanner
 4.    Port Scanner
 5.    Wordpress Scanner
 6.    Cross-Site Scripting [ XSS ] Scanner
 7.    Wordpress Backup Grabber
 8.    SQL Injection [ SQLI ] Scanner

[+] Select Option
    > 5
[+] Enter Domain
    e.g, site.com
    > www.ewubd.edu
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://www.ewubd.edu
 | IP: 103.133.165.32

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[+] URL isn't redirecting

[+] Interesting Headers Found:
 | server : Apache/2.4.34 (Ubuntu)

[i] Information from Headers:
 | Server : Apache/2.4.34 (Ubuntu)

[!] X-Frame-Options header Missing
[!] Page might be vulnerable to Click Jacking
[!] http://www.ewubd.edu
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]


[+] Detecting Wordpress
[!] No Wordpress Detected
```

Figure 4. 25: Scan – 5

This another word-press scan here the URL is ‘www.ewubd.edu’ on this website, there is cloud flare which is blocking access and there are no word-press themes and this site is not vulnerable but the important information form this scan is info about the server. This URL is running on Linux based apache server.

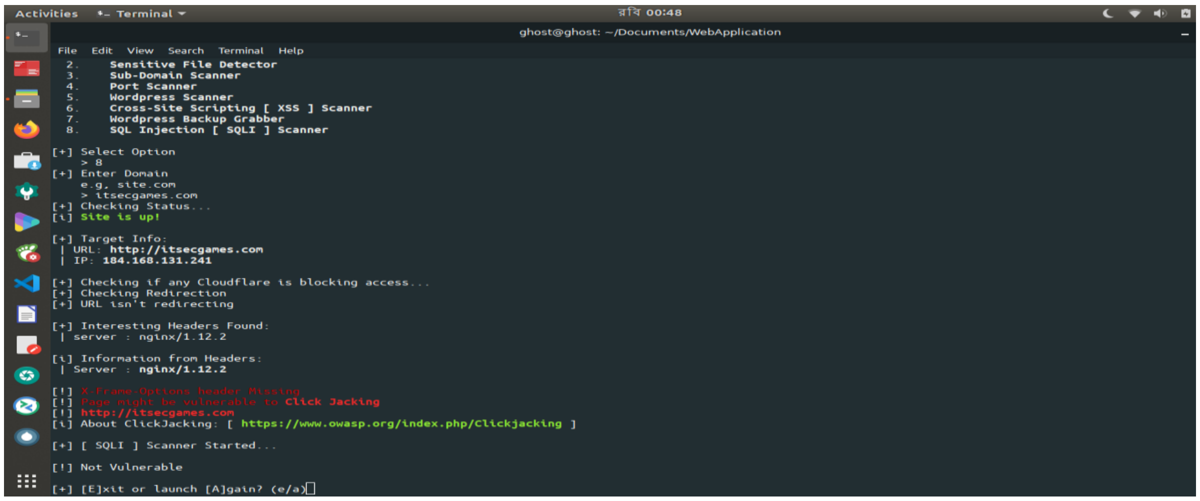


Activities
Terminal
রবি 00:51
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
[+] Select Option
    > 7
[+] Enter Domain
    e.g, site.com
    > www.owsp.org
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://www.owsp.org
 | IP: 205.186.175.171

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[i] Host redirects to https://www.owsp.org/
    Set this as default Host? [Y/N]:
    > n

[+] Interesting Headers Found:
 | x-powered-by : PHP/7.2.12
 | server : nginx/1.16.1
 | link : <https://www.owsp.org/wp-json/>; rel="https://api.w.org/", <https://www.owsp.org/>; rel=shortlink
 | x-ua-compatible : IE=edge

[i] Information from Headers:
 | Powered by: PHP/7.2.12
 | Server : nginx/1.16.1

[!] X-Frame-Options header Missing
[!] Page might be vulnerable to Click Jacking
[!] https://www.owsp.org/
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]

[+] Scan Started
[+] Searching Wordpress Backups...
[?] Note: Press CTRL+C to skip

[+] Progress 17 / 17 ...

[+] [E]xit or launch [A]gain? (e/a)
```

Figure 4. 26: Scan- 6

Here we are doing another word-press scan but now we are searching for the backups of word-press and server information. Here the URL in the link tag will give us the chance to grab the word-press backup.



Activities
Terminal
রবি 00:48
ghost@ghost: ~/Documents/WebApplication
File Edit View Search Terminal Help

```
  2.     Sensitive File Detector
  3.     Sub-Domain Scanner
  4.     Port Scanner
  5.     Wordpress Scanner
  6.     Cross-Site Scripting [ XSS ] Scanner
  7.     Wordpress Backup Grabber
  8.     SQL Injection [ SQLI ] Scanner

[+] Select Option
    > 8
[+] Enter Domain
    e.g, site.com
    > itsecgames.com
[+] Checking Status...
[i] Site is up!

[+] Target Info:
 | URL: http://itsecgames.com
 | IP: 184.168.131.241

[+] Checking if any Cloudflare is blocking access...
[+] Checking Redirection
[+] URL isn't redirecting

[+] Interesting Headers Found:
 | server : nginx/1.12.2

[i] Information from Headers:
 | Server : nginx/1.12.2

[!] X-Frame-Options header Missing
[!] Page might be vulnerable to Click Jacking
[!] http://itsecgames.com
[i] About ClickJacking: [ https://www.owasp.org/index.php/Clickjacking ]

[+] [ SQLI ] Scanner Started...

[!] Not Vulnerable

[+] [E]xit or launch [A]gain? (e/a)
```

Figure 4. 27: Scan – 7

Here we’re checking itsecgames.com for SQL-injection. As this website is a static website with now database to inject so it’s not vulnerable.

But sometimes some websites with SQL-injection vulnerability can’t be detected using this app. The limitations are the payloads.

Chapter 5

Conclusion and Further Development

The web application has become very popular at present. Anyone can communicate and exchange information with each other within a short moment from any corner of the world. With the advancement of web application popularity, the chances of web applications have become a target for getting information. Vulnerabilities help the attacker to gather information and exploit them. Many attempts have been taken to scan web applications and finding vulnerabilities. The tool 'D-tect' can find the eight most serious web application attacks. It can also gather so much information about any kind of web application. This tool can be used for reconnaissance to any particular web application. That means many kinds of information such as ports, database versions, WordPress version, etc. can be easily known by scanning. The tool will gather information about the database and check whether it is vulnerable to SQL injection. The tool will detect any kind of WordPress in the web application. This tool will check the WordPress username enumerator. The WordPress backup grabber vulnerability can be checked by the tool. If there exist any kind of sensitive files that can be also detected by using the tool. web applications may have subdomains. The tool will check the subdomain for a particular web application. Ports will be analyzed so that the vulnerable port address can be detected. The tool will also check for cross-site scripting vulnerabilities. If the cross-site scripting vulnerability is found, then the tool will show the corresponding results.

So, using the 'D-tect' tool will help us to find some most dangerous and harmful web application vulnerabilities. Information is a very serious concern in the present situation. If any kind of vulnerability is found, the host of the web application can correct it and establish a better security structure for them and the user of the web application. It will make the web application better and the information will be kept safe.

The tool 'D-tect' can detect the eight most web application vulnerabilities. It is good for the security implementation of any kind of web application. The tool is developed in python language. Initially, the tool analyzes only eight vulnerabilities. Some features can be added to this tool for further development. The tool can be integrated into a website so that users can easily check their web application security issues. This tool is a developer-friendly tool. Because many developers develop a different kind of web application. Some use WordPress for designing the framework. Ajax, DotNet, JavaScript, etc. are used for designing and logical implementation. So, loopholes can remain beneath the coding. If a developer can get the tool and check their developed web application. Many dictionary variables are written in the code. These dictionary variables help us to define parameters that we want to check. This also helps to implement the module we need for a particular operation on the instant. More dictionaries can be added for port scanning. Other ports can be scanned. More web application vulnerability testing such as security misconfiguration, insecure deserialization, using components with known variables, etc. The tool can be integrated into the mobile version by using APIs. The 'D-tect' tool is designed in python version 2.7. the tool can be upgraded in python 3.7 version in the future. The prevention of different web application vulnerabilities can be added to the tool in the future. This will help to make a secure platform for web application developers and users. It will prevent information theft and reduce the probability of using information for wrong purposes.

Bibliography:

[1] S. Webapp, “Web app development: the six different types of web apps.” [Online]. Available: https://en.yeeply.com/blog/6-different-kinds-web-app-development/. [Accessed: 22-Dec-2019].

[2] R. P. Adhyaru, “Techniques for Attacking Web Application Security,” *Int. J. Inf. Sci. Tech.*, vol. 6, no. 1/2, pp. 45–52, 2016.

[3] M. Baykara, “Investigation and Comparison of Web Application Vulnerabilities Test Tools,” vol. 7, no. 12, pp. 197–212, 2018.

[4] C. Watson *et al.*, “Advances in Electrical and Computer Engineering,” *Adv. Electr. Comput. Eng.*, vol. العدد الحا, no. 1, pp. 93–102, 2018.

[5] Z. Durić, “WAPTT - web application penetration testing tool,” *Adv. Electr. Comput. Eng.*, vol. 14, no. 1, pp. 93–102, 2014.

[6] A. Hasan and D. Meva, “Special Issue based on proceedings of 4TH International Conference on Cyber Security (ICCS) 2018 Web Application Safety by Penetration Testing,” pp. 159–163, 2018.

[7] Wordpress, “What is WordPress? | WordPress 101 Tutorials.” [Online]. Available: https://ithemes.com/tutorials/what-is-wordpress/. [Accessed: 22-Dec-2019].

[8] Subdomain, “What are Subdomains? (Definition and Examples).” [Online]. Available: https://www.wpbeginner.com/glossary/subdomain/. [Accessed: 22-Dec-2019].

[9] W. Vulnerability, “Online WordPress Security Scan for Vulnerabilities | WP Sec.” [Online]. Available: https://wpsec.com/. [Accessed: 22-Dec-2019].

[10] Harpreet Passi, “OWASP - Top 10 Vulnerabilities in web applications (updated for 2018).” [Online]. Available: https://www.greycampus.com/blog/information-security/owasp-top-vulnerabilities-in-web-applications. [Accessed: 23-Dec-2019].

[11] 15 vulnerabile Site, “15 Vulnerable Sites To (Legally) Practice Your Hacking Skills.” [Online]. Available: https://dst.com.ng/15-vulnerable-sites-legally-practice-hacking-skills/.

[Accessed: 25-Dec-2019].

[12] Cure 53, “GitHub - cure53/HTTPLeaks: HTTPLeaks - All possible ways, a website can leak HTTP requests.” [Online]. Available: https://github.com/cure53/HTTPLeaks. [Accessed: 25-Dec-2019].

[13] Game of Hacks | Checkmarx, “Game of Hacks | Checkmarx.” [Online]. Available: http://www.gameofhacks.com/. [Accessed: 25-Dec-2019].

[14] HollyGraceful, “Hacking Web Applications: — GracefulSecurity.” [Online]. Available: https://www.gracefulsecurity.com/hacking-web-applications/. [Accessed: 25-Dec-2019].

[15] Acunetix, “XSS Vulnerability Scanning | Acunetix.” [Online]. Available: https://www.acunetix.com/vulnerability-scanner/xss-vulnerability-scanning/. [Accessed: 25-Dec-2019].

[16] Acunetix SQL injection, “What is SQL Injection (SQLi) and How to Prevent It.” [Online]. Available: https://www.acunetix.com/websitesecurity/sql-injection/. [Accessed: 25-Dec-2019].

[17] Positive Technologies, “Web Application Vulnerabilities: Statistics for 2018.” [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/web-application-vulnerabilities-statistics-2019/. [Accessed: 25-Dec-2019].

[18] G. Erdogan, “Security Testing of Web Based Applications,” no. July, 2009.

[19] F. van der Loo, “Comparison of penetration testing tools for web applications,” p. 50, 2011.

[20] C. George, “Web Application Security Web Application Security 101,” no. January, pp. 1–10, 2010.

[21] J. N. Goel and B. M. Mehtre, “Vulnerability Assessment & Penetration Testing as a Cyber Defence Technology,” *Procedia Comput. Sci.*, vol. 57, pp. 710–715, 2015.

[22] MindMajix, “Top 20 Python Frameworks For Web Development [Updated 2020].” [Online]. Available: https://mindmajix.com/top-20-python-frameworks-list. [Accessed: 28-Dec-2019].

[23] pypi, “dash · PyPI.” [Online]. Available: https://pypi.org/project/dash/. [Accessed: 28-Dec-

2019].

[24] TutorialsTeacher.com, “Python Data Types.” [Online]. Available: https://www.tutorialsteacher.com/python/python-data-types. [Accessed: 29-Dec-2019].

[25] Cloudfare Inc, “Cloudflare - The Web Performance & Security Company | Cloudflare.” [Online]. Available: https://www.cloudflare.com/. [Accessed: 29-Dec-2019].

[26] Odino, “Secure your web application with these HTTP headers.” [Online]. Available: https://odino.org/secure-your-web-application-with-these-http-headers/. [Accessed: 29-Dec-2019].

[27] Python org, “20.16. urlparse — Parse URLs into components — Python 2.7.17 documentation.” [Online]. Available: https://docs.python.org/2/library/urlparse.html. [Accessed: 29-Dec-2019].

[28] Brute XSS, “XSS 101 - Brute XSS.” [Online]. Available: https://brutelogic.com.br/blog/xss101/. [Accessed: 29-Dec-2019].

Appendix:

```
import re,urllib,os,socket,sys

from pip._vendor.distlib.compat import raw_input

from moduleBS import BeautifulSoup
from urlparse import urlparse
from dtectcolors import Style,Fore,Back,init
init()
if os.name == 'nt':
   os.system('cls')
else:
   os.system('clear')

d4rk = 0
dr1 = "D4rk "


# -- Colors Start --
boldwhite     =     Style.BRIGHT+Fore.WHITE
boldgrey      =     Style.DIM+Fore.WHITE
reset        =     Style.RESET_ALL
green        =     Style.BRIGHT+Fore.GREEN
lightgreen    =     Fore.GREEN
red          =     Fore.RED
boldred         =     Style.BRIGHT+Fore.RED
# -- Colors End --


# -- Switches Start --
wpenumerator   =     "off"
filedetector   =         "off"
headercheck       =     "on"
subdomainscan  =     "off"
portscan      =     "off"
wpscan           =     "off"
xssscanner    =     "off"
wpbackupscan   =     "off"
sqliscanner       =     "off"
# -- Swiches End --


def dtect():
   print("  ____   _____ _____ ____ _____ ")
   print(" |  _ \ |_   _| ____/ ___|_   _|")
   print(" | | | | |__| | | _|| |      | |  ")
   print(" | |_| | |__| | | |__| |___   | |  ")
   print(" |____/   |_| |_____\____| |_|  v1.0")
   print("")
   print(" D-TECT - Pentest the Modern Web")
   print(" Author: Ashikin Talha - ( https://github.com/Ashikwome )")
   print(" Author: Ayan Chowdhury")
   print("")
   def menu():
      global
filedetector,wpenumerator,subdomainscan,portscan,wpscan,xssscanner,wpbackupscan,sqlis
canner
      print(" -- "+boldwhite+"Menu"+reset+" -- \n \n  1.     "+boldwhite+"WordPress
Username Enumerator"+reset+"   \n  2.   "+boldwhite+"Sensitive File Detector"+reset+"
\n  3.    "+boldwhite+"Sub-Domain Scanner"+reset+"\n  4.     "+boldwhite+"Port
Scanner"+reset+"        \n  5.   "+boldwhite+"Wordpress Scanner\n"+reset+"  6.
"+boldwhite+"Cross-Site Scripting [ XSS ] Scanner\n"+reset+"  7.
"+boldwhite+"Wordpress Backup Grabber\n"+reset+"  8.    "+boldwhite+"SQL Injection [
SQLI ] Scanner\n"+reset)
      option = raw_input("[+] Select Option\n    > ")
      if option == "1":
         wpenumerator = "on"
      elif option == "2":
         filedetector = "on"
      elif option == "3":
         subdomainscan = "on"
      elif option == "4":
         portscan = "on"
      elif option == "5":
         wpscan = "on"
      elif option == "6":
         xssscanner = "on"
      elif option == "7":
         wpbackupscan = "on"
      elif option == "8":
         sqliscanner = "on"
      else:
         print("[+] Incorrect Option selected")
         menu()

   def sock(i,secretswitch=0):
      secret = secretswitch
      global data,page,sourcecode
      if redirect == 1:
         data = host+i
      else:
         data = host.strip("/")+'/'+i
      page = urllib.urlopen(data)
      sourcecode = page.read()
      if secret == "1":
         return sourcecode
   def cloudflare():
      data = host #+'/'
      page = urllib.urlopen(data)
      pagesource = page.read()
      if "used CloudFlare to restrict access</title>" in pagesource:
         print("[!] Cloudflare blocked the IP")
         again()
   def alive():
      try:
         global page,splithost,ip
         data = host#+'/'
         page = urllib.urlopen(data)
         source = page.read()
         splithost = str(data.split("://")[1].split("/")[0])
         ip = socket.gethostbyname(splithost)
         print("[i] "+green+"Site is up!"+reset)
         print("  \n[+] Target Info:\n | URL: "+boldwhite+"%s"+reset+"\n | IP: 
"+boldwhite+"%s"+reset+"\n  ")%(data,ip)
         print("[+] Checking if any Cloudflare is blocking access...")
         cloudflare()
         redirectcheck()
      except(IOError):
         print("[!] "+red+"Error connecting to site! Site maybe down."+reset)
         again()
   def responseheadercheck():
      print('')
      headers       = ['set-cookie','x-cache','Location','Date','Content-
Type','Content-Length','Connection','Etag','Expires','Last-
Modified','Pragma','Vary','Cache-Control','X-Pingback','Accept-Ranges']
      headersfound   = []
      interesting    = []
      caution       = []
      cj = 0
      for i in page.headers:
         if i.lower() in str(headers).lower():
            pass
         elif i == "server":
            structure = str(i)+" : "+str(page.headers[i])
            headersfound.append(structure)
            structure = "Server : "+boldwhite+str(page.headers[i])+reset
            interesting.append(structure)
         elif i == "x-powered-by":
            structure = str(i)+" : "+str(page.headers[i])
            headersfound.append(structure)
            structure = "Powered by: "+boldwhite+str(page.headers[i])+reset
            interesting.append(structure)
         elif i == "x-frame-options":
            cj = 1
            pass
         else:
            structure = str(i)+" : "+str(page.headers[i])
            headersfound.append(structure)
      if cj == 0:
         caution.append("[!]"+red+" X-Frame-Options header Missing\n"+reset+"[!] 
"+red+"Page might be vulnerable to "+boldred+"Click Jacking\n"+reset+"[!] 
"+boldred+page.geturl()+reset+"\n[i] About ClickJacking: [ 
"+green+"https://www.owasp.org/index.php/Clickjacking"+reset+" ]")
      print("[+] Interesting Headers Found:")
      for i in headersfound:
         print(" | %s")%(i)
      if len(interesting) != 0:
         print("\n[i] Information from Headers:")
         for i in interesting:
            print(" | %s")%i
      print('')
      if cj == 0:
         print(caution[0])
      print('')
   def parameterarrange(payload):
      parsedurl = urlparse.urlparse(host)
      parameters = urlparse.parse_qsl(parsedurl.query, keep_blank_values=True)
      parameternames = []
      parametervalues = []

      for m in parameters:
         parameternames.append(m[0])
         parametervalues.append(m[1])


      for n in parameters:
         try:
            print("Checking '%s' parameter")%n[0]
            index = parameternames.index(n[0])
            original = parametervalues[index]
            parametervalues[index] = payload
            return urllib.urlencode(dict(zip(parameternames,parametervalues)))
            parametervalues[index] = original
         except(KeyError):
            pass
   def SQLIscan(site):
      print("[+] [ SQLI ] Scanner Started...\n")
      vuln = []
      payloads = {
               '2':'"',
               '1':'\''
      }
      errors = {
               'MySQL':'You have an error in your SQL syntax;',
               'Oracle':'SQL command not properly ended',
               'MSSQL':'Unclosed quotation mark after the character string',
               'PostgreSQL':'syntax error at or near'
      }
      path =
urlparse.urlparse(site).scheme+"://"+urlparse.urlparse(site).netloc+urlparse.urlparse
(site).path
      parsedurl = urlparse.urlparse(host)
      parameters = urlparse.parse_qsl(parsedurl.query, keep_blank_values=True)
      parameternames = []
      parametervalues = []

      for m in parameters:
         parameternames.append(m[0])
         parametervalues.append(m[1])


      for n in parameters:
         found = 0
         print("[+] Checking '%s' parameter")%n[0]
         try:
            for i in payloads:
               pay = payloads[i]
               index = parameternames.index(n[0])
               original = parametervalues[index]
               parametervalues[index] = pay
               modifiedurl =
urllib.urlencode(dict(zip(parameternames,parametervalues)))
               parametervalues[index] = original
               modifiedparams = modifiedurl
               payload = urllib.quote_plus(payloads[i])
               u = urllib.urlopen(path+"?"+modifiedparams)
               source = u.read()
               #print ("[+] Checking HTML Context...")

               for i in errors:
                  if errors[i] in source:#htmlcode[0].contents[0]:
                     dbfound = " | Back-End Database: "+green+str(i)+reset
                     found = 1
                     break
               if found != 1:
                  break
         except(KeyError):
            pass

         if found == 1:
            print("[!] "+red+"SQL Injection Vulnerability Found!"+reset)
            print (dbfound)
            vuln.append("'"+n[0]+"'")
            found = 0
      if len(vuln) != 0:
         print(" | Vulnerable Parameter/s:"),
         for i in vuln:
            print(i),

      else:
         print("[!] Not Vulnerable")
   def XSSscan(site):
      print("[+] [ XSS ] Scanner Started...")
      vuln = []
      payloads = {
               '3':'d4rk();"\'\\/}{d4rk',
               '2':'d4rk</script><script>alert(1)</script>d4rk',
               '1':'<d4rk>'
      }
      path =
urlparse.urlparse(site).scheme+"://"+urlparse.urlparse(site).netloc+urlparse.urlparse
(site).path
      parsedurl = urlparse.urlparse(host)
      parameters = urlparse.parse_qsl(parsedurl.query, keep_blank_values=True)
      parameternames = []
      parametervalues = []

      for m in parameters:
         parameternames.append(m[0])
         parametervalues.append(m[1])
    for n in parameters:
      found = 0
      print(" | Checking '%s' parameter")%n[0]
      try:
        for i in payloads:
          pay = payloads[i]
          index = parameternames.index(n[0])
          original = parametervalues[index]
          parametervalues[index] = pay
          modifiedurl =
urllib.urlencode(dict(zip(parameternames,parametervalues)))
          parametervalues[index] = original
          modifiedparams = modifiedurl
          payload = urllib.quote_plus(payloads[i])
          u = urllib.urlopen(path+"?"+modifiedparams)
          source = u.read()
          code = BeautifulSoup(source)
          if str(i) == str(1):
            #print ("[+] Checking HTML Context...")
            if payloads[i] in source:#htmlcode[0].contents[0]:
              #print("[+] XSS Vulnerability Found.")
              found = 1
          script = code.findAll('script')
          if str(i) == str(3) or str(i) == str(2):
            #print("[+] Checking JS Context...")
            if str(i) == str(3):
              #JS Context
              for p in range(len(script)):
                try:
                  if pay in script[p].contents[0]:
                    #print("[+] XSS Vulnerability Found")
                    found = 1
                except(IndexError):
                  pass
            if str(i) == str(2):
              if payloads['2'] in source:
                #  print("[+] XSS Vulnerability Found")
                found = 1
      except(KeyError):
        pass

      if found == 1:
        vuln.append("'"+n[0]+"'")
        found = 0
    if len(vuln) != 0:
      print("[!] "+red+"Vulnerable Parameter/s:"+reset),
      for i in vuln:
        print(boldred+i+reset),
    else:
      print("[!] Not Vulnerable")
  def portscanner():
    print("[i] Syntax  :  Function")
    print("23,80,120:Scans Specific Ports, e.g,ScansPort23 ,80and120 ")
    print("23-80:Scans a Range of Ports, e.g, Scans Port from 23 to 80")
print("23:Scans a single port, e.g, Scans Port 23")
print("all:Scans all ports from 20 to 5000")
print(" ")
portoption = raw_input("[+] Enter Range or Port:\n > ")
wasmultiple        =        0
wasrange       =        0
wasone        =    0
if ',' in portoption:
   wasmultiple = 1
   multipleport = portoption.split(',')
   notexpected = 0
   for i in multipleport:
      if not str(i).isdigit():
         print("[!] Incorrect Syntax!")
         notexpected = 1
   if notexpected == 1:
      again()
   totallength = multipleport
elif '-' in portoption:
   wasrange = 1
   rangeport = portoption.split('-')
   totalrange = range(int(rangeport[0]),int(rangeport[1])+1)
   if len(rangeport) != 2:
      print("[!] Incorrect Syntax!")
      again()
   totallength = totalrange
elif portoption == 'all':
   totallength = range(20,5000)
elif portoption.isdigit():
   wasone = 1
   oneport = int(portoption)
   totallength = range(1)
else:
   print("[+] Incorrect Syntax!")
   again()
print("[+] Scanning %s Port/s on Target: %s")%(len(totallength),ip)
ports = 5000
found = 1
protocolname = 'tcp'
progress = 20
loopcondition = range(20,5000)
if portoption == 'all':
   loopcondition = range(20,5000)
   ports = 5000
   progress = 20
elif wasmultiple == 1:
   loopcondition = multipleport
   ports = int(len(multipleport))
   progress = 0 #int(min(multipleport))
elif wasrange == 1:
   loopcondition = totalrange
   ports = int(rangeport[1])
   progress = int(rangeport[0])-1
elif wasone == 1:
   onlyport = []
```

```
         onlyport.append(portoption)
         loopcondition = onlyport
         progress = 0
         ports = 1
      else:
         loopcondition = range(20,5000)
      for i in loopcondition:
         i = int(i)
         progress += 1
         sys.stdout.write("\r[+] Progress %i / %s ..."% (progress,ports))
         sys.stdout.flush()
         portconnect = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
         response = portconnect.connect_ex((ip, i))
         if(response == 0) :
            print ('\n | Port: '+boldwhite+'%d'+reset+' \n | Status: 
'+green+'OPEN'+reset+'\n | Service: '+boldwhite+'%s'+reset+'\n')%
(i,socket.getservbyport(i, protocolname))
            found += 1
         portconnect.close()
      if found == 1:
         print("\n | "+red+"No Open Ports Found!"+reset)
   def subdomainscanner():

      import sys
      print("\n[+] Subdomain Scanner Start!")
      wordlist = 
["mail","localhost","blog","forum","0","01","02","03","1","10","11","12","13","14","1
5","16","17","18","19","2","20","3","3com","4","5","6","7","8","9","ILMI","a","a.auth
-
ns","a01","a02","a1","a2","abc","about","ac","academico","acceso","access","accountin
g","accounts","acid","activestat","ad","adam","adkit","admin","administracion","admin
istrador","administrator","administrators","admins","ads","adserver","adsl","ae","af"
,"affiliate","affiliates","afiliados","ag","agenda","agent","ai","aix","ajax","ak","a
kamai","al","alabama","alaska","albuquerque","alerts","alpha","alterwind","am","amari
llo","americas","an","anaheim","analyzer","announce","announcements","antivirus","ao"
,"ap","apache","apollo","app","app01","app1","apple","application","applications","ap
ps","appserver","aq","ar","archie","arcsight","argentina","arizona","arkansas","arlin
gton","as","as400","asia","asterix","at","athena","atlanta","atlas","att","au","aucti
on","austin","auth","auto","av","aw","ayuda","az","b","b.auth-
ns","b01","b02","b1","b2","b2b","b2c","ba","back","backend","backup","baker","bakersf
ield","balance","balancer","baltimore","banking","bayarea","bb","bbdd","bbs","bd","bd
c","be","bea","beta","bf","bg","bh","bi","billing","biz","biztalk","bj","black","blac
kberry","blogs","blue","bm","bn","bnc","bo","bob","bof","boise","bolsa","border","bos
ton","boulder","boy","br","bravo","brazil","britian","broadcast","broker","bronze","b
rown","bs","bsd","bsd0","bsd01","bsd02","bsd1","bsd2","bt","bug","buggalo","bugs","bu
gzilla","build","bulletins","burn","burner","buscador","buy","bv","bw","by","bz","c",
"c.auth-
ns","ca","cache","cafe","calendar","california","call","calvin","canada","canal","can
on","careers","catalog","cc","cd","cdburner","cdn","cert","certificates","certify","c
ertserv","certsrv","cf","cg","cgi","ch","channel","channels","charlie","charlotte","c
hat","chats","chatserver","check","checkpoint","chi","chicago","ci","cims","cincinnat
i","cisco","citrix","ck","cl","class","classes","classifieds","classroom","cleveland"
,"clicktrack","client","clientes","clients","club","clubs","cluster","clusters","cm",
"cmail","cms","cn","co","cocoa","code","coldfusion","colombus","colorado","columbus",
"com","commerce","commerceserver","communigate","community","compaq","compras","con",
"concentrator","conf","conference","conferencing","confidential","connect","connectic
ut","consola","console","consult","consultant","consultants","consulting","consumer",
"contact","content","contracts","core","core0","core01","corp","corpmail","corporate"
,"correo","correoweb","cortafuegos","counterstrike","courses","cr","cricket","crm","c
rs","cs","cso","css","ct","cu","cust1","cust10","cust100","cust101","cust102","cust10
3","cust104","cust105","cust106","cust107","cust108","cust109","cust11","cust110","cu
st111","cust112","cust113","cust114","cust115","cust116","cust117","cust118","cust119
","cust12","cust120","cust121","cust122","cust123","cust124","cust125","cust126","cus
t13","cust14","cust15","cust16","cust17","cust18","cust19","cust2","cust20","cust21",
"cust22","cust23","cust24","cust25","cust26","cust27","cust28","cust29","cust3","cust
30","cust31","cust32","cust33","cust34","cust35","cust36","cust37","cust38","cust39",
"cust4","cust40","cust41","cust42","cust43","cust44","cust45","cust46","cust47","cust
48","cust49","cust5","cust50","cust51","cust52","cust53","cust54","cust55","cust56","
cust57","cust58","cust59","cust6","cust60","cust61","cust62","cust63","cust64","cust6
5","cust66","cust67","cust68","cust69","cust7","cust70","cust71","cust72","cust73","c
ust74","cust75","cust76","cust77","cust78","cust79","cust8","cust80","cust81","cust82
","cust83","cust84","cust85","cust86","cust87","cust88","cust89","cust9","cust90","cu
st91","cust92","cust93","cust94","cust95","cust96","cust97","cust98","cust99","custom
er","customers","cv","cvs","cx","cy","cz","d","dallas","data","database","database01"
,"database02","database1","database2","databases","datastore","datos","david","db","d
b0","db01","db02","db1","db2","dc","de","dealers","dec","def","default","defiant","de
laware","dell","delta","delta1","demo","demonstration","demos","denver","depot","des"
,"desarrollo","descargas","design","designer","detroit","dev","dev0","dev01","dev1","
devel","develop","developer","developers","development","device","devserver","devsql"
,"dhcp","dial","dialup","digital","dilbert","dir","direct","directory","disc","discov
ery","discuss","discussion","discussions","disk","disney","distributer","distributers
","dj","dk","dm","dmail","dmz","dnews","dns","dns-
2","dns0","dns1","dns2","dns3","do","docs","documentacion","documentos","domain","dom
ains","dominio","domino","dominoweb","doom","download","downloads","downtown","dragon
","drupal","dsl","dyn","dynamic","dynip","dz","e","e-com","e-
commerce","e0","eagle","earth","east","ec","echo","ecom","ecommerce","edi","edu","edu
cation","edward","ee","eg","eh","ejemplo","elpaso","email","employees","empresa","emp
resas","en","enable","eng","eng01","eng1","engine","engineer","engineering","enterpri
se","epsilon","er","erp","es","esd","esm","espanol","estadisticas","esx","et","eta","
europe","events","domain","exchange","exec","extern","external","extranet","f","f5","
falcon","farm","faststats","fax","feedback","feeds","fi","field","file","files","file
serv","fileserver","filestore","filter","find","finger","firewall","fix","fixes","fj"
,"fk","fl","flash","florida","flow","fm","fo","foobar","formacion","foro","foros","fo
rtworth","forums","foto","fotos","foundry","fox","foxtrot","fr","france","frank","fre
d","freebsd","freebsd0","freebsd01","freebsd02","freebsd1","freebsd2","freeware","fre
sno","front","frontdesk","fs","fsp","ftp","ftp-
","ftp0","ftp2","ftp_","ftpserver","fw","fw-
1","fw1","fwsm","fwsm0","fwsm01","fwsm1","g","ga","galeria","galerias","galleries","g
allery","games","gamma","gandalf","gate","gatekeeper","gateway","gauss","gd","ge","ge
mini","general","george","georgia","germany","gf","gg","gh","gi","gl","glendale","gm"
,"gmail","gn","go","gold","goldmine","golf","gopher","gp","gq","gr","green","group","
groups","groupwise","gs","gsx","gt","gu","guest","gw","gw1","gy","h","hal","halflife"
,"hawaii","hello","help","helpdesk","helponline","henry","hermes","hi","hidden","hk",
"hm","hn","hobbes","hollywood","home","homebase","homer","honeypot","honolulu","host"
,"host1","host3","host4","host5","hotel","hotjobs","houstin","houston","howto","hp","
hpov","hr","ht","http","https","hu","hub","humanresources","i","ia","ias","ibm","ibmd
b","id","ida","idaho","ids","ie","iis","il","illinois","im","images","imail","imap","
imap4","img","img0","img01","img02","in","inbound","inc","include","incoming","india"
,"indiana","indianapolis","info","informix","inside","install","int","intern","intern
al","international","internet","intl","intranet","invalid","investor","investors","in
```

```
via","invio","io","iota","iowa","iplanet","ipmonitor","ipsec","ipsec-
gw","iq","ir","irc","ircd","ircserver","ireland","iris","irvine","irving","is","isa",
"isaserv","isaserver","ism","israel","isync","it","italy","ix","j","japan","java","je
","jedi","jm","jo","jobs","john","jp","jrun","juegos","juliet","juliette","juniper","
k","kansas","kansascity","kappa","kb","ke","kentucky","kerberos","keynote","kg","kh",
"ki","kilo","king","km","kn","knowledgebase","knoxville","koe","korea","kp","kr","ks"
,"kw","ky","kz","l","la","lab","laboratory","labs","lambda","lan","laptop","laserjet"
,"lasvegas","launch","lb","lc","ldap","legal","leo","li","lib","library","lima","linc
oln","link","linux","linux0","linux01","linux02","linux1","linux2","lista","lists","l
istserv","listserver","live","lk","load","loadbalancer","local","log","log0","log01",
"log02","log1","log2","logfile","logfiles","logger","logging","loghost","login","logs
","london","longbeach","losangeles","lotus","louisiana","lr","ls","lt","lu","luke","l
v","ly","lyris","m","ma","mac","mac1","mac10","mac11","mac2","mac3","mac4","mac5","ma
ch","macintosh","madrid","mail2","mailer","mailgate","mailhost","mailing","maillist",
"maillists","mailroom","mailserv","mailsite","mailsrv","main","maine","maint","mall",
"manage","management","manager","manufacturing","map","mapas","maps","marketing","mar
ketplace","mars","marvin","mary","maryland","massachusetts","master","max","mc","mci"
,"md","mdaemon","me","media","member","members","memphis","mercury","merlin","message
s","messenger","mg","mgmt","mh","mi","miami","michigan","mickey","midwest","mike","mi
lwaukee","minneapolis","minnesota","mirror","mis","mississippi","missouri","mk","ml",
"mm","mn","mngt","mo","mobile","mom","monitor","monitoring","montana","moon","moscow"
,"movies","mozart","mp","mp3","mpeg","mpg","mq","mr","mrtg","ms","ms-exchange","ms-
sql","msexchange","mssql","mssql0","mssql01","mssql1","mt","mta","mtu","mu","multimed
ia","music","mv","mw","mx","my","mysql","mysql0","mysql01","mysql1","mz","n","na","na
me","names","nameserv","nameserver","nas","nashville","nat","nc","nd","nds","ne","neb
raska","neptune","net","netapp","netdata","netgear","netmeeting","netscaler","netscre
en","netstats","network","nevada","new","newhampshire","newjersey","newmexico","newor
leans","news","newsfeed","newsfeeds","newsgroups","newton","newyork","newzealand","nf
","ng","nh","ni","nigeria","nj","nl","nm","nms","nntp","no","node","nokia","nombres",
"nora","north","northcarolina","northdakota","northeast","northwest","noticias","nove
ll","november","np","nr","ns","ns-
","ns0","ns01","ns02","ns1","ns2","ns3","ns4","ns5","ns_","nt","nt4","nt40","ntmail",
"ntp","ntserver","nu","null","nv","ny","nz","o","oakland","ocean","odin","office","of
fices","oh","ohio","ok","oklahoma","oklahomacity","old","om","omaha","omega","omicron
","online","ontario","open","openbsd","openview","operations","ops","ops0","ops01","o
ps02","ops1","ops2","opsware","or","oracle","orange","order","orders","oregon","orion
","orlando","oscar","out","outbound","outgoing","outlook","outside","ov","owa","owa01
","owa02","owa1","owa2","ows","oxnard","p","pa","page","pager","pages","paginas","pap
a","paris","parners","partner","partners","patch","patches","paul","payroll","pbx","p
c","pc01","pc1","pc10","pc101","pc11","pc12","pc13","pc14","pc15","pc16","pc17","pc18
","pc19","pc2","pc20","pc21","pc22","pc23","pc24","pc25","pc26","pc27","pc28","pc29",
"pc3","pc30","pc31","pc32","pc33","pc34","pc35","pc36","pc37","pc38","pc39","pc4","pc
40","pc41","pc42","pc43","pc44","pc45","pc46","pc47","pc48","pc49","pc5","pc50","pc51
","pc52","pc53","pc54","pc55","pc56","pc57","pc58","pc59","pc6","pc60","pc7","pc8","p
c9","pcmail","pda","pdc","pe","pegasus","pennsylvania","peoplesoft","personal","pf","
pg","pgp","ph","phi","philadelphia","phoenix","phoeniz","phone","phones","photos","pi
","pics","pictures","pink","pipex-
gw","pittsburgh","pix","pk","pki","pl","plano","platinum","pluto","pm","pm1","pn","po
","policy","polls","pop","pop3","portal","portals","portfolio","portland","post","pos
ta","posta01","posta02","posta03","postales","postoffice","ppp1","ppp10","ppp11","ppp
12","ppp13","ppp14","ppp15","ppp16","ppp17","ppp18","ppp19","ppp2","ppp20","ppp21","p
pp3","ppp4","ppp5","ppp6","ppp7","ppp8","ppp9","pptp","pr","prensa","press","print >>
sys.stdout,er","print >> sys.stdout,serv","print >>
sys.stdout,server","priv","privacy","private","problemtracker","products","profiles",
"project","projects","promo","proxy","prueba","pruebas","ps","psi","pss","pt","pub","
```

```
public","pubs","purple","pw","py","q","qa","qmail","qotd","quake","quebec","queen","q
uotes","r","r01","r02","r1","r2","ra","radio","radius","rapidsite","raptor","ras","rc
","rcs","rd","re","read","realserver","recruiting","red","redhat","ref","reference","
reg","register","registro","registry","regs","relay","rem","remote","remstats","repor
ts","research","reseller","reserved","resumenes","rho","rhodeisland","ri","ris","rmi"
,"ro","robert","romeo","root","rose","route","router","router1","rs","rss","rtelnet",
"rtr","rtr01","rtr1","ru","rune","rw","rwhois","s","s1","s2","sa","sac","sacramento",
"sadmin","safe","sales","saltlake","sam","san","sanantonio","sandiego","sanfrancisco"
,"sanjose","saskatchewan","saturn","sb","sbs","sc","scanner","schedules","scotland","
scotty","sd","se","search","seattle","sec","secret","secure","secured","securid","sec
urity","sendmail","seri","serv","serv2","server","server1","servers","service","servi
ces","servicio","servidor","setup","sg","sh","shared","sharepoint","shareware","shipp
ing","shop","shoppers","shopping","si","siebel","sierra","sigma","signin","signup","s
ilver","sim","sirius","site","sj","sk","skywalker","sl","slackware","slmail","sm","sm
c","sms","smtp","smtphost","sn","sniffer","snmp","snmpd","snoopy","snort","so","socal
","software","sol","solaris","solutions","soporte","source","sourcecode","sourcesafe"
,"south","southcarolina","southdakota","southeast","southwest","spain","spam","spider
","spiderman","splunk","spock","spokane","springfield","sprint >>
sys.stdout,","sqa","sql","sql0","sql01","sql1","sql7","sqlserver","squid","sr","ss","
ssh","ssl","ssl0","ssl01","ssl1","st","staff","stage","staging","start","stat","stati
c","statistics","stats","stlouis","stock","storage","store","storefront","streaming",
"stronghold","strongmail","studio","submit","subversion","sun","sun0","sun01","sun02"
,"sun1","sun2","superman","supplier","suppliers","support","sv","sw","sw0","sw01","sw
1","sweden","switch","switzerland","sy","sybase","sydney","sysadmin","sysback","syslo
g","syslogs","system","sz","t","tacoma","taiwan","talk","tampa","tango","tau","tc","t
cl","td","team","tech","technology","techsupport","telephone","telephony","telnet","t
emp","tennessee","terminal","terminalserver","termserv","test","test2k","testbed","te
sting","testlab","testlinux","testo","testserver","testsite","testsql","testxp","texa
s","tf","tftp","tg","th","thailand","theta","thor","tienda","tiger","time","titan","t
ivoli","tj","tk","tm","tn","to","tokyo","toledo","tom","tool","tools","toplayer","tor
onto","tour","tp","tr","tracker","train","training","transfers","trinidad","trinity",
"ts","ts1","tt","tucson","tulsa","tumb","tumblr","tunnel","tv","tw","tx","tz","u","ua
","uddi","ug","uk","um","uniform","union","unitedkingdom","unitedstates","unix","unix
ware","update","updates","upload","ups","upsilon","uranus","urchin","us","usa","usene
t","user","users","ut","utah","utilities","uy","uz","v","va","vader","vantive","vault
","vc","ve","vega","vegas","vend","vendors","venus","vermont","vg","vi","victor","vid
eo","videos","viking","violet","vip","virginia","vista","vm","vmserver","vmware","vn"
,"vnc","voice","voicemail","voip","voyager","vpn","vpn0","vpn01","vpn02","vpn1","vpn2
","vt","vu","w","w1","w2","w3","wa","wais","wallet","wam","wan","wap","warehouse","wa
shington","wc3","web","webaccess","webadmin","webalizer","webboard","webcache","webca
m","webcast","webdev","webdocs","webfarm","webhelp","weblib","weblogic","webmail","we
bmaster","webproxy","webring","webs","webserv","webserver","webservices","website","w
ebsites","websphere","websrv","websrvr","webstats","webstore","websvr","webtrends","w
elcome","west","westvirginia","wf","whiskey","white","whois","wi","wichita","wiki","w
ililiam","win","win01","win02","win1","win2","win2000","win2003","win2k","win2k3","wi
ndows","windows01","windows02","windows1","windows2","windows2000","windows2003","win
dowsxp","wingate","winnt","winproxy","wins","winserve","winxp","wire","wireless","wis
consin","wlan","wordpress","work","world","write","ws","ws1","ws10","ws11","ws12","ws
13","ws2","ws3","ws4","ws5","ws6","ws7","ws8","ws9","wusage","wv","ww","www","www-
","www-01","www-02","www-1","www-2","www-
int","www0","www01","www02","www1","www2","www3","www_","wwwchat","wwwdev","wwwmail",
"wy","wyoming","x","x-
ray","xi","xlogan","xmail","xml","xp","y","yankee","ye","yellow","young","yt","yu","z
","z-log","za","zebra","zera","zeus","zlog","zm","zulu","zw"]
      progress = 0
      for i in wordlist:
         progress += 1
         sys.stdout.write("\r[+] Progress %i / %s ..."% (progress,len(wordlist)))
         sys.stdout.flush()
         try:
            s = socket.gethostbyname(i+'.'+splithost)
            if (s):
               so = socket.gethostbyname_ex(i+'.'+splithost)
               print("\n[+] Subdomain found!\n | Subdomain: %s.%s \n | Nameserver:
%s\n | IP: %s")%(i,splithost,so[0],s)
               if s == '127.0.0.1':
                  print("[!] "+red+"Sub-domain is vulnerable to "+boldred+"Same-Site
Scripting! "+reset+"\n[!] About Same-Site Scripting:\n[!] [
"+green+"https://www.acunetix.com/vulnerabilities/web/same-site-scripting"+reset+" ]
")
               print('')
         except(socket.gaierror):
            pass
   def enumform(listofIDs,listofnames):
      lengthofnames =  len(max(listofnames, key=len))
      lengthofIDs = len(max(listofIDs, key=len))
      if lengthofnames < 12:
         lengthofnames = 12
      print ("[i] "+green+"Found the following Username/s:"+reset)
      print ("\t+-"+'-'.center(6, '-')+'-+-'+'-'.center(lengthofnames, '-')+"-+" )
      print ("\t| "+'ID/s'.center(6, ' ')+' | '+'Username/s'.center(lengthofnames, '
')+" |")
      print ("\t+-"+'-'.center(6, '-')+'-+-'+'-'.center(lengthofnames, '-')+"-+")
      for i,d in zip(listofnames,listofIDs):
         print ('\t| '+d.center(6, ' ')+" | "+i.center(lengthofnames, ' ')+' |')
      print ("\t+-"+'-'.center(6, '-')+'-+-'+'-'.center(lengthofnames, '-')+"-+")
      print("")
   def wpbackupscanner():
      backups = ['wp-config.php~','wp-config.php.txt','wp-config.php.save','.wp-
config.php.swp','wp-config.php.swp','wp-config.php.swo','wp-config.php_bak','wp-
config.bak','wp-config.php.bak','wp-config.save','wp-config.old','wp-
config.php.old','wp-config.php.orig','wp-config.orig','wp-config.php.original','wp-
config.original','wp-config.txt']
      print("[+] Scan Started")
      print("[+] Searching Wordpress Backups...")
      print("[?] Note: Press CTRL+C to skip\n  ")
      progress = 0
      backup = []
      backupurl = []
      try:
         for i in backups:
            progress += 1
            sys.stdout.write("\r[+] Progress %i / %s ..."% (progress,len(backups)))
            sys.stdout.flush()
            sock(i)
            if page.getcode() == 200:
               detecting = sock(i,"1")
               if "define('DB_PASSWORD'" in detecting:
                  s1 = i
                  s2 = data
```

```
                backup.append(s1)
                backupurl.append(s2)
      except(KeyboardInterrupt):
         print("\n[+] File detection skipped")
      print('')
      for ifile,iurl in zip(backup,backupurl):
         print("[!] "+boldred+"Backup Found!\n"+reset+" | "+red+"Filename: 
"+boldred+"%s"+reset+"\n | "+red+"URL: "+boldred+"%s\n"+reset)%(ifile,iurl)
   def wpenumeration():
      import time
      global d4rk,dr1,host
      page = urllib.urlopen(host)
      url = page.geturl()
      if page.geturl() != host:
         print("[i] The remote host redirects to '"+str(url)+"' \n    Following the 
redirection...")
         host = page.geturl()
      print("\n[+] Scan Started : "+lightgreen+"%s"+reset) % time.strftime("%c")
      print ("[+] Enumeration Usernames...")
      T = 33
      found = 0
      listofusernames = []
      listofids = []
      for i in range(30):
         authorlink = host+"?author="+str(i+1)
         url = urllib.urlopen(authorlink)
         source = url.read()
         if url.geturl() == authorlink:
            break
         else:
            com = str(host)+"/author/"
            res = url.geturl()
            res = res.split("/")
            while len(res) >=3:
               res.pop(0)
            listofusernames.append(res[0])
            listofids.append(str(i+1))
            found = 1
      d4rk = dr1+str(1)+str(T)+str(7)
      if found == 0:
         print("[+] "+red+"No Usernames detected"+reset)
      else:
         enumform(listofids,listofusernames)
      print("[+] Enumeration Completed.")
      print("[+] Scan Ended : "+lightgreen+"%s"+reset) % time.strftime("%c")
   def wpscanner():
      print("  \n[+] Detecting Wordpress")
      wp = 0
      i = 'wp-admin/'
      sock(i) sock is afunc wich is declared in
      if "wp-login.php?redirect_to" in page.geturl():
         wp = 1
         print(green+"[i] "+green+"Wordpress Detected!"+reset)
         if wpenumeration == "on":
            wpenumeration()
         else:

            wpenumeration()
      if wp == 0:
         i = 'wp-content/index.php'
         sock(i)
         if page.getcode() == 200 and "" in page.read():
            print("[!] "+green+"Wordpress Detected!"+reset)
            wp = 1
            if wpenumeration == "on":
               wpenumeration()
            else:
               wpbackupscanner()
               wpenumeration()
      if wp == 0:
         print("[!] "+red+"No Wordpress Detected"+reset)
   def redirectcheck():
      global redirect,host
      redirect = 0
      print("[+] Checking Redirection")
      page = urllib.urlopen(host)
      url = page.geturl()
      if page.geturl() != host:
         option = raw_input("[i] "+boldgrey+"Host redirects to "+str(url)+reset+" \n
Set this as default Host? [Y/N]:\n    > ")
         if option.lower() == "y":
            host = page.geturl()
            redirect = 1
      else:
         print("[+] URL isn't redirecting")
   def again():
      global
wpenumerator,filedetector,subdomainscan,portscan,wpscan,xssscanner,wpbackupscan,sqlis
canner
      # -- Switches Reset --
      wpenumerator   =     "off"
      filedetector   =         "off"
      subdomainscan  =     "off"
      portscan      =     "off"
      wpscan           =     "off"
      xssscanner    =     "off"
      wpbackupscan   =     "off"
      sqliscanner       =     "off"
      # -- Swiches Reset --
      inp = raw_input("\n[+] [E]xit or launch [A]gain? (e/a)").lower()
      if inp == 'a':
         dtect()
      elif inp == 'e':
         exit()
      else:
         print("[!] Incorrect option selected")
         again()

# -- Program Structure Start --
   menu()
```

```
    try:
        global host
        host = raw_input("[+] Enter Domain \n    e.g, site.com\n    > ")
        if 'https://' in host:
            pass
        elif 'http://' in host:
            pass
        else:
            host = "http://"+host
        print("[+] Checking Status...")
        alive()
        responseheadercheck()
        if xssscanner == "on":
            XSSscan(host)
        if sqliscanner == "on":
            SQLIscan(host)
        if wpbackupscan == "on":
            wpbackupscanner()
        if filedetector == "on":
            files = 
['robots.txt','crossdomain.xml','.htaccess','clientaccesspolicy.xml','infophp.php','l
og.txt','logs.txt','CHANGELOG.txt','awstats/data/']
            print("[+] Scan Started")
            print("[+] Searching sensitive files...")
            print("[?] Note: Press CTRL+C to skip\n  ")
            try:
                for i in files:
                    if i == "awstats/data/":
                        sock(i)
                        if "<title>Index of /awstats/data</title>" in sourcecode:
                            print("[!] awstats detected!\n[!] URL: %s")%(data)
                    else:
                        sock(i)
                        if page.getcode() == 200:
                            print("[!] File Found!\n | Name: %s\n | URL: %s\n")%(i,data)
            except(KeyboardInterrupt):
                print("\n[+] File detection skipped")
        if wpenumerator == "on":
            print("  \n[+] Detecting Wordpress")
            wp = 0
            i = 'wp-admin/'
            sock(i)
            if "wp-login.php?redirect_to" in page.geturl():
                wp = 1
                print(green+"[i] "+green+"Wordpress Detected!"+reset)
                wpenumeration()
            if wp == 0:
                i = 'wp-content/index.php'
                sock(i)
                if page.getcode() == 200 and "" in page.read():
                    print("[!] "+green+"Wordpress Detected!"+reset)
                    wp = 1
                    wpenumeration()
            if wp == 0:
                print("[!] "+red+"No Wordpress Detected"+reset)
```

```
      if wpscan == "on":
         wpscanner()
      if subdomainscan == "on":
         subdomainscanner()
      if portscan == "on":
         portscanner()
      again()
   except(KeyboardInterrupt) as Exit:
      print("\n[+] Exiting...")
      sys.exit()
# -- Program Structure End --
dtect()
```