CHALMERS
UNIVERSITY OF TECHNOLOGY

Visualizing HTTP traffic flows from packet data

Master's Thesis in Computer Science - Computer systems & Networks

SHAILAJA MALLICK

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 8, 2015

Visualizing HTTP traffic flows from packet data

SHAILAJA MALLICK

Examiner: MAGNUS ALMGREN
Supervisor: VINCENZO MASSIMILIANO GULISANO
Supervsior at Fox-IT: BARRY WEYMES

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone +46(0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, June 8, 2015

Abstract

The Hyper Text Transfer Protocol (HTTP) implemented in browsers to find information on Internet is a large part of the traffic caused by the browsers which are the predominant source of data communication in networks. The content of a website varies from static displays of a simple page to rich media applications along with a large number of third-party advertisements. While having the advantage of gathering as much information as possible for a user, websites also have the disadvantage of making it possible for an attacker to exploit its structure and design. Any susceptible website if manipulated by an attacker with the injection of malicious software i.e. malware will have the potential to compromise a user's machine along with sensitive information that he/she has. In order to investigate these security incidents, one needs to have a thorough analysis of HTTP network traffic streams as it provides a bigger picture of the request/ response mechanism along with the information of embedded requests if any.

However, with the complex structure of web pages, it is a huge manual effort to scan through hundreds of megabytes of HTTP streams extracted from packet capture tools. Differentiating from benign traffic to malicious traffic in such large size files makes it a slow and lengthy process. However, if this process can be automated by a tool which can extract HTTP streams from packet capture files, process and filter it and finally, provide a visual representation of the HTTP traffic flow over time efficiently, it can be beneficial for a forensic analyst to easily figure out the malicious traffic.

The main purpose of this thesis is to develop such a prototype which can effectively and efficiently visualize the flow(s) of HTTP traffic from websites with a partial focus on malicious advertisement- 'malvertising'. This paper describes the methodology used to extract the HTTP traffic from packet data and thereby using the data or metadata from the extracted information to visualize it in the form of a graph over time. The results show that it is even possible for large size files to clearly display the traffic flows efficiently with the ability to further analyze each node in the graph. It also shows that if malicious traffic is found, it can be traced back to its parent host and thereby, it is possible to understand the root cause.

Keywords: security, HTTP flows, malicious URLs, visualization.

Acknowledgements

This thesis was carried out in collaboration with the department of Computer Science & Engineering, Chalmers University of Technology and FOX-IT.

I would like to thank my examiner, Magnus Almgren and supervisor, Vincenzo Massimiliano Gulisano for their support, valuable advices, feedback that greatly improved the report and guidance through this master thesis.

I would also like to thank Barry Weymes, Yonathan Klijnsma and Gina Doekhie of FOX-IT for assistance with the techniques, methodology, providing with new ideas and feedback during biweekly meetings.

Finally, I would like to take the opportunity to thank the Swedish Institute for granting the SI scholarship award which enabled me to pursue the masters degree. I would also thank my family for all the support throughout the master studies.

Shailaja Mallick
Gothenburg, June 2015

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

With the rapid growth of technology, Internet and in particular, the web, has become an important part of our everyday computing experience. Numerous web applications are being developed and deployed that provide various useful services to the user such as electronic documents, entertainment, business-related applications and education. These services are gaining popularity due to their ease of use and fast development. The increased usage of these Internet services makes an end user to rely more and more on the web to flexibly and conveniently perform tasks that would in general require physical presence.

However, the popular and ubiquitous nature of the web attracts the attention of malicious entities. The web and its global user community have observed various forms of attack in the past [1][2]. The most prominent among these attacks is the use of the web as a means to inject malware i.e. malicious software. Malware includes different kinds of computer viruses, worms, Trojans, spyware and other unwanted software. Attackers are using the web as a platform to distribute malware by exploiting vulnerable web pages and browsers. These web-based malware mainly propagate when a user visits such vulnerable websites and the attacker injects malicious scripts into the web pages via embedded objects. Once these scripts are executed it triggers the web browser to download malware to be installed on the user's machine. Thereby, through a series of HTTP redirections it exploits the web browser's vulnerability. Attackers are also known to inject the malicious software via on-line advertising by attracting users to click on fraudulent web pages by tricking them to provide their private information for downloading rogue soft-

ware e.g., the "fake-AV" attack where users are warned of malware infection, and made to download those fake anti-virus software for a fee [3]. Thus, by triggering a legitimate user to click on a series of HTTP requests that contain advertisements lead to malware containing sites which ultimately inject malware into the user's system. Now, this leads to the further questions about HTTP and how it is used along with the complex on-line advertisements in the spread of malicious software. As we know the Hypertext Transfer Protocol (HTTP) is the underlying feature of most of the data communication happening in the Internet. This protocol is being implemented in browsers to find information in the Internet, thereby, making it a large part of data traffic in networks. The increase in information gathering process by the users from Internet makes it more susceptible and take advantage of the ubiquitous nature of HTTP. Mainly, the attackers are now employing HTTP redirections, which automatically redirects users' requests through a series of intermediate websites [3] before reaching the destined website, to hide malicious software distribution servers to evade detection. These attackers often run immense malicious software supporting frameworks such as corrupted websites, botnet and other hosted servers under their control. They utilize HTTP redirections (e.g, HTTP 3xx, which is described further in Section 2.1.5 in this report) or via dynamically executed scripts to switch from one set of intermediate web servers (e.g., when some of which are detected or shutdown) to another. This is mainly done to escape detection by, e.g, DNS/URL blacklisting, static or dynamic Javascript and web content analysis, or honeynets [4] while retaining control of the malicious infrastructure [3]. Such multiple HTTP redirections in a sequence not only make it more difficult for security analysts to detect the malicious servers; they also allow attackers to use cloaking techniques [3] to launch stealthy, targeted attacks while making them extremely hard to detect. For example, attackers can selectively attack users based, for e.g., on the browsers used (e.g., vulnerable versions of Internet Explorer), or the IP addresses of client machines (e.g., ignoring the known honeynets set up by security analysts or clients running virtual machines, supplying "benign" content to search engine bots, or targeting only government agencies/ corporations of interest) [3]. Hence, efficient and effective analysis of the flow of HTTP traffic by security analysts is a crucial key in the detection of the URLs that redirect legitimate users' requests.

The different patterns of attack happening over web can be identified if there are tools that can help in visualizing most part of the problems and causes that lead to malicious activities. Though there are a number of network traffic and few HTTP flow analyzing tools (described further in this report in Chapter 6) available as per the VizSec Security Community [5], the existing analyzers mostly deal with the detection of anomalies in the network as a whole without much emphasis par-

ticularly in HTTP request flow analysis. The limited functionality and sometimes poor user interface of such analysis tools lead to unsatisfactory results. The difficulty in reading hundreds of lines from packet capture logs also adds to the delay in achieving the results in the manner required. As the spread of malicious codes is never going to stop together with the additional trend of the growth of usage of the Internet, there will always be a threat to the security of Internet and ultimately, concerning the security of the private information of the end user. Therefore, this thesis emphasizes the implementation of visual display of the HTTP request flow extracted from packet capture files. Such visual display will enable a security analyst to effectively and efficiently visualize malicious URLs, if any, from redirected websites which are difficult to detect in a text-based analysis tool.

As stated by Russ McRee in his paper [6], "What you don't see can hurt you", security analysts should try different ways to gain as much information as possible from their networks so that they can make timely correct decisions. Thus, visualization can be a more suitable and feasible way than text based analysis because of the following reasons:

1. Visualization can show much more information in the available viewing space [7].
2. By making use of human perception capabilities it is possible to process information very efficiently [8].

Therefore, visualization enhances and improves the analysis efficiency by increasing the size of information taken in each single process by a security analyst.

In this thesis, visualization methods are applied to assist in the analysis of HTTP traffic flows extracted from the head and body of each HTTP request/ response pair. The outcome of the work is a prototype that first parses any packet capture files and then, with the aid of visualization techniques, displays a graph on a browser. The prototype should enable a fast identification of any malicious URLs and the relationship between redirected requests, convenient investigation of interesting minute details. Ultimately it should help an analyst to gain a comprehensive understanding of the HTTP flows and provide effective solutions while mitigating security breaches due to the above mentioned malicious activities.

1.2 Scope

As mentioned in Section 1.1, visualization is the ultimate goal of this thesis, and it is understood that it involves some knowledge about computer graphics theory and its application. However, understanding computer graphics or its techniques

is not the main issue or concern of this work, rather the focus is more on the way such techniques represent the data at a higher level for analysis.

Besides the implementation of the visualization process, the first major step is the extraction of URLs from HTTP requests. In our case, we take into account the *Referer* field to detect the redirected websites due to malicious advertisements, though there is also the possibility to infect an end user through these advertisements without having a Referer field while landing on the destined page. This scenario of not having a Referer field makes it even more difficult to detect such malicious activities as the URLs might be obfuscated i.e., unclear and difficult to comprehend, in the included Javascripts and hence, de-obfuscation techniques need to applied to detect them. As this can be considered as future work, it is not in the scope of this thesis. In the development of the prototype presented in this thesis, we mainly focus on the extraction of all HTTP requests from the *Referer* field and also from the *body* of each HTTP response from varying sizes of packet capture files and then, visually showing the relationship of the traffic flow in the form of a graph.

1.3 Report Outline

This report is divided into eight chapters and a bibliography. The theoretical background about the basic methods and components that make up this thesis is described in Chapter 2. The system design process including description of the functional module composition is provided in Chapter 3. The implementation process of the prototype is explained in Chapter 4. In Chapter 5, the prototype is validated and its results are presented. Chapter 6 provides a survey on the related work that has been done with respect to visualization in different domains and a comparison with our solution. Chapter 7 have a discussion about the whole design, its results from validation in a more comprehensive manner with also an emphasis on the ethics part. Finally a conclusion is given in Chapter 8, with some points related to future work.

2

Background

This chapter presents details of the Hyper Text Transfer protocol (HTTP) along with its parameters and request/ response pair information. With this background, malvertising is presented and finally details about the scope of visualization and its challenges are presented.

2.1 Hyper Text Transfer Protocol (HTTP)

As we are all familiar with the term HTTP and knows about it, this section covers specific details about HTTP that will be needed later to understand the part of visualization.

The Open Systems Interconnection (OSI) [9] model is a framework that specifies and regulates the functions of a communication system by dividing it into seven logical layers. HTTP is a protocol which is present in the application layer of the OSI model. This layer is the one which is closest to the end user [12]. HTTP is a generic, application-level and stateless protocol for distributed, collaborative and hypermedia information systems. It is built on top of the Transmission Control Protocol/Internet Protocol (TCP/IP) communication protocol which functions as a request-response protocol in the client-server communication model [10].

2.1.1 Overview of HTTP

HTTP is the protocol where hypertext is interchanged and transmitted. Hypertext is basically an ordered text which uses hyperlinks. Hyperlinks are references to a

document within a document [10]. There are three important features that makes the HTTP so powerful [11]. They are as follows:

1. The HTTP is connectionless i.e. after a client sends a request to the server it disconnects and then waits for the response. Once the request is processed, the connection is re-established in order to send the response to the client. Here, the client is the browser which initiates the request[1].

2. The HTTP can be used to send any type of data as long as both the client and the server know how to handle it.

3. The HTTP is a stateless protocol which means that the state of both the client and the server is valid per request. This feature makes it possible for the browser not to keep hold of the request/response information between different request/response pairs over the web pages.

In order to understand the flow of HTTP request/response between the client and server, one need to first have an idea of a web application's architecture and where in that HTTP fits in. HTTP, being a request/response protocol, is based on the architecture where web browsers, search engines etc. act like HTTP clients and the web server acts as the server. A request to the server is made by the client in the form of a request method represented by Uniform Resource Identifiers (URIs) more specifically Uniform Resource Locators (URLs) and by the version of the protocol and probably the content of the body. The response from the server as received by the client contains three parts: information about the protocol version of the message, success or failure status code and the body of the content of requested data.

2.1.2 Parameters in HTTP

There are different HTTP parameters which are used in the communication and should be known in order to understand the message structure used in the request/response. Listed below are a few of the important parameters of HTTP [11].

1. HTTP Version: The first line of the HTTP message indicates the HTTP version field. For example: HTTP/1.0 or HTTP/1.1.

2. Uniform Resource Identifiers (URIs): These are formatted strings which associate a resource such as a website, web service, document, image etc. with a name, location etc. Depending on the context of its use, URIs can be

[1]Actually, never versions are more persistent

represented in two forms: absolute and relative form. In the absolute form, URIs always starts with a scheme name followed by a colon. For example: ”http:”. Uniform Resource Locators (URLs) are a subset of URIs that refers to resources by their location rather than by other characteristics. Example of a URI: http://google.com/%eith/home.html

3. Date & Time formats: The date and time format of HTTP must be expressed in Greenwich Mean Time (GMT) and care should be taken as the HTTP-date is case-sensitive [12]. For example: Sun, 15 Feb 2015 10:30:45 GMT

4. Character Sets: The character set refers to the mechanism of converting a sequence of octets into a sequence of characters which are identified by case-insensitive tokens used as charset values. For example: US-ASCII or ISO-8859-1 [12] are valid character sets. By default the value is US-ASCII if no value is specified.

5. Content Coding: This is mainly used to compress or transform a document without losing its identity or information before transferring it over a network. The document is stored in the encoded form, transmitted and only decoded by the receiver. The content coding values can be found in the Accept-encoding and Content-encoding header field of the message (details about this can be found in the subsequent sections). The content coding values are registered by the Internet Assigned Numbers Authority (IANA) which includes the following content coding as tokens [12]:

   (a) gzip- developed by the file compression program ”gzip” (GNU zip)

   (b) compress- developed by the UNIX file compression program ”compress”

   (c) deflate- produced by the combination of ”zlib” and ”deflate” format

   (d) identity- it is the default encoding format and used only with the Accept-encoding header field.

6. Media type: The type and subtype of the Internet Media types are represented in the Content-type and Accept header field of HTTP messages. For example: ”Accept:image/jpeg” where image is the type and jpeg is the subtype of the Internet Media Types.

7. Language Tags: In order to communicate information or data between human beings, the language spoken, written or conveyed needs to be identified which is achieved by the ”language tags”. It is used in the Accept-language and Content-language header field of HTTP messages. It consists of a primary language tag and a series of sub tags. For example tags include: en, en-US, en-cockney.

2.1.3 HTTP-Message

To establish a connection with the server, HTTP makes use of the URI to recognize a resource. After the connection is established, HTTP messages in the form of a request from client to server and response from server to client are sent over the network. The format of these messages resemble to one that is used by Internet email or Multi-purpose Internet Mail Extensions (MIME) [11].

2.1.3.1 Message Types

The request and response messages uses the following format for the transmission of the payload of the message [12].

HTTP-message = Request | Response ;HTTP/1.1 messages

Both type of messages: request and response include a start-line, zero or more headers, an empty line indicating the end of the header fields, and a message-body [12]. The message start line is represented in the following way [12]:

start-line = Request-Line | Status-Line

The example showing the start line is as follows:

GET /helloworld.htm HTTP/1.1 (Request-Line sent by the client)
HTTP/1.1 200 OK (Response-Line sent by the server).

2.1.3.2 Message Headers

There are four different message header fields included in the HTTP message. They are: General-headers, Request-headers, Response-headers and Entity-headers. The general format of the header field consists of a name followed by a colon (":"). The order of the receipt of the header fields is not significant but it is considered as a good practice if the general-header is sent first, followed by the request-header, response-header and then the entity-header field. The generic syntax of the message-header is as follows [12]:

message-header = filed-name ":" [field-value]

2.1.3.3 Message Body

The message body is an optional part of the HTTP message but if it is present it is used to transmit the entity-body which is associated with the request or response. If the message body is present in a request, it implies that the nature of the message body is included in the Content-type and Content-length header fields. In case of response, the inclusion or exclusion of the message body in the message is dependent on both the request method and the response status code. The actual

HTTP request and response information/content is carried by the message body.

2.1.4 HTTP-Request

The HTTP request is sent by the client i.e. the browser to the server in the form of a request message which contains a request line, zero or more header fields, an empty line marking the end of the header fields and an optional message body [11]. The request line includes the type of request method that indicates the action that needs to be performed. There is also a request URI included in the request line which helps in identifying upon which resource (i.e. website, document or web service) the request should be applied. For example, if the request method is "GET", it indicates that information can be retrieved from the server with the given URI whereas the method "POST" is used to send data to the server. For example, if a client wants to retrieve information directly from the origin server, a TCP connection with port 80 of the host "www.chalmers.se" would be established and send the following lines:

GET /education/master_thesis.html HTTP/1.1
Host: www.chalmers.se

2.1.4.1 Request Header fields

For the client to transfer information about the request and also about itself, the request header fields help by acting as the request modifiers. Below is a list of a few of the important header fields [12].

1. Accept: used to specify the media type
2. Accept-Charset: specifies which character sets are relevant for the response
3. Accept-Encoding: specifies the document encoding method used by the browser
4. Accept-Language: specifies the preferred web server's language
5. Host: specifies the requested resource's Internet host and port number which is obtained from the URI
6. Referer: specifies the URL of a previous web page that was followed by a click from the original requested web page
7. User-Agent: specifies the software of the client that initiates the request

2.1.5 HTTP-Response

Once the client sends the request to the sever, the request is processed by the server and it responds to the client with an HTTP response message. Like the HTTP request message, the HTTP response message also has similar fields, for example, status-line, zero or more headers etc. The message status-line consists of the following format [12]:

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

where SP is the US-ASCII for space, CR is for carriage return and LF is for linefeed.

The Status-Code stands for a 3-digit number that represents the result or the status of the request. For example:

1. 1XX: Informational (the request is being processed)
2. 2XX: Success (the request was successfully processed)
3. 3XX: Redirection (in order to complete the request, further action is required)
4. 4XX: Client Error (the request contains bad format)
5. 5XX: Server Error (the server failed to process the request)

2.1.5.1 Response Header fields

The information that cannot be placed in the status line is included in the response header fields which are passed on to the client as a response. These header fields provide information about the server and also information about the requested URI. Below is a list of a few of the header fields included in the response sent by the server [12].

1. Server: specifies the software used by the server to process the received request. For example, Server: Apache
2. Accept-Ranges: specifies the approval of the range requests for any resource. For example, Accept-Ranges: bytes
3. Age: specifies the time in seconds of the generated response by the server to the client.

2.2 Malvertising

With the increased usage of web applications such as on-line transactions, documentation, education etc. has lead to web services being an important and popular choice for users on a daily computing basis. Though it has become quite popular, it comes with a price i.e. it has lead to attacks like distributing malicious software using web applications as the platform. If a system is infected with any malicious software, it ultimately leads to the compromise of personal information, attaining control of the victim's system or other such variants which can be quite dangerous. But then the question arises how the victim's system gets infected with the simple use of web browsers. The malware i.e., the malicious code or software gets injected when a user visits malicious websites (mostly unintentionally or by HTTP-redirection). When a user visits any website either for personal or commercial use, the user comes across a number of on-line advertisements (ads) being posted along side of the requested web page. These ads have become popular as they provide a significant amount of yield to the publishers, therefore, making those ads as attractive as possible will definitely become an addition to the revenue [13]. However, these benefits have led to different types of malicious activities being carried out by attackers. In these web-based distribution of malicious software, the channel of attack is initiated and established by potential victims which ultimately lowers the defense barrier for attackers to cross [13]. These attacks are mostly pull-based which feature techniques divided into two categories [14]. One technique is based on social engineering tactics where a website visitor is enticed to download and execute the malware. The other technique is the focus in this thesis as it automatically leads to download and execution of the malware in the victim's system by using deceitful methods of malware incorporation in browsers. These generate a successful exploitation chain which continues to the execution of malicious code. One of the famous example of malvertising is the New York Times [15] that happened in late 2009. Some visitors to the website of The New York Times were asked to buy fake anti- virus software as their system screens were filled with images showing a scan of computer viruses. This malicious advertisement took over the browsers of those people visiting the website. The software that was asked to download was a rogue and intelligently designed to infect the user's machine than a useful program for protection.

There are different modes of infections which are used to inject malicious advertisements in a vulnerable domain by an attacker. Once they are triggered they exploit this vulnerability to inject the web malware. The modes are described as below [16]:

1. **Malvertising through malicious widgets and redirections**: The use

of widgets in advertising and traffic redirection has some flaws which pose high risks to the domains where it is used. Once such widgets are installed it facilitates in adding third party content which the attackers use to exploit the domain of the publisher of the advertisement by traffic redirection.

2. **Remote malvertising with hidden iframes**: HTTP uses iframes to embed one web page into another which can also be used to load dynamic content for advertising. Hidden iframes can be used to inject codes that help in the spread of malware. Attackers can also hide malicious software by Javascript obfuscation techniques to encode malicious links [16]. They load malvertisements in these iframes to inject infections and thereby, making the detection process difficult.

3. **Malvertising through infected Content Delivery Networks**: Content Delivery Networks (CDN) are the preferred method of infection injection by attackers because if any CDN web servers get infected it can cause wide spread distribution of malicious codes. CDN are basically third-party ad servers that provide information to different domains on the web and therefore, if a parent server gets infected it will lead to infection of all its child servers and the chain continues [16].

4. **Malvertising through malicious banners**: In this mode of attack scenario, attackers make use of SQL injections or XSS flaws in websites that are hosted by a server and thereby, take full control of a website. Attackers update the database with malicious iframes and when a user visits such a vulnerable website, malicious banners are displayed which, if clicked, leads to user system infection [16].

In general the process of exploitation of websites by attackers happens by injecting suspicious content in the vulnerable web browser [14]. The content that is injected is basically a link which after a number of redirections leads to another web page which is intelligently designed to host a script formulated to exploit the web page and ultimately, the web browser. Other kinds of injections include the exploitation of image links and other external information placed in a HTML file that is used to exploit web pages without even requiring compromising a web server [14].

There are plenty of solutions available to avoid such modes of attack such as installing widgets with proper access control, malware monitoring systems, system updates and patches. However, the boundary of web usage is so wide spread that also the attack scenarios widen broadly. In order to mitigate such attacks one needs to have effective tools to detect and analyze such malicious advertisements. This thesis work mainly focuses on the attacks carried by malvertising through redirections.

2.3 Data Visualization

This section presents an overview of what in general visualization is along with its scope and challenges. This section also provides some background information and features about the plotting schemes that have been used in the design of the prototype.

2.3.1 Overview

A picture is often cited to be worth a thousand words and, for some (but not all) tasks, it is clear that a visual presentation- such as a map or photograph- is dramatically easier to use than a textual description or a spoken report [17]. By having a visual display or representation, it becomes even more attractive to provide information regarding context, to provide feedback for changes and to be able to select regions. Data or Information Visualization is a research field that provides interactive visual representation of abstract data for an increased understanding or learning. Data Visualization is sometimes also confused with the term *infographics*. However, they are conceptually quite different. Basically, infographics are the manually laid out or sketched representation of a limited amount of data. Being manually done, it is difficult to change or update any data if required. On the other hand, Data Visualization refers to the representation of data that is drawn algorithmically, can be regenerated easily and can handle large volumes of data. With the growth of computer usage and the data being generated, the ability to store and collect these large volumes of data is growing at more rapid rate than the ability to analyze it. The process of analysis of these complex data is difficult but is an important aspect in many fields. However, lots of money and time are being spent for getting the best results of a packet inspection as the possibility of proper analysis of information by applying tools is quite limited in number. Visualization aims at bridging this gap by employing more intelligent means in the analysis process [18]. The main idea behind this is visual representation of data, direct human interaction, drawing conclusions and finally making better decisions. In a time-critical situation, the decision-makers or analysts should be able to make effective decisions by enabling them to examine huge, multi-dimensional, multi-source and time-varying streams of data [18]. To be more precise, the process of visual analysis is iterative in nature which includes information collection, pre-processing of data, representation of knowledge, interaction and making decisions.

Ben Shneiderman [17] describes type by task taxonomy of information visualizations. This includes seven data types where all items have attributes and a basic search task is to select all items that satisfy values of a set of attributes. The seven

data types are described below [17]:

1. **1-dimensional:** These datasets are linear in nature which include textual documents, and program source code; basically each item in the collection is a line of text with a string of characters. Example: A bifocal display which provides detailed information in the focus area and less information in the surrounding context area [19].

2. **2-dimensional:** This includes planar data which includes geographical data, floor plans or newspaper layout where each item covers some part of the total area. Example: Geographic Information Systems.

3. **3-dimensional:** This data type includes real world objects which have volume and some kind of relationship with other items such as the human body, molecules, buildings. Example: There are quite a number of examples in 3-dimensional computer graphics but very few with respect to information visualization.

4. **Temporal:** These are time lines which are different from 1-dimensional data and are widely used for medical records, historical representations. Example: Tools for visualization of time include the perspective wall [17] and LifeLines [20].

5. **Multidimensional:** This kind of data type includes relational or statistical databases where items with n attributes become points in n-dimensional space. Example: The early HomeFinder developed dynamic queries and sliders for user-controlled visualization of multidimensional data [17].

6. **Tree:** These data types are a collection of items where each item is a link to a parent item with multiple attributes basically having a tree structure. Example: The tree map approach is applied to computer directories, sales data, business decision-making [21], and web browsing [17].

7. **Network:** These data types are an extension of the tree which include networks like acyclic, rooted, directed. Example: Network visualization is an old but still imperfect art because of the complexity of relationships and user tasks [17]. An ambitious 3-dimensional approach was an impressive early accomplishment [22], and new interest in this topic has been spawned by attempts to visualize the World Wide Web [23][24].

2.3.2 Scope and Challenges

According to Daniel et al. [18], visual analytics can be defined as a combined form of visualization, human factors and data analysis. This particular field integrates

methodology from geospatial, scientific and information visual analysis with focus on human-computer interaction. This is so because human factors such as interaction, cognition, presentation, collaboration play an important role in the decision-making process. As a whole visual analytics is the process which makes use of visualization as a means to have a thorough analysis of crucial information and the scope is not limited only to visualization. The detailed scope of visual analytics can be depicted from Figure 2.1.

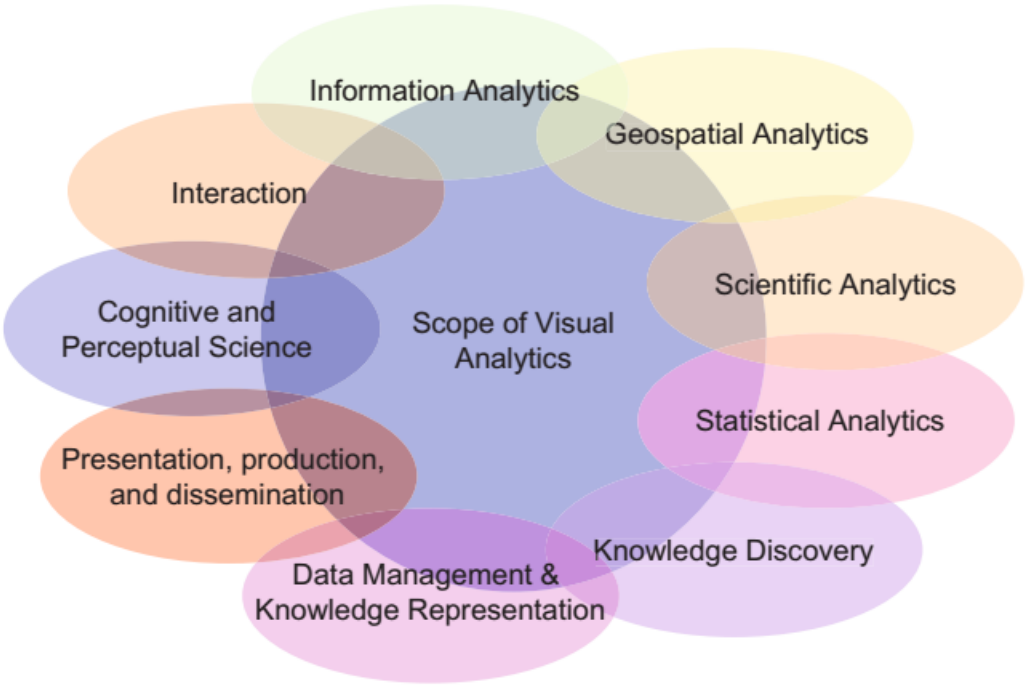


Figure 2.1: The Scope of Visual Analytics [18]

As can be seen from the Figure 2.1, the field of visual analytics can be applied in various areas such as physics and astronomy, business applications, environmental monitoring, disaster and emergency management, security, software analytics, biology, medicine and health, engineering analytics, personal information management, mobile graphics and traffic.

Though there are many benefits and the extent is beyond the scope set in this thesis, visualization in general faces some technical challenges. Daniel et al. [18] have listed a few points on the technical challenges of visualization. Among the challenges listed, the first one lies in the area of problem solving, decision science and human-computer interaction. Meaningful visualizations and clear representa-

tions play an important role in the process of problem solving as it makes it easy for having a better understanding of the technology involved and also comprehending the logic, reasoning faster. Another challenge has been the user's unacceptability. Though many visualization tools have been deployed, they have not been able to overcome the usage barrier. Hence, evaluation of a developed system is important to understand the extent of applicability of the designed tool. Data streams producing large and new data is also an issue especially in the field of real-time data analysis. For example, the area of network traffic involves monitoring of large volumes of data and it gets difficult to log all data for future use. In such scenarios, efficient and effective methods are needed for data compression and extraction. Such issues also lead to the problem of scalability i.e. the ability to compute and effectively display large data sets and also taking care of the computational overhead. Interpretability is also one of the biggest challenges as it is the ability to correctly understand and interpret data so that correct output is generated from any raw input. In order to get the correct output, preprocessing of data should also be carried out to avoid potential problems such as data enrichment, data reduction, data parsing, data cleaning [18].

As most of the applications comprise of these issues, solving one problem would be possible but it would not solve the overall problem. Therefore, one needs to be very careful during the process of applying visualization to any data set while keeping in mind the current state of the area and its challenges.

2.3.3 Visualization processing features

As it is known that the ultimate goal of this thesis is the design of a web interface to visualize HTTP flow, this section presents some background knowledge about the different features that are used in the process of visualization.

Humans perceive certain forms of data rapidly as the eyes have the capability to process certain features in parallel and store it in the memory. This rapid capturing and perceiving ability can be made into effect which will help in easily identifying and differentiating network events. For example, Figure 2.2[2] describes the different forms of data representation which can be encoded in visualization for fast identification of special patterns of visual. Therefore, the size, shape, colour, orientation and also the number of forms used in an application are all important from the visualization perspective.

[2]Inspiration from Colin Ware's book [8]

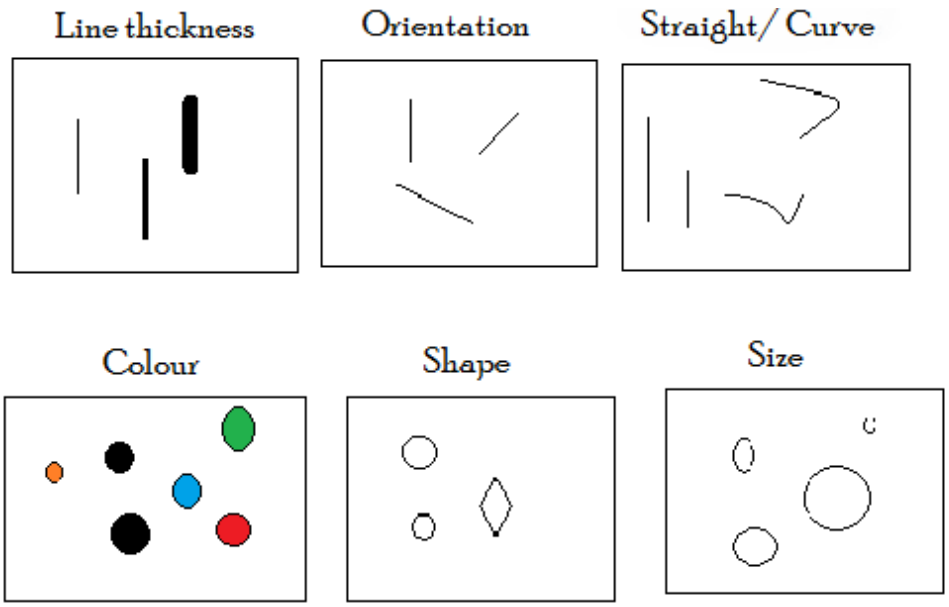


Figure 2.2: Different forms of data representation

The next part is understanding the plotting schemes that is specifically used in this thesis. A plot is a technique of representing a set of data in the form of a graph (visual display) showing the relationship between two or more data points. There are several plotting techniques such as line graphs, scatter plot, histograms, bar charts which are applied and adopted widely in data analysis and effectively serve the purpose of a security analyst. In this thesis, scatter plot and force-directed graph plotting schemes have been used for representing the data.

Scatter plot
Scatter plot is a two-dimensional representation of two variables of a data set which is displayed as a collection of points. Each point have the value of one variable showing the position of horizontal axis and the other of vertical axis. A scatter plot is a good way of finding any correlation between points or observing cluster of points. Figure 2.3 illustrates the plotting scheme of scatter plot.

Force-directed
The other plotting scheme that has been used is the force-directed graph. It is an aesthetically pleasing way of drawing graphs. The purpose of the graph is to position the nodes in a two-dimensional or three-dimensional space where all the edges are almost of equal length and force is assigned among the nodes and edges based on their relative positions in the drawing space. In such kind of graphs, the repulsive charge force keep the nodes apart from each other and due to pseudo-gravity

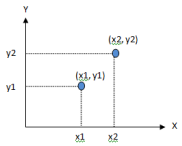


Figure 2.3: Plotting scheme of the scatter plot

Figure 2.4: Force-directed graph [25]

force keep the nodes centered in the drawing space [25]. Figure 2.4 illustrates a representation of force-directed graph.

3

System Design

This section gives a brief overview of the system design that has been incorporated in this work. The design has been used for the collection of required data and also for getting the desired output in the form of a graph.

3.1 Overview of System Design

This section describes the design and methodologies used in this thesis. The design approach is divided into two parts: the first part consists of the method involved in the extraction of the HTTP traffic from pcap files and the second part consists of the methods used in visualization.

The end result of the visualization process, which is described later, consists of two graphs displaying the HTTP traffic flow. One graph displays the linking between the Referer's URL based on time stamp and the other graph shows linking between URLs extracted from the Referer header of the HTTP request and the body of the HTTP response. However, there are only a few differences in the system design approach i.e., the second graph design includes the part of unzipping the body of the HTTP response in order to extract other hidden URLs. Figure 3.1 provides an overall view of the system design approach used in the visualization of the first graph i.e., the scatter plot graph. Figure 3.2 shows the system design approach used in the visualization of the second graph which is the force-directed graph. The choice of the above mentioned two graphs for the prototype is an important aspect and it is further discussed in Chapter 7.

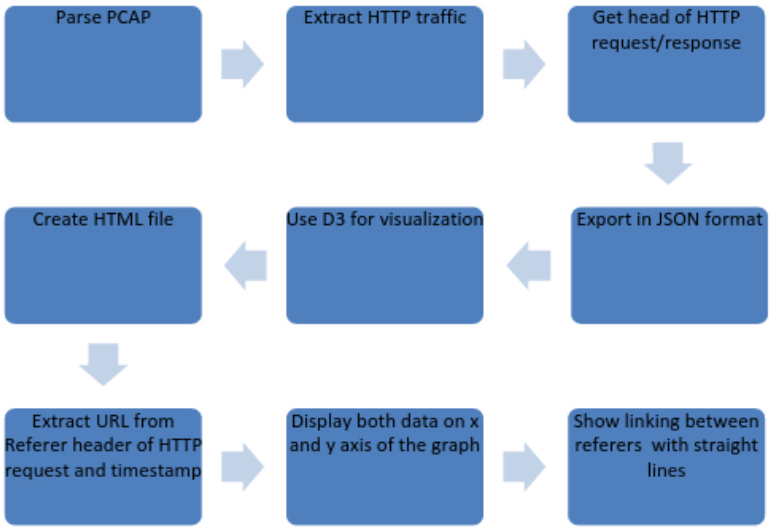


Figure 3.1: Process flow in the design of the scatter plot graph

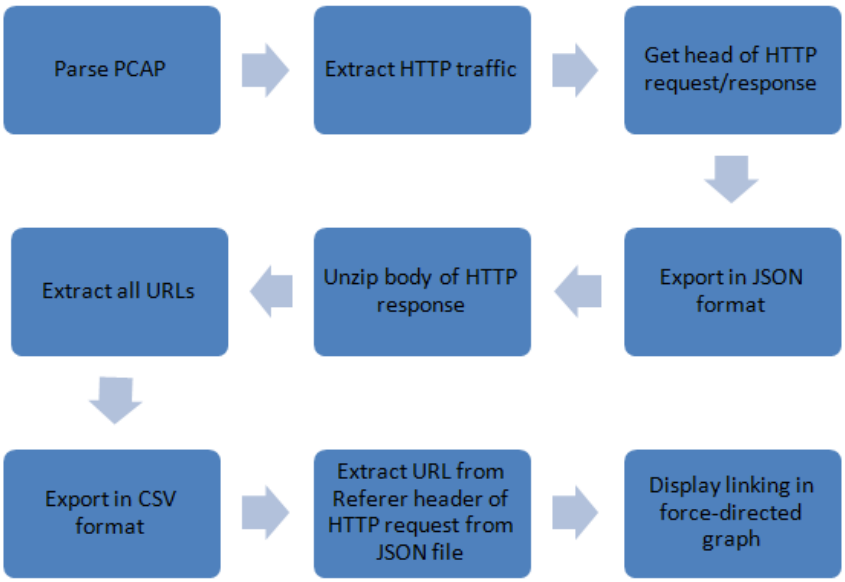


Figure 3.2: Process flow in the design of the force-directed graph

The detailed description of the implementation of the system design process is provided in Chapter 4.

3.2 System Design requirement from the user's perspective

After having an overall idea of the system design process and module composition, it is also necessary to know the design requirements from the user's perspective. As this thesis involves the usage of this prototype by security analysts, it is worthwhile noting the fact that it should effectively and efficiently solve the purpose of viewing the HTTP flow when a user visits a web page. This section provides a list of the general requirements which should be kept in mind while designing such a prototype. It should have [43]:

1. User-friendly interface without requiring much expertise
2. Dynamic view of HTTP flow events over time
3. Ability to parse any pcap files of varying size
4. Clear and concise view showing the relationship between URLs
5. Color coding of points in the graph for easier differentiation
6. Overall situational awareness view

3.3 System Design functional module composition

The design of both the graphs; scatter plot and force-directed graph consist of the decomposition of functional modules from the start of the HTTP traffic extraction until the viewing of the HTTP flow. The choice of tools used in the design are *Wireshark* and *CapTipper*.

The pcap files at first are obtained by using Wireshark. Wireshark is an open-source packet analyzing tool which can be used to capture network data from live network connection. It captures the traffic that goes through a system's Ethernet driver and the traffic can be saved in pcap format [41].

In the next step, "Extract HTTP traffic" as shown in Figure 3.1-3.2, the HTTP traffic is extracted from the pcap files by using the CapTipper tool, which basically parses the given pcap file and extracts the HTTP traffic. CapTipper [42] is an open source tool developed to analyze, explore and inspect HTTP malicious traffic through its powerful interactive console. Once downloaded from its web

page [42] saved locally. After extracting the HTTP traffic, the data i.e., HTTP request and response header information along with additional data is exported in an understandable and analyzable format such as JSON and CSV.

Moving to the visualization process (described in Section 4.2), the web interface is used for the display of the graph and other related details in it. Hence, various components which assist in the development of the web page and the graphics for the graphs are listed below:

1. HTML: It is the Hyper Text Markup Language for creating web pages and thus, it is used for the page contents using the HTML elements which consist of tags in angular brackets. Web browsers can read any HTML files and make them visible for the end user.

2. CSS: It is the Cascading Style Sheets is also a markup language used for giving an aesthetic look to web pages. It mainly enables the differentiation of any documents data or information from its presentation by using CSS elements like layout, colors and different kind of fonts.

3. JavaScript: It is a programming language which is a part of web browsers and is used for providing user interaction after its implementation.

4. SVG : It is the Scalable Vector Graphics which is used in the implementation of vector graphics i.e. image formats with a geometric description which is used for drawing the graph.

These different components seamlessly incorporate into one another thereby, making the web as the successful platform for the representation of any document in an interactive manner. Therefore, in our implementation of the visualization process these components play a major role in the whole design process and ultimately gaining the advantage of having a well defined graph with all the dataset points showing each other's relationship.

This section provided a general description of the functional module composition which has been used during the implementation procedure. The implementation process using the above mentioned modules is discussed in Chapter 4.

4

Implementation

In order to meet the general requirements summarized in Section 3.2, the implementation of the prototype makes use of the most suitable components which is incorporated in the system. All the selections for this design framework and implementation have been carried out after undergoing a survey on the existing tools for viewing network activity which has been described in Chapter 6 and comparing it with our prototype usage.

This chapter will include the complete implementation process and principles. The chapter is divided into two parts: Data extraction and Visualization. The first part explains the methods used in the extraction of required data and the second part explain the methodology of visualization process. The language that is being used for the implementation is Python as it is very easy to use along with its other libraries and modules.

It is also important to note the system configuration used for implementing the prototype. The configuration is described in Table 4.1 which has been maintained throughout the development and validation phase of the work.

The implementation of the prototype consists of two parts; the Python program which stores the data and the visualization interface which displays the given data. The Python program that is written for carrying out the implementation of the prototype is *Main.py* which is the sole file that is used for extraction of URL, decompressing text file and generating the output in JSON format.

OS version	Ubuntu 14.04
Language used	Python 2.7
Tool for packet capture	Wireshark
Tool for parsing pcap files	CapTipper
Browser for viewing graph	Firefox

Table 4.1: Table showing the required system configuration

4.1 Data Extraction

This section of the thesis describes the methods used in capturing packet data to storing of data to be used in the second part of the implementation i.e., the visualization. It also provides a detailed description of the commands used along with the explanation of important modules implemented. No database has been used for storing and manipulating data except a few collections of files as it is easy to implement that for such a prototype and also fast for experimentation.

4.1.1 Parsing Packet Capture Files

As described in the Chapter 3, I decided to use open source tools to speed up the implementation of the prototype. The choice of the tool to parse pcap files is Wireshark. In order to obtain a pcap file, Wireshark needs to be installed in the system. After it is installed, it can be launched and by clicking on an interface under the interface list it starts capturing packets on that interface. An example of the display of packets while capturing is represented in Figure 4.1 and Figure 4.2 shows detail of a request/ response pair captured in a pcap file.

Once the pcap files are saved locally, they can be parsed to extract the HTTP traffic in an analyzing format by using the CapTipper tool. This thesis makes use of this tool with some modification. The required HTTP traffic is generated in a readable and analyzable format as output without launching the interactive console or web server of the CapTipper tool. The manipulation that is done is basically by importing a few important modules of this tool to the Python program *Main.py* with the addition of the use of Python dictionaries to store the relevant data and thereby, getting the desired output in JSON format. A few snippets of the code in Main.py can be seen from Figure 4.3 and all the source codes can be

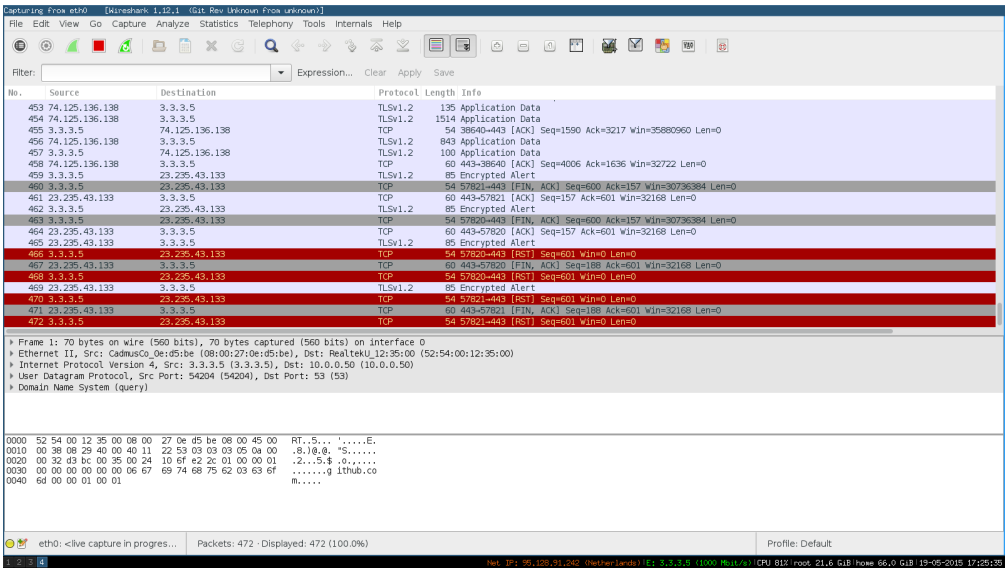


Figure 4.1: Display of packets after the start of capture.

```
GET /?2d5f484b4c495e03544c454242034e4240 HTTP/1.1
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash,
application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, app
lication/xaml+xml, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/
msword, */*
Referer: http://ads.yahoo.com/st?ad_type=iframe&ad_size=300x250&site=167876&section_code
=110717532&cb=1388703600.827840&publisher_blob=${RS}|eIvGkmKKH1z_RwopUsVtygnFVPH4gFLF73A
ADA4J|978550093|LREC|1388703600.827840|${SECURE-DARLA}&yud=smpv%3d3%26ed%3dzxE1dA7ngBKTw
GI.lLm2.Sr9G5Ou3GCxm8467DVrYY3E3xcbjOZvoXXoOSZuroyN9S3h.JuzSRvj92qBSAqkxmgZkS3Zkruust7Gr
_d7e3ArwlyQjyvBiO2q3DqnoZoR.xxXYo5J4GA_TLCs7pb1LJsgAx20sYdNjTHgTD4ZZ.cxl1nJ8mmnj8K218rzP
qzAa3a78AgA._91&pub_url=http%3A%2F%2Fmail.yahoo.com&_msd=1&_xcf=0&rmxbkn=0&_cbv=90345386
3
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0
.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C; .NET4.0E; InfoPath.2)
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: 201116.ry.ro.jxugiyiazd.irritatedpound.net

HTTP/1.1 200 OK
Date: Thu, 02 Jan 2014 22:59:22 GMT
Server: Apache/2.2.26 (CentOS)
X-Powered-By: PHP/5.3.27
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html
```

Figure 4.2: PCAP file showing HTTP header information.

accessed from the Github repository[1].

The modules that are imported are *CapTipper.py*, *CTCore.py*, *CTServer.py parse_pcap.py* and *CTConsole.py*. A few lines of code are commented out in each of the original files as they are not required in the implementation of the prototype. The functions *all_conversations*, *show_hosts*, *hosts* and *client* included in Main.py are

[1] `https://github.com/soni1088/Visualization`

```
1  import CapTipper
2  import sys
3  import CTCore
4  import re
5  import pprint
6  import json
7  import csv
8  import zlib
9
10
11 from CTCore import show_hosts
12 from CTCore import all_conversations
13 from CTCore import hosts
14 from CTCore import client
15
16 from CTConsole import console
17 from collections import defaultdict
18
19
20
21 #To extract the referer,host & uri#
22 def info(): ...
```

Figure 4.3: Python program *Main.py* used in the implementation

imported from CTCore.py file of the CapTipper tool as these functions help in the extraction of data such as URLs, all the HTTP request/response header information, shows the flow of hosts along with a unique identification number (ID) which is used later in the visualization process described in Section 4.2 as the Referer ID. It also provides information about the client including its IP address, MAC address, and User-Agent information from the client function.

4.1.2 Storing Data in JSON format

Once the relevant data that is required in achieving the ultimate goal is figured out, the data needs to be presented in a format which is clear, understandable and easy to analyze. As most of the data, i.e., HTTP request/ response header information is in the form of attribute-value pairs, using JSON as the format fits well into the scenario. Code for parsing and also generating the data in JSON format is easily possible in Python. The JSON file that is generated and used later in our visualization is *graph.json*. Figure 4.4 shows the representation of HTTP request/ response header information in JSON format.

The *Header_Dict()* function included in Main.py serves the purpose of generating data in JSON format. To run the program for generating the output, the following command is used:

python Main.py <pcap file name>

```
[
    [
        {
            "req_header": {
                "Accept-Language": " en-US,en;q=0.5",
                "Accept-Encoding": " gzip, deflate",
                "Connection": " keep-alive",
                "Accept": " text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
                "User-Agent": " Mozilla/5.0 (Windows NT 5.1; rv:30.0) Gecko/20100101 Firefox/30.0",
                "Host": " www.borange.com",
                "Referer": " http://inspirationti.me/websites/simple",
                "Pragma": " no-cache",
                "Cache-Control": " no-cache"
            },
            "uri": " http://www.borange.com/",
            "sources": [],
            "host": " http://www.borange.com/",
            "id": 0,
            "destinations": [
                1,
                2,
                3,
                4,
                5,
                6
```


Figure 4.4: Extracted HTTP header information of one request in JSON format.

The next step in the implementation process is how the prototype is used to generate the first graph; scatter plot graph which is described in the Section 4.2.

4.1.3 Additional data

The text files in the name of *hosts.txt*, *details.txt*, *reqhead.txt*, *reshead.txt* and *resbody.txt* also generated by running the Main.py program contain other additional information. These text files help in placing an additional value to the information gathering process. As these files are in text format, the information displayed can be easily viewed and also easy to understand and interpret especially for a security analyst who can avoid the pain of reading and scrutinizing hundreds of lines from a pcap file.

4.1.4 Decompressing the Body of HTTP Response

As it was mentioned earlier in Section 3.1 about the implementation of a second graph, in order to get the hidden URLs, we need to extract them from the body of HTTP Response. However, as it can be seen from the snippet taken from the generated JSON file in Figure 4.5 that the value of the Content-Encoding header field of each HTTP Response is *gzip*, it has to be first unzipped to be able to read it in normal text form.

The decompressing of the body is done in the Main.py by adding a line of code to the *resp_body()* function which is shown in the Figure 4.6.

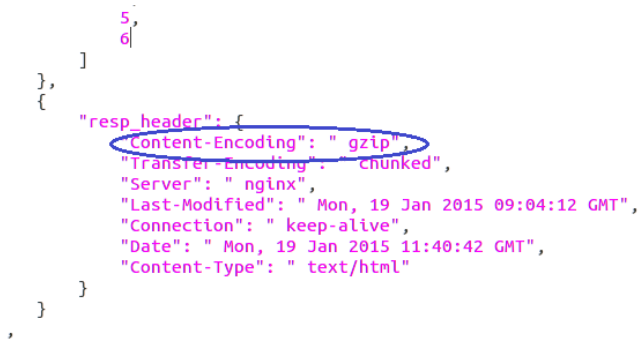


Figure 4.5: The oval drawn shows the gzip value of the Content-Encoding header

```
#Information of response body#
def resp_body():
    fr= open('respbody.txt','w')
    sys.stdout= fr
    for i in range(len(all_conversations)):
        try:
            print "Res_Body[{}]=".format(i),zlib.decompress(all_conversations[i].res_body, zlib.MAX_WBITS+16)
        except:
            print all_conversations[i].res_body
    fr.close()
```

Figure 4.6: The lines of code showing the use of zlib and also how the all_conversation is used to extract data

The functions in the modules of *zlib* library that is used here allows compression and decompression and it is easily compatible with Python modules. The *zlib.decompress()* function decompresses the given data in the string by returning a string of uncompressed data. Then, the uncompressed data is stored in a text file which in this implementation is named as *respbody.txt* as it contains the uncompressed data of the body of HTTP Response. This text file is then used to search for all the URLs in each of the body of HTTP Response by performing a regular expression (regex) search. The regex search is basically a search of patterns and strings by the *re* module of Python. In this case, the regex search is performed for finding the patterns of URL by using the syntax as shown in Figure 4.7.

```
re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', lines[i])
```

Figure 4.7: URL pattern search in regex

The re.findall() function along with the regex checks if any particular string in the text file matches the given regex and returns all non-overlapping lists of strings that matches the given pattern. The pattern basically represents a URL format

starting with http or https with colon and double forward slash and then all alphanumeric characters including special characters. It returns the string in the order it is found in the given file.

Once the desired URLs are retrieved, it is stored in a CSV file under the column names newid, source and target. The newid column shows a new identification number given to each extracted URL, source column represents the Referer ID which holds the URLs found in each of its response body and correspondingly, the target column stores the found URLs from the uncompressed text file. This CSV file is further used in the visualization process in designing the second graph which is described in detail in Section 4.2. The screen shot of the CSV file contents can be seen in Figure 4.8.

newid	source	target
1	1	http://support.avast.com/
2	1	http://forum.avast.com/
3	1	http://avast.jobs.cz
4	1	https://blog.avast.com
5	1	https://www.facebook.com/avast
6	1	https://twitter.com/avast_antivirus
7	1	https://plus.google.com/u/0/102077833873126564701/posts
8	1	http://www.youtube.com/avast
9	1	http://static.avast.com/web/i/form-close.png

Figure 4.8: CSV file with extracted URL and other data

4.2 Visualization

This section describes the implementation of the final visualization process of the system. The modules described in Section 3.2 help in the processing of the input data to get the desired output in the form of a graph. As mentioned earlier in Section 2.3 that understanding the flow of a user's visit to a web page can be best depicted if it is displayed in a visual manner and hence, getting an output with points having directed edges is the best one can expect. Therefore, this section describes both of the graph's building methods to get the browser-based interface as the main window showing the relationships of HTTP requests. The interface for the display of the graph which runs on the browser has been achieved using the *D3 Javascript* library [43]. The visualization process also takes care of all the

requirements from a user's perspective as mentioned earlier in the Section 3.2.

4.2.1 Building the Graphs

The prototype in this thesis work is built by making use of the example of D3's scatter plot graph (for first graph) and force-directed graph (for second graph) [43]. The Javascript files named *graph.js* and *force.js* contain the code for the visual interface design of the scatter plot and force-directed graph which is embedded in the main HTML file named *page.html*. This main HTML file includes all the components described in Section 3.3 and provides an interactive interface with buttons to view either or both graphs at a time. It also displays the flow of hosts for reference in text format in detail to give an additional source of information. It also has a *Refresh* image as a button to return to the home page. The full code is provided in Appendix B for easier access and understanding. Figure 4.9 shows the main HTML file which is used for viewing the graphs.

Figure 4.9: Web page *page.html* for viewing the graphs

The CSS elements in our implementation includes font styling, style defined for path of the line in the graph, style for the axis of the graph, for the body and text font, colour, etc., also the style for the circles used in scatter plot graph.

The next part is adding the D3 script :
<scriptsrc="http://d3js.org/d3.v3.min.js"></script>
to the body of the HTML file. One does not need to create or initialize for the D3 to work, just by including the script in the page it grabs the required modules for

its functioning. Hence, once the page is loaded, the body of the page is selected and the SVG element is appended to it. This is common to both the graphs and it is included in the *page.html* file.

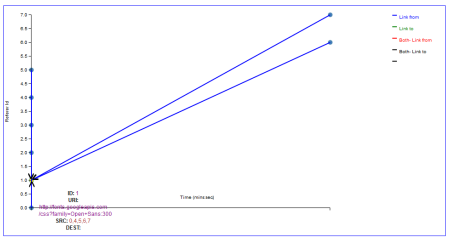
4.2.1.1 Scatter plot graph

This section presents the methods and functions used in order to build the first graph i.e., the scatter plot graph.

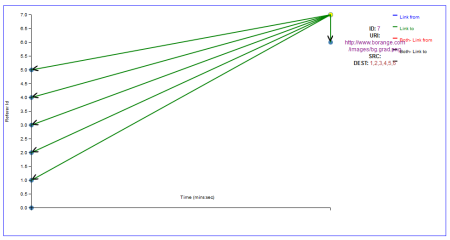
Once the Javascript file named *graph.js* is embedded in the main HTML file, it creates the button called *View Graph* and when it is clicked, the scatter plot graph is displayed. The Javascript file calls the function *graphData()* included in *graph.js* to return the graph display. It also contains the SVG element and other methods for display and positioning of the graph.

The margin and the x and y axis domain is set up in order to fit perfectly into the page. The data to be presented on the graph is extracted from the JSON file by using D3 *forEach()* function. The SVG element then selects the *circle* element which basically means to have circle as a data point corresponding to the dataset. To draw the circles, its radius, centre from x and y axis has to be defined.

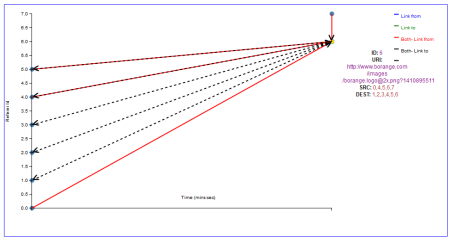
To include user interaction with the graph, transition functions such as *mouseover* and *mouseout* are used. These functions help when a user wants to look into detailed description of the display by hovering the mouse over any circle in the graph. The mouseover function displays the relationship between the URLs by showing coloured directed edges. Now, as it is mentioned earlier in Section 2.3, the directed edges are basically straight lines drawn between two points in a graph with an arrow head showing the direction of relationship. The straight line for the directed edge in the graph is drawn by taking x and y axis co-ordinates of the point where the mouse is hovered and x and y axis co-ordinates of the point's source or destination or both lists. This source and destination list is taken from the JSON file which shows to which and from which the Referer ID is linked. This basically means each request displays if it has been referred from or to other IDs' and in the graph the colour of the directed edge depicts it. The information of each Referer IDs' source and destination list along with ID and URL is also shown as a tool tip which displays information when the mouse is hovered over any point in the graph along with the directed edge. There are also different colour coded lines for showing directed edges based on the source and destination lists which can be seen by hovering the mouse on different nodes in the graph. These graphs can be seen from the sub figures in Figure 4.10.



(a) Graph showing directed edge with node having source list



(b) Graph showing directed edge with node having destination list



(c) Graph showing directed edge with node having both source and destination list

Figure 4.10: Visualization of scatter plot graph

4.2.1.2 Force directed graph

This part of the report explains the methods and functions used to get the force-directed graph as the second graph showing the relationship of the URLs extracted from the body of each response and URL of the Referer.

At first the function *urlData()* is called from the *page.html* which then runs the Javascript from the file *force.js* to display the graph. At first, data is extracted from the csv file where source and target behave as the nodes in the graph and also show the links. The links array contains the objects: source and target and the nodes array contains its properties. The *d3.layout.force()* creates a force function for specifying the link distance between nodes, the size, dimension of visualization and also specifies the arrays of nodes and links. The nodes in the graph are drawn as circles and once the *start()* function runs, the force layout starts its calculations. The calculated value results in a directed edge graph showing the linking between source and target and also show how far each node is connected to other. By clicking once by mouse on any node increases the size of the node greater than

double of its original size along with the font of the text displayed over the node and by double-clicking on the same node returns the original size and font of the node and its value. The graph can also be dragged around and the graph layout keeps moving until they get reasonably close to the value of the *linkDistance*. The resulting graph can be seen from Figure 4.11. In this case also, by hitting the refresh button one can return to the homepage.

Figure 4.11: Visualization of the force-directed graph

5

Validation & Results

This chapter presents the validation of the prototype and its result against different aspects such as scalability, memory usage, and identification of malicious URLs. The chapter starts with a walk-through to demonstrate how easy it is to understand a user's visit to a site through the graphs. Finally, the chapter also presents feedback from three persons from Fox-IT, well versed in the field of computer science owing to their academic background.

As mentioned earlier in Chapter 1, the goal of this thesis is to design and implement a prototype for displaying HTTP flows to enable an expert to analyze the traffic and identify malicious URLs. One of the key requirements is that the prototype is easy to use and that it is able to find the malicious URLs effectively. In Section 5.1, I describe how the prototype would function for a website visit while in Section 5.2; I describe the detection of malicious URLs. Given the large size of data, scalability is also important and such validation results are presented in Section 5.3.

5.1 Example of a user's website visit flow

This section takes an example of a user visiting a website and explains the working of the graph through it. The detailed information of the redirected requests and embedded objects like images having valid Referer header can be seen under the caption "Host flow details" in the web interface.

Considering a use case scenario, suppose the user visits the web page `http:`

//www.nu.nl. The user clicks on an image having a URL http://media.nu.nl/m/m1oxkujaufsv_std640.jpg. In such case, to understand what the source of the image is, the scatter plot graph can be used for viewing it. When the mouse is hovered over the ID of URL http://media.nu.nl/m/m1oxkujaufsv_std640.jpg which is 42 in the scatter plot graph, it displays a directed edge from 42 to http://www.nu.nl whose ID is 0. The directed edge shows that ID 42 has a link from ID 0 i.e., ID 0 is the source of the ID 42. The nodes in the graph also display the directed edges from IDs which have links to another ID in the destination list. In this case, the ID 0 has a link to ID 190 which means 190 has 0 as a valid value in its Referer header. It can also be seen that when the user hovers the mouse over any ID, the subsequent display shows the node's ID, url, src and dest as additional information. Therefore, it gets easier to understand and perceive the HTTP flows in a better way with the help of the visual display. Figure 5.1 and Figure 5.2 show the linking as described in this section.

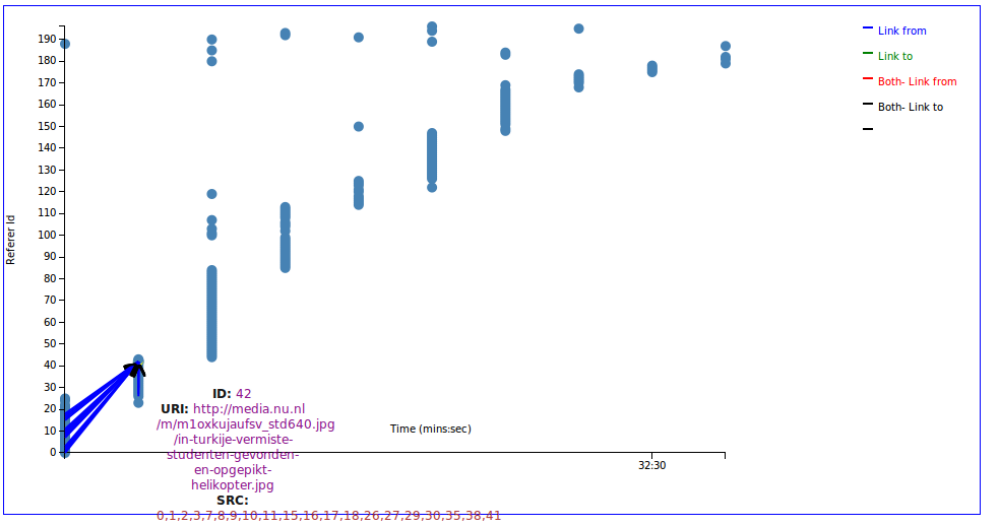


Figure 5.1: The directed edge showing the linking between ID 0 and 42

Now in order to see if there are any hidden URLs from the body of each HTTP response, one would click on the button saying "View force-directed graph". This graph shows for example, the HTTP request with ID 0 containing the URL http://www.w3.org/1999/xhtml in the body of its response and similarly for other requests as well in different colour coded schemes which can be seen from Figure 5.3. Thereby, the graphical interface helps an analyst to view the relationship between the URLs through the directed edges. Thus, it can be seen how an analyst can visualize the flow of HTTP requests just with the clicks of a few

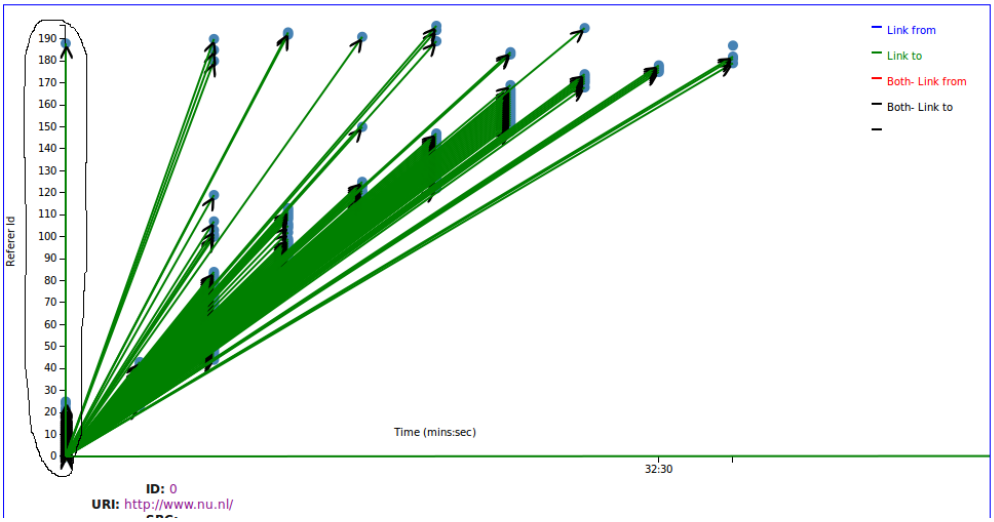


Figure 5.2: The directed edge showing the linking between ID 0 and 190

buttons and also can be used to detect malicious URLs if any.

Figure 5.3: The directed edge showing the linking between ID 0 and the URL `http://www.w3.org/1999/xhtml`

5.2 Identification of malicious URLs

The prototype is tested against a pcap file which is already known to contain a malicious URL in the HTTP flow. It is also known that the pcap file contains 281

HTTP request/response pairs which add to the complexity in the visualization process. However, viewing the malicious URL in the graph is the desired result. When the pcap file was fed into the prototype, it was parsed and the head and body information of each HTTP request/response pair was extracted in JSON and CSV format respectively. By running the web page *page.html* and hitting the button "View Scatter plot graph", it was found that the graph displayed the node with URL "http://adtv.in/crossdomain.xml" which was identified to be the malicious URL and it was also possible to view the source i.e., the parent of the URL causing the redirection. This showed that the prototype could visualize the HTTP flows and identify the malicious URL when there is suspicion about malicious traffic within such large number of HTTP request/response pairs in the packet data and thereby, achieve the goal of this thesis.

5.3 Scalability & Memory usage

The scalability is validated based on two criteria: the time taken to parse large pcap files and the effectiveness of visualization by its interactivity. Due to the restricted access to files with large number of attacks happening on an everyday basis, capturing packets of a client network was limited and hence, the maximum size of pcap file that was possible to have access to was 80 MB. The validation was performed on a 2.90 GHz Intel(5) Core with 8GB of memory. It was observed that the parsing of the pcap file took less than 10 seconds depending on the pcap file size but the output was generated in the desired format within 1 second regardless of the pcap file size. The observation also showed that the time taken to build the HTTP flow graph showing relationships took 2 seconds after hitting the desired button and then hovering the mouse to see the directed edges. It was found that the graph was able to show all the IDs listed in the JSON file which in this case was 281 and was able to display the linking without any obscurity. It can also be concluded that the memory usage of all the installation files and programs used in the running of the prototype took only 158 MB of the memory of the system. It is also important to note how the system works with larger files with number of HTTP request/response pairs greater than 500. It was found that though the graph is scalable to handle such large numbers, however, there is the possibility that nodes in the scatter plot graph will be cluttered and hovering the mouse over the nodes would have to be done very precisely and carefully to get the desired result. However, the force-directed graph under large number of request/response pairs would display the nodes in a much better way as the nodes can be dragged and placed anywhere in the viewing space thereby, making it easier

to analyze. Hence, it can be said that in order to get the best result within the viewing space there is a possibility that each user session is to be made smaller in size where a pre-processor can be used to divide most user sessions into their own files. Then, the prototype can be used on these small files. The validation results obtained from a 80 MB size pcap file with 281 request/response pairs show that the graphical representation is clear to identify malicious URLs under such complexity.

Considering the case of studying the memory usage, it is validated by plotting a graph with pcap size against number of HTTP request/response pairs. The Figure 5.4 presents the results of the validation from which it can be inferred that the memory usage against such criteria grows logarithmically.

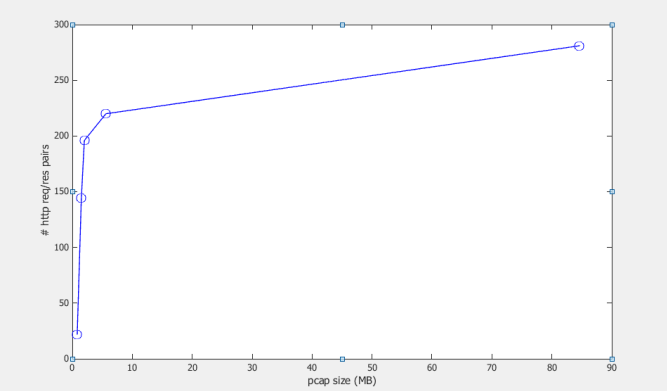


Figure 5.4: Graph showing the memory usage plotted with pcap size against number of HTTP request/response pairs

5.4 Record of the feedback

In order to get the feedback on the usability of the prototype, three different people were asked to use it and provide their opinion. Two were security experts, who are called *expert 1* and *expert 2* in this survey, one of the primary goal audiences of the tool. However, I also wanted to investigate how a novice person would use the tool, so included in the survey is also an intern at the company with no prior knowledge of the subject or the goals of the thesis. The test results is presented in Table 5.1 which corresponds to the answers given by the experts from the company and Table 5.2 corresponds to the answers from the intern. All the people were asked to answer in their own words.

Sample questionnaire	**Expert 1's response**	**Expert 2's response**
How easy was it to install and use the prototype manually?	Very easy	Very easy
How user-friendly is the web interface?	Very easy	Extremely easy
How clear was the visual display of the graphs?	Very clear	Clear
Is the purpose of the prototype achieved?	Easy to visualize HTTP flows	Easy to identify the malicious URL
Any suggested improvements?	Few more details can be added to the web interface	The scale of the viewing space can be increased

Table 5.1: Table showing expert's responses of the questionnaire

Sample questionnaire	**Intern's response**
How easy was it to install the prototype?	Very easy
How user-friendly is the web interface?	Extremely easy
How clear was the visual display of the graphs?	Clear
Any suggested improvements?	The scale of the viewing space can be increased

Table 5.2: Table showing intern's response of the questionnaire

By looking at the feedback received from the experts, it can be inferred that the display of the graphs were not obscured and easy to interact with. Thereby, meeting one of the design requirements in the implementation of the prototype. However, if the responses of the intern are taken into consideration it can be concluded that even novices seem to be able to use the prototype at ease. The barrier to learn the prototype before usage seems to be very low. Although a novice can easily use and install the prototype, prior training is required in order to identify malicious URLs but a novice still would not be as efficient as the experts.

Therefore, it can be concluded from the responses received that the prototype solves the purpose while few improvements can be done such as increasing the

scale of the viewing space to give a more wider view and also adding other details which basically is aesthetically improving the web interface to make it more understandable for the user.

6

Related Work

Application of visualization has become a dominant research topic in recent years as many researchers and analysts have attempted to apply it to the field of computer security and in particular cyber security. Visualization being one of the most active research field [5] among the security researchers community, quite a number of exploration and finding has been done. The scope of visualization is not limited to itself; rather it also requires some knowledge about various fields such as computer graphics, human-computer interaction and data mining, which is the process of finding patterns in large volumes of data.

This chapter describes the work that has already been done in relation to visualization. Though not much work has been done in particular to visualization of HTTP traffic flow, there have been many researches and also papers published with respect to visualization of network traffic, Internet traffic and routing information [6]. It has also been found that visualization tools that have been developed have helped researchers and analysts to easily monitor the network traffic and have a thorough analysis [5]. This would be a difficult task if done manually, as it would have the risk of missing something that is important. Without leaving out the fact that network traffic or HTTP traffic is huge in volume, its analysis is a complex task which can be leveraged by the use of a good visualization tool. The following sections provide information about the working of various tools along with an overall comparison with the prototype that is being developed in this thesis.

6.1 Packet level visualization tools

The RUMINT tool developed by Greg Conti [26] is one of the visualization tools that takes network packets in the pcap format from packet capture tools like Ethereal. It provides seven different visualization windows which helps an analyst to compare very large number of packets. However, the tool is limited to 30,000 packets for comparison. This tool displays a parallel coordinate plot for determining correlation of up to 19 packet header fields, a two dimensional plot i.e., scatter plot that shows information of a combination of header fields. It also provides animated visualization of packets that are derived from IP addresses and port numbers along with a byte frequency visualization that presents a graph of bytes contained within each packet [26]. Another unique feature of RUMINT is that it provides Video Cassette Recorder (VCR) like application which provides playback capability for a clearer view of the events of the network activity over time. Figure 6.1 shows a screen shot of the tool while in action.

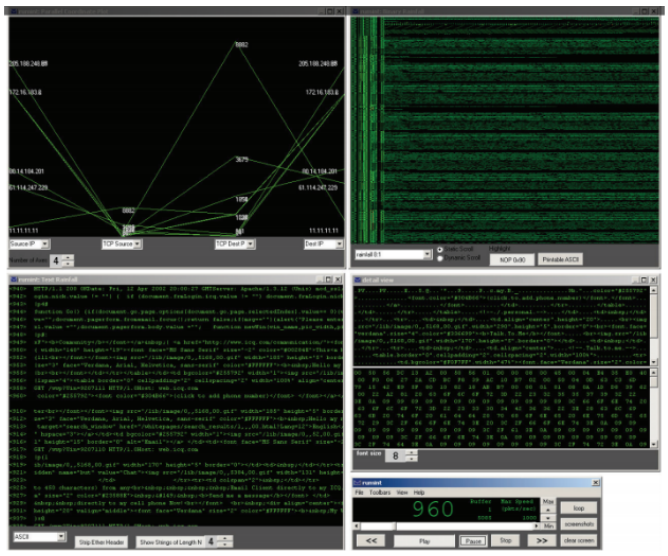


Figure 6.1: RUMINT showing a parallel coordinate plot (top left) for comparison of up to 19 packet header fields, a binary rainfall view (top right) for plotting the raw bits from each packet, a text rainfall view (bottom left) to display printable ASCII characters, one per row, and a detail view (bottom right) to see a single packet in hexadecimal and ASCII [27].

Apart from RUMINT, NetGrok [28] is one among the packet level tools developed by Blue et al. for visualizing computer network usage in static and real-time,

streaming data. This tool combines techniques such as overview, zoom and filter along with force directed network graphs [29] and treemaps [30] visualization. Both the network graph and the treemap display informations such as IP hosts, their bandwidth usage and also groups and links between nodes [28]. The treemap can handle more number of nodes without any blockage as compared to network graph. NetGrok as a tool can read pcap-formatted network traces, capture traces from live environments, data filtration by bandwidth, number of connections and time [28]. As a whole it provides fast analysis of network traffic usage and anomaly detection on a real time basis which can be seen from Figure 6.2.

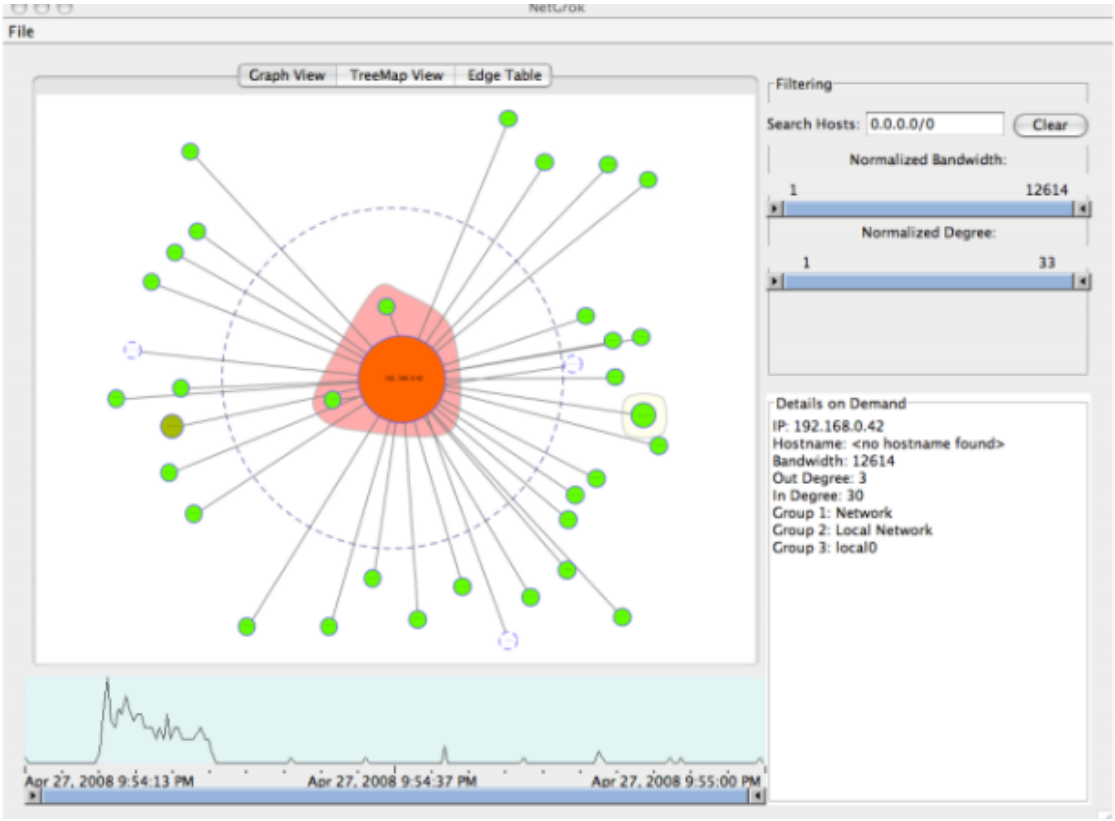


Figure 6.2: NetGrok's visual elements include a main visualization (upper left), a time-line histogram (lower left), a filter and search panel (upper right), and a details on demand window (lower right) [28].

Among the numerous packet level visualization tools developed, Time-based Network Visualizer or The Network Visualizer (TNV) [31], as shown in Figure 6.3, is another visualization tool which is platform independent and is based on Java.

It reads data from any pcap formatted file, captures live packet data, exports or saves the captured file into a database for later analysis over time.

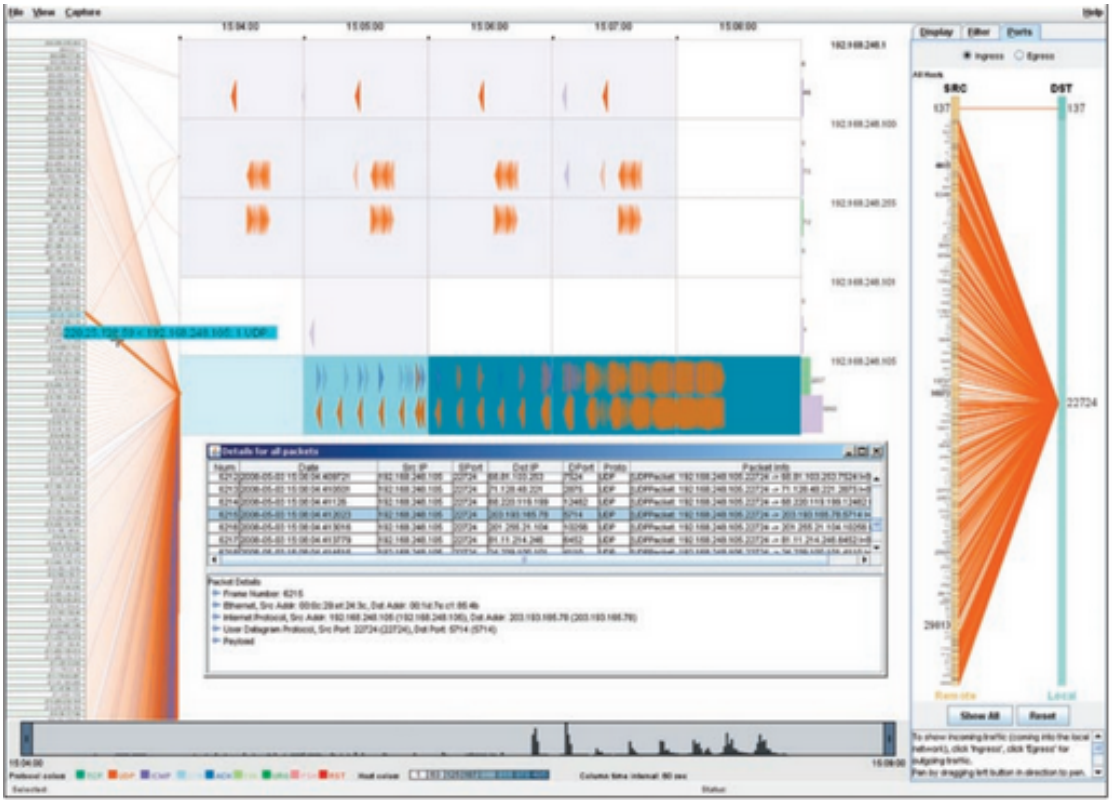


Figure 6.3: TNV showing remote hosts at the left and local hosts in a matrix of host rows and time columns at the right. The matrix cells' colors represent the number of packets for that time interval. Packets can optionally be superimposed on the aggregated local host cells [31].

As can be seen from Figure 6.3, TNV facilitates normal network activity analysis and investigates security related events taking place in the packets. The important aspect of the visualization feature of TNV is that it shows a link between local and remote hosts along with network packets shown as triangles whose point represents the direction of the packet. For a specific time period, a user can also select a cell within the matrix of local hosts to show the details of the packet or activity of the port. In brief, TNV provides an overview of the detailed activity of the network packets with sometimes being slow for large pcap files.

The tools RUMINT, NetGrok and TNV all deal with network data and in packet level which is not the case in the prototype design of this thesis. A comparison

has been made in Section 6.5 which can be referred to for full information.

6.2 Network graph visualization tools

AfterGlow is one of the widely used and downloaded visualization tool which assists in the generation of linked or network graphs with the help of a collection of scripts written in Perl [32]. It is a tool that does not have any particular graphical user interface; rather it is used in command line mode. It is very simple to work with; the input is in Comma Separated File (CSV) format which in itself is generated from a packet analysis tool. This tool can be applied to various resources such as packet capture files in pcap format, email logs and various other kinds of logs for example, firewall logs, web logs. The output generated by AfterGlow is in two formats [32]. One is a dot attributed graph language file with its input being a well known open source library: graphviz and the other format is a Geographic Data Files (GDF), which for example can be visualized using gephi, which is another open graph visualization tool. Figure 6.4 shows an overview of the tool.

Etherape [33] is another network monitoring tool, developed for the UNIX platform. Unlike AfterGlow, it displays network traffic in the form of graphs using a graphical user interface. Packets can be captured from live environments or can be read from Ethereal or tcpdump capture files. The most used protocols are displayed with a node and link color and the user has the flexibility to select any level of the protocol stack to work with. In order to have detailed information of the protocol division and other statistics of traffic, one needs to click on a node or a link which opens another dialog box for scrutinizing. These statistics at the end can also be exported to an XML file for further analysis. Overall, as seen in Figure 6.5, it is an interesting tool which gives a holistic view of the network traffic in an interactive manner.

6.3 Network traffic visualization tools

Oberheide et al. [34] describe a tool called, Flamingo, which is developed for visualizing Internet traffic. This software, as seen in Figure 6.6, is basically used to analyze the Internet traffic flow in real-time and they present an interactive visualization and manipulation tool for easier analysis of network flows for the security analysts and researchers. The environment setup of this software tool is basically

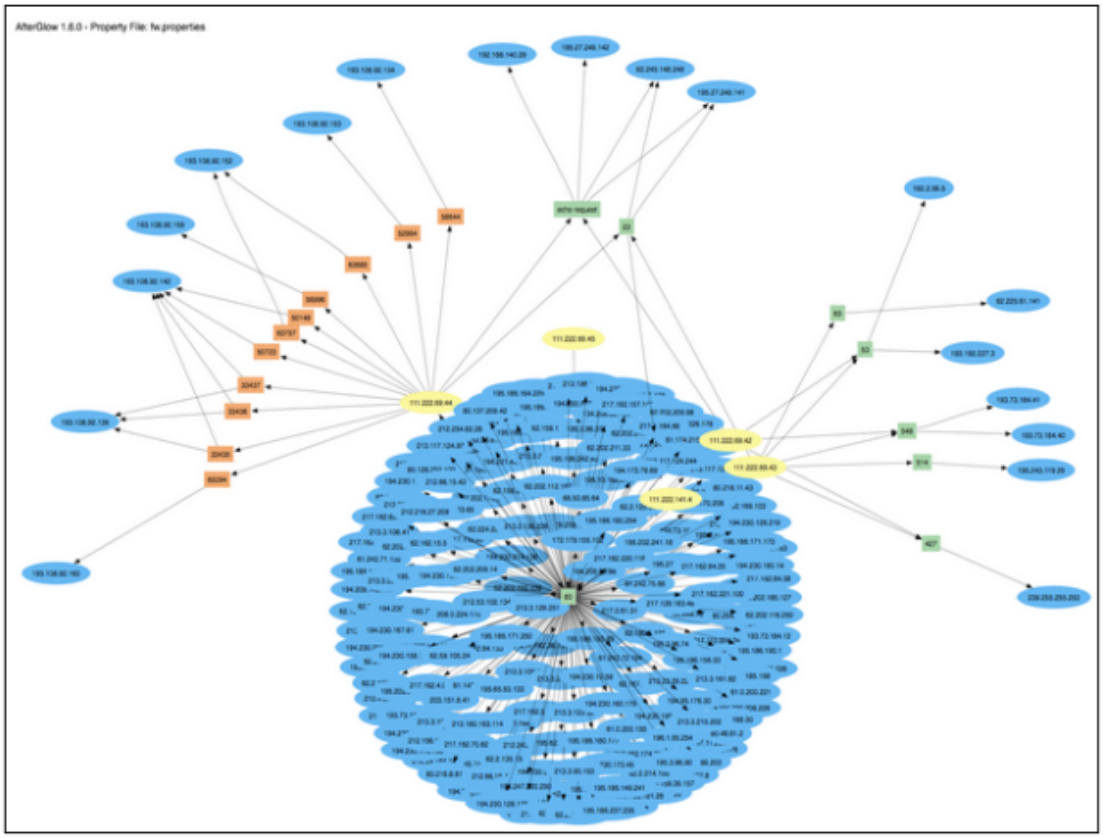


Figure 6.4: AfterGlow showing firewall data (source ip - port - destination ip) [32].

comprised of a client and server where the server receives raw network flow data, processes it and then sends it to the client.

Once the client receives the processed data, it is used for visualization, 3D space navigation and a filtering option which makes it easier for an analyst to extract or monitor relevant information [34].

Similar tools such as VisFlowConnect [35] and NVisionIP [36] have also been developed to visualize network flow data, however, the approach of each of them is partly different. The VisFlowConnect tool provides visualization of network flow data in the 2D plane with source and destination IP addresses in the coordinate axis with an exception of not being able to display the entire IP address space. On the other hand, the NVisionIP tool provides a graphical interface in particular for class B networks. Flamingo [34], in contrast, uses the entire address space to navigate and locate data points of particular interest.

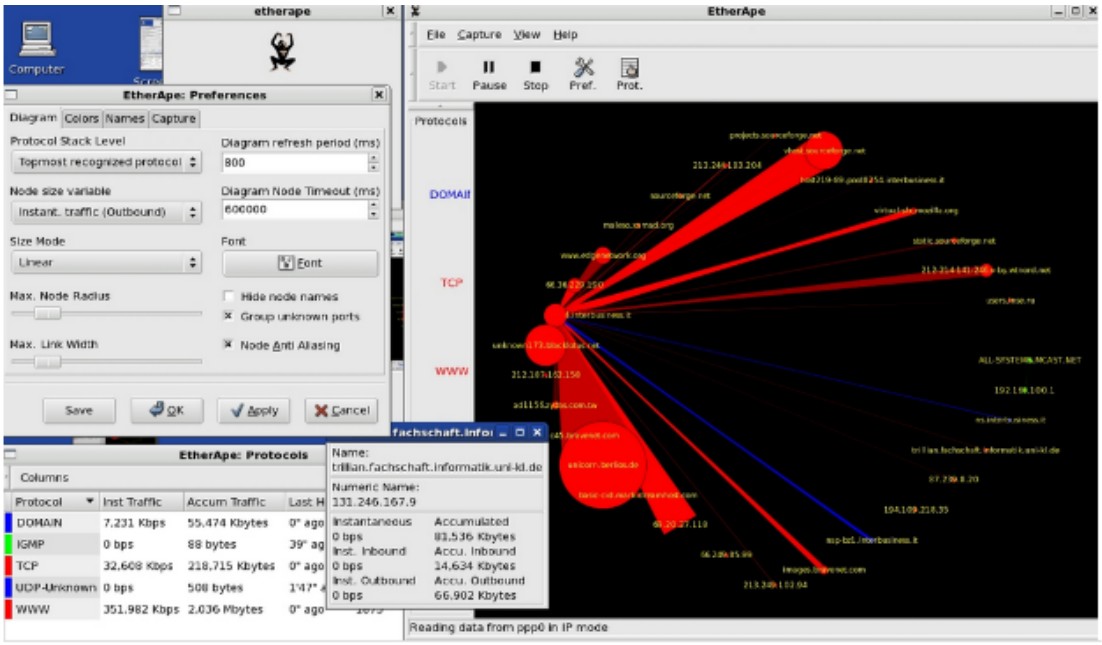


Figure 6.5: Etherape showing a detailed view [33].

Meghan and Peter [37] in their Network Analysis Visualization (NAV) project developed a Java-based tool which helps in evaluating high level occurrences of network traffic. The design of the system works by first capturing log files from live data in real time. At the start of the log files, information like time range for the capture, local network range and mask (which differentiates between local and remote hosts) and also some optional filtering capabilities are fed into the NAV. The output is displayed from any common log format.

The NAV features include, as seen in the above Figure 6.7, an IP wall view, services view and detail view. The IP wall view is located on the left side of the application and provides information about the amount of traffic transmitted either by local or remote hosts. The services view located on the right side of the application shows the amount of traffic generated for particular services over time. This view has the capacity to display up to twelve graphs at a time with eight being the default number of graphs. The detail view located at the bottom of the screen has the capability to provide a detailed view of the data dragged and dropped in its frame. It displays detailed information of data such as source IP address/ port number, destination IP address/ port number, TCP or UDP flags each at a time. NAV as a whole provides textual and also time based filtering of relevant data.

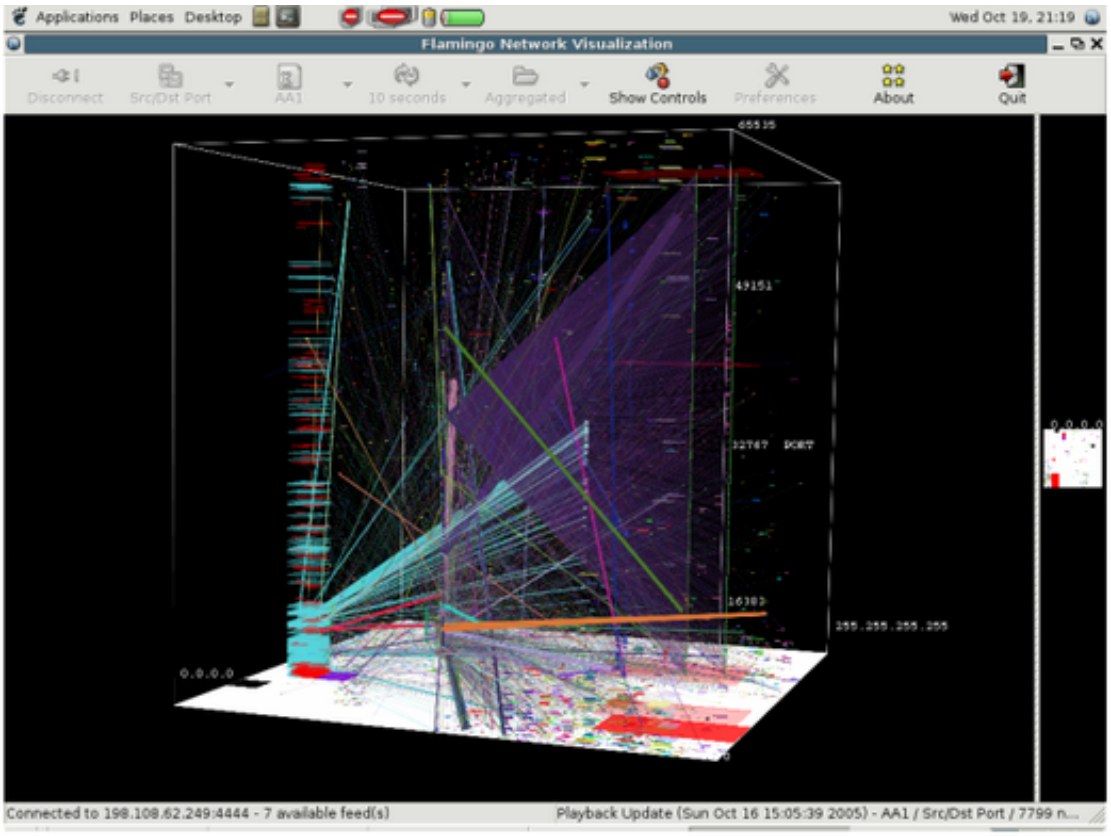


Figure 6.6: Flamingo showing a traffic anomaly amongst other live network traffics in the shape of a triangular fan [34].

The tools described in Section 6.2-6.3 have been also compared with this thesis prototype which is described in Section 6.5.

6.4 Malicious HTTP redirection detection tools based on user browsing activity

This section presents a brief introduction about the different detection systems that have been developed in order to track user browsing activities leading to malvertising. In almost all the detection systems, web crawling has been used to identify, detect and understand the cause of malicious activities which is different from the approach taken in this thesis. A comparison has been made in Section 6.5 which can be referred to for full information.

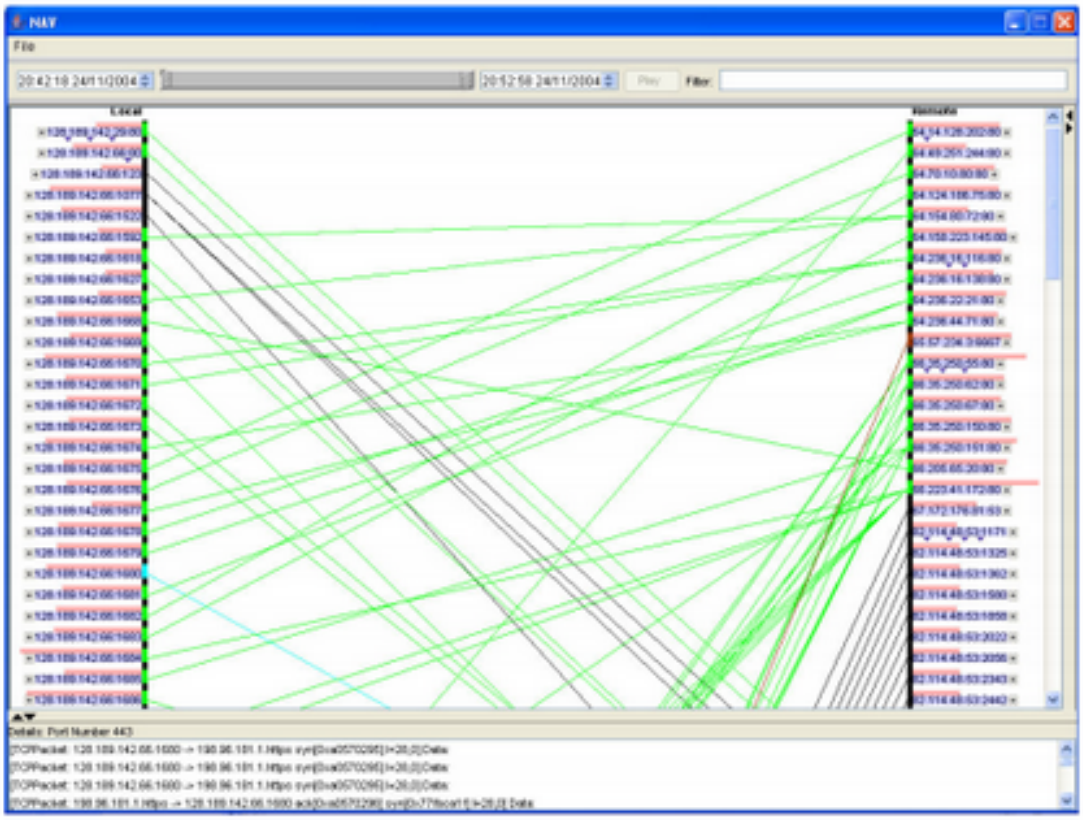


Figure 6.7: NAV showing the IP wall view. The left list represents local addresses and the right represents remote addresses. Connections are represented by coloured lines. Widgets are provided to allow ranges to be collapsed and to disable visualization of connections associated with a given element or range [37].

6.4.1 SURF

Lu et al. [38] present a detection system developed by them for detecting search poisoning cases. Search poisoning is a method of attack in which malicious websites are created by cybercriminals who use search engine optimization tactics to make those malicious websites to show up prominently in search results. The detection system, Search User Redirection Finder (SURF), runs as a browser component. The function of this system is to extract a number of robust detection features whose attribute can be said to be difficult to evade from search-then-visit browsing sessions. Consequently, it is able to achieve the primary function of accurately classifying malicious search user redirections resulting from user clicking on search results which are poisoned or malicious. Search poisoning techniques target any search term that can maximize the number of incoming search users (e.g., popular keywords) [38]. The authors have mainly focused on detecting malicious search

user redirections, which means to be any redirection that starts from a search landing page assisted by search poisoning and ends at a malicious page (i.e., the final target of the redirection). Though SURF deals with monitoring users click on a page, redirections till landing the destined page based on the query submitted by the user, the prototype described in this thesis rather deals on user visiting any website and ultimately understanding the flow of requests.

6.4.2 MadTracer

To detect malvertising activities and understand its pattern, Li et al. describe about a system developed by them called MadTracer, which automatically generates detection rules and utilizes them to inspect advertisement delivery processes and detect malvertising activities [39]. MadTracer detects malicious advertisements by identifying redirections using sequence of requests initiated by advertisements. A path is built on the traces of requests by building a crawler as a Firefox add-on. This crawler is used for visiting pages every three days and tries to obtain different advertisements. All the requests, responses, browser events are recorded and an advertisement redirection chain is constructed which is further used for identifying malicious URLs. The technique of detection also differs from the prototype of this thesis as the former techniques focus more on particular kind of redirection like in this case it is advertisements and use web crawlers. It also does not focus much on the visual display of the network events which is unlike in this thesis.

6.4.3 Browsing Activity Trees

Both of the tools, SURF and MadTracer, focus on particular kind of redirections i.e., either by search engine or redirections, however, Mekky et al. [3] present a methodology for detecting HTTP redirections using trees. Their method is based on building per-user chains from passively collected traffic and extracting statistical features by using machine learning-based technique for detecting malicious HTTP redirection. The URLs in the tree are marked as malicious if they are flagged by an Intrusion Detection System (IDS) and therefore, the browsing activity tree augments IDS rather than replacing it. This methodology also does not incorporate any visual effects which on the other hand is the main criteria in our thesis.

6.4.4 HViz

Gugelmann et al. [40] take a different approach than SURF, MadTracer or Browsing activity trees by visualizing the timeline of HTTP activities of workstation. The prototype developed by the authors is called HViz which facilitates incident

investigation, aggregation and correlating HTTP events between workstations in order to reduce the number of events that are exposed to an investigator [40]. This tool supports in investigating why a particular service receives HTTP requests. The HTTP events are grouped based on domain and represented in the form of coloured nodes with directed edges. This tool is different than the prototype developed in this thesis as HViz does not deal with extracting URLs from the body of HTTP response and also it mainly focuses on HTTP events taking place in a workstation which is not the case in this thesis.

6.5 A brief comparison with the prototype of this thesis

All the tools listed in Section 6.1-6.3 are similar in one way or the other from the fact that they deal with network traffic visualization or help in monitoring the network activity. However, they differ from the prototype that is being described in this thesis as none of them help to visualize the HTTP traffic flow from packet data in detail. The prototype developed in this thesis will give the advantage to an analyst or a researcher to view the HTTP traffic flow over time w.r.t., the HTTP *Referer* header giving him/her the opportunity to easily visualize the pattern of the flow thereby, helping in the detection of malicious URLs.

Though the detecting systems described in Section 6.4 deal with visualizing HTTP flows and identifying malicious URLs but the approach taken by the authors is different as they consider criteria such as specific keyword search, redirections or only URLs taken from the headers of HTTP request/ response pairs for building the systems. This is completely different than the approach taken in this thesis as it deals with extraction of both the head and body information from HTTP flows and using it to build an interactive web-based interface for viewing the HTTP flows in the form of a graph.

7

Discussion

The purpose of this thesis was to design a prototype which can be used to easily and effectively identify and visualize HTTP flows extracted from pcap files. The thesis also gives a literature survey of visualization tools in general and a comparison with the working of the prototype implemented in this thesis which adds knowledge to the research area of visualization. The results of the validation also show that the prototype is user-friendly due to the interactive web interface and also the prototype is easy to learn and run.

If we consider the choice of the two patterns of the graphs, the reason being choosing the one that fits best based on the required result. The scatter plot graph pattern was chosen as the ultimate goal was to display the traffic flow over time which can be achieved in a graph having x and y axis and therefore, scatter plot was the best fit for this thesis. Considering the choice of force directed graph pattern, as there was no such requirement of displaying over time, this was the best fit as the graph can be dragged and dropped in the viewing space. Therefore, it can be concluded that the chosen system design and implementation approach was successful in achieving the desired result of visual display of the HTTP flows.

7.1 Ethics

Ethical decisions and behavior from any person in society, but even more so from a computer and security professional is very important. The European Union finds it important to build trust in the digital society of the future [45] and the work in this thesis is one step closer to such a future. The decisions from an ethical perspective should take care of the welfare of society without compromising on

their needs which in other words is known as sustainable development. Hence, as technologists there are common code of ethics which should be followed and below is the list of a few of them taken from " A Guide to Forensic Testimony" [44] which has been followed in the work in this thesis.

1. Technology is important to modern society.
2. Technologists must take care not to endanger the life, health, safety, and welfare of the public.
3. Technologists must maintain and update their technical skills.
4. Technologists should be honest and forthright in their dealings with others.
5. Technologists should give proper credit to others for their work and honor property rights, including copyrights and intellectual property.
6. Technologists should help the public understand technology and support the professional development of peers.

As this thesis places an emphasis on effective visualization of HTTP flows for identifying malicious URLs, stopping the attacker from affecting the user would be an appropriate ethical response. While working on packet data captured on client network in a company, it is ethical not to disclose the clients name and network data to unauthorized personals.

7.2 Future Work

As a future work, there are several possibilities related to the subject of this thesis. The prototype can be extended to work on live packet data in addition to pcap files. It can be also extended to extract URLs from obfuscated Javascript files hidden in the body of HTTP response. The scalability of the prototype can be improved for example by using graphic processing unit rather than CPU which will enhance the performance speed. Another area of extension can be the inclusion of zooming and filtering options as additional features to the home page of the visualization.

8

Conclusion

HTTP is the underlying protocol of most of the data communication happening in the Internet. Browsers being the predominant source of data traffic in networks and HTTP implemented in browsers make the HTTP vulnerable to find loopholes for an attacker to implant malicious software. Therefore, attackers are using the web as the platform for employing HTTP redirections and hence, detecting malicious URLs becomes difficult for an analyst when the requests are large in number. In this thesis, I have discussed my contribution to provide an HTTP traffic flow visualization prototype for identifying malicious HTTP redirections. To achieve this goal, a number of open-source tools were integrated into my prototype to make it reliable and stable. Packets are captured and stored in a pcap file before being processed to extract information about HTTP request/response pairs. The retrieved data was then used to construct two types of dynamic graphs to be used by the security analyst and they were presented in a web-based interface. There are two kinds of graphs for visualizing: the first one being a scatter plot graph showing relationship between URLs extracted from the head of HTTP request/response pair and the second being a force-directed graph showing relationship between URLs extracted from the head and body of HTTP response. The prototype provides a bird's-eye view of HTTP traffic flows as well as a graphical view on its web-based interface. By validating the prototype under varying sized pcap files, it was found that the prototype is scalable at least up to 80 MB pcap files with 281 request/response pairs. It was also found that the usability of the tool is satisfactory due to the interactive web-based interface. Installing and using the prototype also does not consume much memory. In addition, the prototype is able to clearly show identification of malicious URLs and also the parent URL causing the redirection. The graph plotted with pcap size and number of HTTP request/response pairs showed logarithmic growth in memory adding another observation to the part of

validation. Experts using the prototype felt that it addressed many of the key aspects of the problem of identifying malicious URLs with precision and that the visualization is expandable to provide a wider range of potentiality in the future. The prototype also showed that it is much easier and efficient for a security analyst to identify malicious URLs and visualize the traffic flows which is otherwise difficult while scanning through hundreds of lines in textual format.

Bibliography

[1] A.D. Rubin and D.E. Geer. *A survey of Web security*, Computer, vol. 31, no. 9, Sept 1998, pp. 34,41.

[2] Simson Garfinkel and Gene Spafford. *Web security, Privacy & Commerce, 2nd Edition*, Sebastpool, CaA, 2001, p. 699-707.

[3] H. Mekky, R. Torres, Zhi-Li Zhang, S. Saha and A. Nucci. *Detecting Malicious HTTP Redirections using trees of user browsing activity*, INFOCOM, IEEE Proceedings, April 27- May 2, 2014, pp. 1159,1167.

[4] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. *All your iFRAMEs Point to Us*, in USENIX Security, 2008.

[5] VizSEC Workshop, `http://www.vizsec.org/`

[6] Russ McRee. *Security Visualization: What you don't see can hurt you*, ISSA Journal, June 2008.

[7] Edward R. Tufte. *the Visual Display of Quantitative Information*, 1986.

[8] Colin Ware. *Book review: Information visualization: Perception for design*, 2000.

[9] OSI Model, `http://en.wikipedia.org/wiki/OSI_model`. [Accessed February, 2015]

[10] HTTP WIKI, `http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`

[11] HTTP, `http://www.tutorialspoint.com/http/http_overview.htm`

[12] HTTP RFC, `http://tools.ietf.org/html/rfc2616`

[13] J. Chang, K.K. Venkatasubramanian, A.G. West and I. Lee. *Analyzing and defending against web-based malware*, ACM Computing Surveys, vol. 45, no. 4, 2013.

[14] Niels Provos and Dean Mcnamee and Panayiotis Mavrommatis and Ke Wang and Nagendra Modadugu and Google Inc. *The ghost in the browser: Analysis of web-based malware*, In Usenix Hotbots, 2007.

[15] The New York Times, September 14, 2009, `http://www.nytimes.com/2009/09/15/technology/internet/15adco.html?_r=2&` [Accessed June, 2015]

[16] Computer Fraud & Security, ISSN 1361-3723 April 2011, `www.computerfraudandsecurity.com` [Accessed January, 2015]

[17] Ben Shneiderman. *The eyes have it: A task by data type taxonomy for information visualizations*, In IEEE Symposium on Visual Languages, 1996, pp. 336–343.

[18] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind and Hartmut Ziegler. *Challenges in Visual Data Analysis*, In Proceedings of the Tenth International Conference on Information Visualization, 2006, pp. 9-16.

[19] Robert Spence and Mark Apperley. *Data base navigation: An office environment for the professional*, Behaviour & Information Technology, vol. 1, no. 1, 1982, pp. 43-54.

[20] Catherine Plaisant, Anne Rose, Brett Milash, Seth Widoff and Ben Shneiderman. *LifeLines: Visualizing personal histories*, Proc. ACM CHI96 Conference: Human Factors in Computing Systems, 1996, pp. 227, 518.

[21] T. Asahi, D. Turo and B. Shneiderman. *Using treemaps to visualize the analytic hierarchy process*, Information Systems Research vol. 6, no. 4 December 1995, pp. 357-375.

[22] Kim M. Fairchild, Steven E. Poltrock and George W. Furnas. *SemNet: Three-dimensional representations of large knowledge bases*, In Guindon, Raymonde (Editor), Cognitive Science and its Applications for Human-Computer Interaction, Lawrence Erlbaum, Hillsdale, NJ, 1988, pp. 201-233.

[23] Keith Andrew. *Visualising cyberspace: Information visualisation in the Harmony internet browser*, Proc. IEEE Information Visualization, 1995, pp. 97-104.

[24] R. J. Hendley, N. S. Drew, A. S. Wood. *Narcissus: Visualizing information*, Proc. IEEE Information Visualization '95, 1995, pp. 90-96.

[25] M. Bostock, `https://github.com/mbostock/d3/wiki/Force-Layout`. [Accessed May, 2015]

[26] RUMINT, `http://www.rumint.org/software/rumint/rumint_overview.pdf`

[27] Sam Abott-McCune, A.J. Newtson, Robert Ross, Ralph Ware and Gregory Conti. *Free visualization tools for security analysis and network monitoring*, `http://www.rumint.org/gregconti/publications/insecure_conti.pdf`

[28] Ryan Blue and Cody Dunne and Adam Fuchs and Kyle King and Aaron Schulman. *Visualizing Real-Time Network Resource Usage*, In Proc. Visualization for Computer Security: 5th Int. Workshop, Vizsec 2008, pp. 119-135.

[29] T.M. J. Fruchterman and E. M. Reingold. *Graph drawing by force-directed placement*, Software - Practice and Experience, vol. 21, no. 11, 1991, pp. 1129–1164.

[30] B. Shneiderman. *Tree visualization with tree-maps: 2-d space-filling approach*, ACM Trans. Graph., vol. 11, no. 1, 1992, pp. 92–99.

[31] TNV, `http://tnv.sourceforge.net/index.php`

[32] AFTERGLOW, `http://afterglow.sourceforge.net/`

[33] ETHERAPE, `http://etherape.sourceforge.net/introduction.html`

[34] J. Oberheide, M. Goff, M. Karir. *"Flamingo: Visualizing Internet Traffic"*, Network Operations and Management Symposium, 2006. NOMS 2006, 10th IEEE/IFIP, vol. 3, no. 7, April 2006, pp. 150,161.

[35] X. Yin, W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju. *Visflowconnect: Netflow visualizations of link relationships for security situational awareness*, Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security(VizSEC), October 2004.

[36] K. Lakkaraju, W. Yurcik, A. Lee, R Bearavolu, Y. Li, and X. Yin. *Nvisionip: Netflow visualizations of system state for security situational awareness*, Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security(VizSEC), October 2004.

[37] M. Allen, P. McLachlan. *"NAV Network Analysis Visualization"*, University of British Columbia, 29 May 2009, `http://www.cs.ubc.ca/~spark343/NAV.pdf`

[38] L. Lu, R. Perdisci, and W. Lee. *Surf: detecting and measuring search poisoning*, CCS, 2011.

[39] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang. *Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising*, CCS, 2012.

[40] D. Gugelmann, F. Gasse, B. Ager and V. Lenders. *Hviz: HTTP(S) traffic aggregation and visualization for network forensics*, Digital Investigation, vol. 12, no. S1, 2015.

[41] Wireshark tool. `https://www.wireshark.org/` [Accessed February, 2015]

[42] Omri Herscovici. *CapTipper- Malicious HTTP traffic explorer tool*, `http://www.omriher.com/2015/01/captipper-malicious-http-traffic.html`. [Accessed January 17, 2015]

[43] M. Bostock, V. Ogievetsky and H. Jeffrey. *D3: Data-Driven Documents*, IEEE transactions on Visualization and Computer Graphics, vol. 17, no. 12, December 2011.

[44] F. C .Smith, R. G. Bace. *A Guide to Forensic Testimony: The Art and Practice of Presenting Testimony as an Expert Technical Witness*, New York Addison-Wesley, pp. 177-179, 2002.

[45] Digital Agenda For Europe, `https://ec.europa.eu/digital-agenda/en/trust-and-security`. [Accessed May 31, 2015]