

2008

IseHarvest: TCP packet data re-assembler framework for network traffic content

Stephen Michael Eilers
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/rtd>

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Eilers, Stephen Michael, "IseHarvest: TCP packet data re-assembler framework for network traffic content" (2008). *Retrospective Theses and Dissertations*. 14940.
<https://lib.dr.iastate.edu/rtd/14940>

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

IseHarvest: TCP packet data re-assembler framework for network traffic content

by

Stephen Michael Eilers

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Co-majors: Information Assurance; Computer Engineering

Program of Study Committee:
Doug Jacobson, Major Professor
Thomas E. Daniels
Barbara Licklider

Iowa State University

Ames, Iowa

2008

Copyright © Stephen Michael Eilers, 2008. All rights reserved.

UMI Number: 1453074

UMI[®]

UMI Microform 1453074

Copyright 2008 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

TABLE OF CONTENTS

LIST OF FIGURES	iii
ABSTRACT	iv
CHAPTER 1. INTRODUCTION	1
1.1 IseHarvest Overview	2
CHAPTER 2. RELATED TECHNOLOGY	4
2.1 Tcpdump	4
2.2 Wireshark	6
2.3 Iris [®]	10
2.4 Tcpflow	12
2.5 Result	14
CHAPTER 3. IMPLEMENTATION	15
3.1 Tcpflow	15
3.2 IseHarvest Extension	18
3.2.1 <i>Packet Filtering</i>	18
3.2.2 <i>Data Extraction and Reconstruction</i>	22
3.2.3 <i>Re-linking</i>	25
3.2.4 <i>Framework</i>	29
3.2.5 <i>Compiling</i>	31
3.2.6 <i>Operating</i>	31
CHAPTER 4. TESTING	33
CHAPTER 5. CONCLUSION	38
5.1 Limitations	38
5.2 Future Work	40
REFERENCES	42
ACKNOWLEDGEMENTS	43

LIST OF FIGURES

Figure 1: IseHarvest Flow	3
Figure 2: Tcpdump Output at Iowa State Homepage [4]	5
Figure 3: Wireshark Capture of ISU Homepage	7
Figure 4: Wireshark TCP Stream of ISU Homepage	9
Figure 5: Iris [®] Capture & Visual of ISU Homepage	11
Figure 6: Tcpflow Output of ISU Homepage	13
Figure 7: GET Request and Document [12].....	20
Figure 8: ISU Homepage with web browser.....	26
Figure 9: ISU Homepage with IseHarvest without re-linking	27
Figure 10: ISU Homepage with IseHarvest with re-linking	29
Figure 11: Wireshark Capturing ISU Homepage for testing.....	35
Figure 12: ISU Homepage Example Directory Output with IseHarvest.....	35
Figure 13: Slashdot with IseHarvest [14].....	36

ABSTRACT

IseHarvest is a network analysis tool designed as a framework for extracting data from TCP streams. Once extracted, it reconstructs the data into individual files and documents from captured network traffic, and saving the data locally. Besides being a framework, IseHarvest passively extracts HTTP data from captured traffic and reconstructs files and web directory layouts in such a way to enable web content monitoring of network resources. By re-linking HTML and CSS files, it provides an after the fact visual of how web sites were viewed over a network and a look at how corresponding web servers are laid out by looking at only captured files.

CHAPTER 1. INTRODUCTION

In today's Internet-driven world, there are large quantities of data being sent over computer networks, both on the large scale (Internet or World Wide Web) and smaller scale (business, school or intranets). Network analysis is becoming a larger part of daily life for these networks and network administration. A common method of analyzing a network is to capture all traffic passing through the network over a period of time, usually performed with an application similar to Wireshark, described in Section 2.2. That captured traffic is then analyzed, using applications such as Wireshark, to retrieve information that varies from what computers were involved in a connection, what type of protocol was used in the traffic, and even to determine whether the information was encrypted or not. This information is very useful in helping us prove that a specified computer accessed a web site, when it accessed a site, what protocols were used, and even what programs were used to access that web site. However, it can be difficult for the common computer user and ad hoc administrator to make sense of this information, due to the technical format, and determine what content was actually viewed. There may be times where it can be desirable to determine what content was viewed by a specific computer to ensure proper use of a business network. Therefore, it is necessary to create an application which will allow an administrator to visually see the websites and images which were viewed over the monitored network.

IseHarvest is intended to improve network analysis to determine what content was passed over the internet connection. It attempts to reconstruct the content in such a manner that it can be easily viewed, whether a user is technically trained or not. A

specific example would be when a business is paying for its employees to attend a seminar or conference. The business is spending money on this opportunity for employees, and if employees are surfing the Internet for unrelated material, they might not be gaining any benefits from this event. If a business were able to find out if employees were wasting time using business resources, that could change how future events are planned and what resources are available.

1.1 IseHarvest Overview

IseHarvest was thought of and created from the need to make network analysis easier by passively reconstructing standard web traffic into corresponding data files. IseHarvest is also intended to provide a framework for implementation and extension of additional network protocols. Its goals include: 1) read captured Transmission Control Protocol (TCP) traffic, 2) extract the HyperText Transfer Protocol (HTTP) data, 3) reconstruct that data into viable documents, videos and web pages, 4) re-link HTML and CSS pages to allow local viewing without directly contacting a website, and 5) provide a framework for future modifications and extensions to be added to IseHarvest. It reads captured network traffic data and reconstructs it directly into the images, documents and graphical web pages that were viewed over the network and stores them in a reconstruction of server directories. This enables users of IseHarvest to easily understand and comprehend what network users viewed over the Internet, what documents were sent or received, and be able to see graphically what web pages have been viewed.

In order to provide a visual aid in understanding IseHarvest's flow, the following diagram represents the process that data follows when run through IseHarvest. Upon

receiving packets from Tcpflow implementation, if identified as HTTP, they are processed and the data is extracted. When completed files are received, if determined to be HTML or CSS files then the data is re-linked to allow easy viewing locally. Finally, all files assembled, are written to the hard drive to be viewed later.

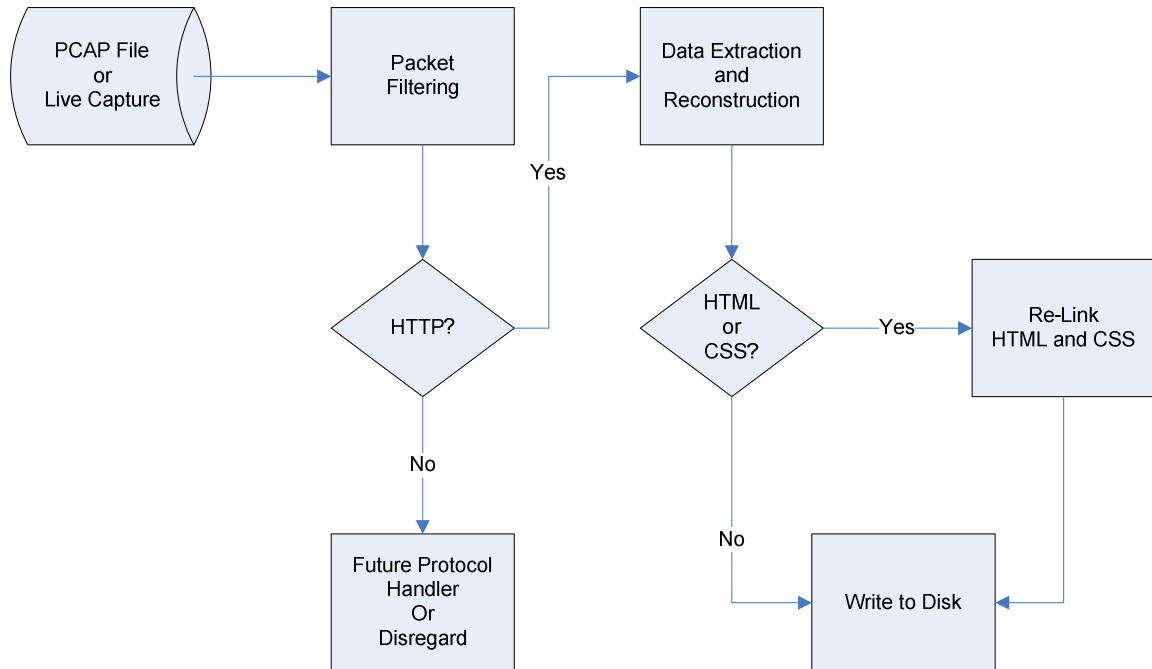


Figure 1: IseHarvest Flow

IseHarvest concurrently has other uses that it may prove to be beneficial for, such as research environments similar to the Internet-Scale Event and Attack Generation Environment (ISEAGE) project at Iowa State University. The ISEAGE simulation environment creates large amounts of web traffic captured during events that could be potentially extracted using IseHarvest. When used in conjunction with tools such as Wireshark, useful information could be deduced from what type of attacks were being performed and how websites were changed or modified as a result. Officials at businesses, schools or universities could potentially use this software, depending on the

situation, to verify whether employees or students are using company computers to view class related material or unrelated material on school or business-controlled network resources. There are numerous possibilities for IseHarvest to be used in web traffic reconstruction. In the following section, the technologies examined when researching IseHarvest will be mentioned.

CHAPTER 2. RELATED TECHNOLOGY

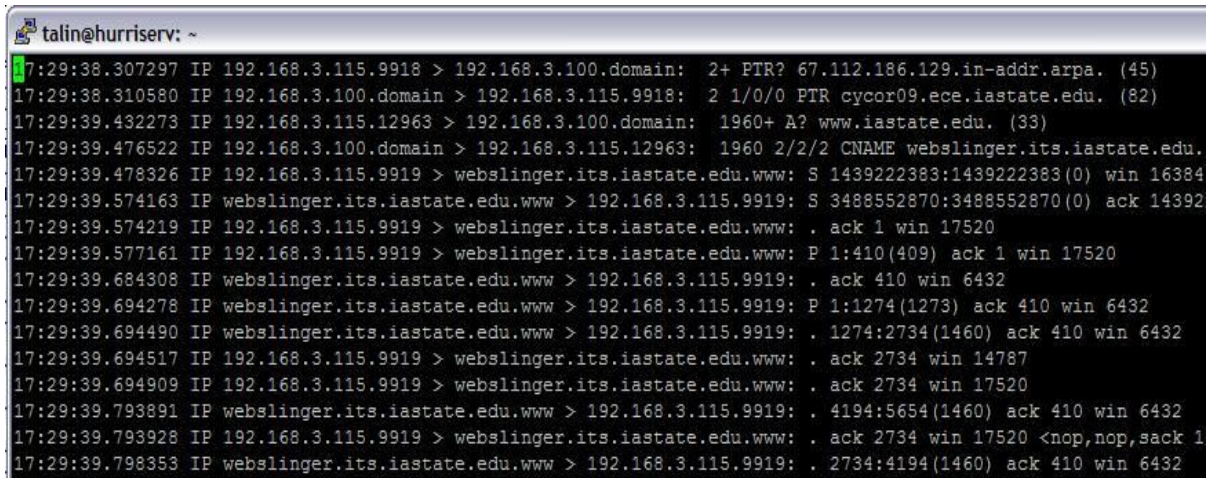
This chapter presents a few technologies that provide functionalities along similar lines to IseHarvest. Concurrently discussed are some of the limitations and reasons why these applications are inadequate for the purpose of providing a more visual-oriented network analysis option.

2.1 Tcpdump

Tcpdump is an open source Unix/Linux compatible, command line application designed to provide a dump or capture of traffic going over the network. Simply put, Tcpdump prints out a description of the contents of data packets captured on a network interface [1]. This dump can be supplemented with specific options to look for certain types of traffic or can be used to dump all traffic that is on the network at the time of its operation. In addition to capturing and filtering received packets, Tcpdump provides a number of functionalities: 1) read and write captured traffic to data files in Packet Capture (PCAP) [2] format, 2) filter packets based on specified parameters, and 3) print limited or full data from each packet based on provided parameters. Tcpdump provides basic options for network analysis while outputting its data to the screen or to data files.

This can limit the ability of Tcpcmdump to provide detailed analysis of network traffic alone and suggests that Tcpcmdump is most commonly used in conjunction with other traffic analysis tools, such as Wireshark discussed in the next section, in order to provide effective analysis.

As mentioned, Tcpcmdump provides a very basic view of the network traffic. It prints a summary of the packets that were captured on the wire, without necessarily storing the data that those packets contained, by default [3]. There are options to provide more detailed information, but the following is of the default screen output:



```

talin@hurriserv: ~
17:29:38.307297 IP 192.168.3.115.9918 > 192.168.3.100.domain: 2+ PTR? 67.112.186.129.in-addr.arpa. (45)
17:29:38.310580 IP 192.168.3.100.domain > 192.168.3.115.9918: 2 1/0/0 PTR cycor09.ece.iastate.edu. (82)
17:29:39.432273 IP 192.168.3.115.12963 > 192.168.3.100.domain: 1960+ A? www.iastate.edu. (33)
17:29:39.476522 IP 192.168.3.100.domain > 192.168.3.115.12963: 1960 2/2/2 CNAME webslinger.its.iastate.edu.
17:29:39.478326 IP 192.168.3.115.9919 > webslinger.its.iastate.edu.www: S 1439222383:1439222383(0) win 16384
17:29:39.574163 IP webslinger.its.iastate.edu.www > 192.168.3.115.9919: S 3488552870:3488552870(0) ack 14392
17:29:39.574219 IP 192.168.3.115.9919 > webslinger.its.iastate.edu.www: . ack 1 win 17520
17:29:39.577161 IP 192.168.3.115.9919 > webslinger.its.iastate.edu.www: P 1:410(409) ack 1 win 17520
17:29:39.684308 IP webslinger.its.iastate.edu.www > 192.168.3.115.9919: . ack 410 win 6432
17:29:39.694278 IP webslinger.its.iastate.edu.www > 192.168.3.115.9919: P 1:1274(1273) ack 410 win 6432
17:29:39.694490 IP webslinger.its.iastate.edu.www > 192.168.3.115.9919: . 1274:2734(1460) ack 410 win 6432
17:29:39.694517 IP 192.168.3.115.9919 > webslinger.its.iastate.edu.www: . ack 2734 win 14787
17:29:39.694909 IP 192.168.3.115.9919 > webslinger.its.iastate.edu.www: . ack 2734 win 17520
17:29:39.793891 IP webslinger.its.iastate.edu.www > 192.168.3.115.9919: . 4194:5654(1460) ack 410 win 6432
17:29:39.793928 IP 192.168.3.115.9919 > webslinger.its.iastate.edu.www: . ack 2734 win 17520 <nop,nop,sack 1
17:29:39.798353 IP webslinger.its.iastate.edu.www > 192.168.3.115.9919: . 2734:4194(1460) ack 410 win 6432

```

Figure 2: Tcpcmdump Output at Iowa State Homepage [4]

The above data includes information such as IP addresses, Domain Name System (DNS) addresses, port numbers, etc... This information was captured while a web browser viewed the Iowa State University homepage [4] and is used in reference to most images unless noted otherwise. The information is useful because it can help potentially determine what computers were involved in connections, general protocols (IP) and the time of packet capture. However, it does not provide much useful information about what a user is specifically looking at and what that information looks like. If this was

from a past traffic capture, the listed host names or IPs could have changed their viewable data, making it extremely difficult, if not impossible, to actually determine the content of those sites. When Tcpdump is used to directly output packets to the hard drive, there is much more detailed information provided in the file due to following the PCAP format; otherwise Tcpdump returns the packet headers and data.

As it can be seen from the above image, Tcpdump provides information that is more directed toward the technically trained user. It does not provide any method for a user to see a complete data file, whether text based or visually, in order to determine what the data content is. This can limit the usefulness when the intent of network analysis is to determine what the actual content of packets included.

Additionally, Tcpdump was looked at to potentially be used as a base application for building IseHarvest upon. It was chosen against because unlike Tcpflow discussed in section 2.3, Tcpdump did not perform much in the way of TCP stream assembly.

2.2 Wireshark

On Windows or UNIX related systems, Wireshark (or Ethereal if referring to older versions) is an open source software package designed to provide network protocol analyzing utilities to its users. Wireshark is a graphical user interface (GUI) network protocol analyzer that provides a method to interactively browse packet data from a live network or from a previously saved capture file. To reach this goal Wireshark provides a number of functionalities: 1) filter captured data based on specified input, 2) organize packets together into complete TCP streams to allow easier analysis of specific connections, 3) read and write captured packets into a variety of formats for compatibility

with other applications and 4) provide statistics to allow correlation between packets and provide results from analyzing network protocols. These functionalities are extremely useful, but can still potentially create difficulties in identifying the actual content being viewed.

Wireshark allows a user to interactively browse through captured packet data from a live network or from previously captured traffic data for the functionalities described above [5]. An example of this is as follows:

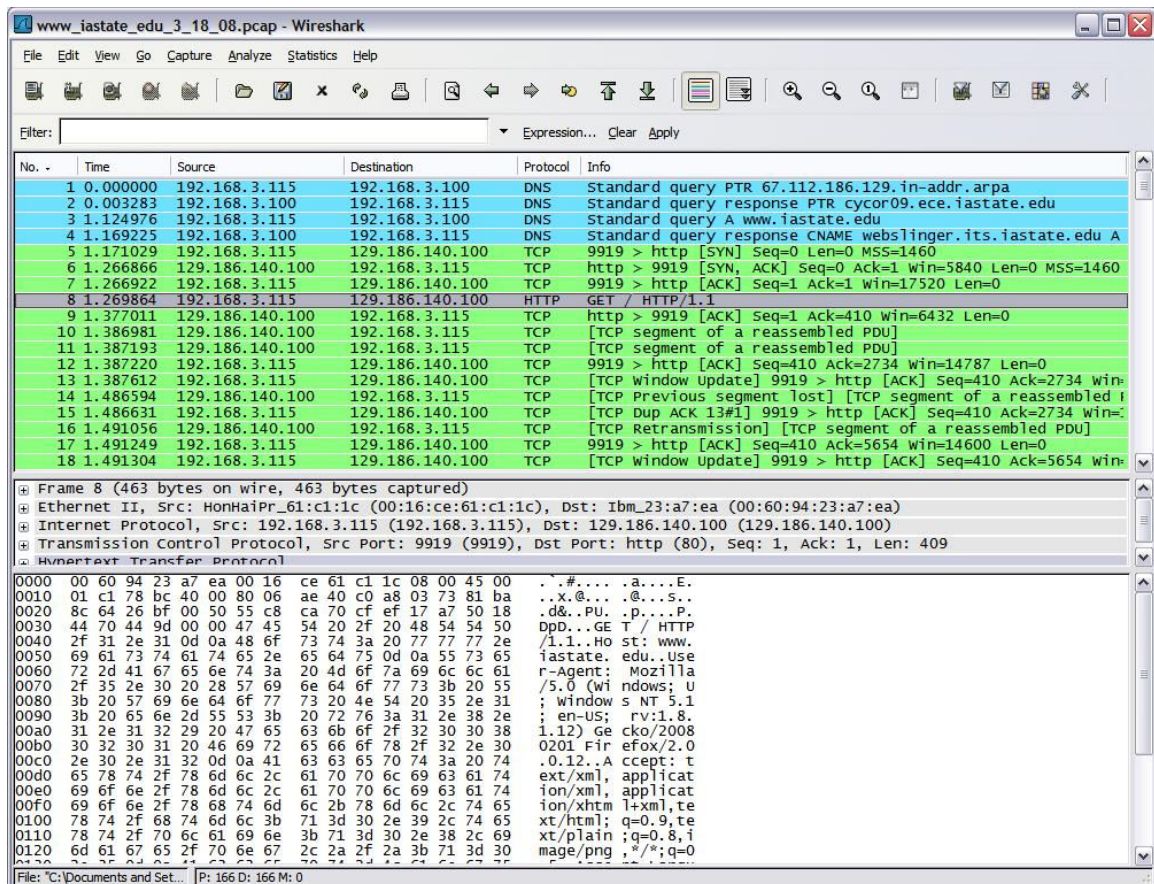


Figure 3: Wireshark Capture of ISU Homepage

Shown above is a capture from viewing Iowa State University's home page. As shown, Wireshark is broken into three major abilities: 1) browse through individual packets

(green and blue colored), 2) break apart and list header information specifically (grey) and 3) show the direct data content of each packet.

Wireshark is extremely useful in technically analyzing data packets. It displays the data effectively and uniformly in byte format, displaying and identifying different byte fields in the data packets. However if a user is not used to examining how packet headers worked, formats of packets, or even the hexadecimal numbering systems used here, then this data may appear as hard to understand gibberish.

Wireshark does begin to approach the purpose and functionality intended for this thesis project. It allows a user to assemble all data into continuous TCP connections or streams. A TCP stream in the context of this paper is a connection between two computers where multiple transmissions take place. These streams may send one file or many files between two computers maintaining the TCP stream. To construct the TCP stream, the user picks a packet and enables the “Follow TCP Stream” function. By following the IP addresses and ports used for each connection, Wireshark identifies all packets that match the parameters of the selected packet to assemble related data packets together and assembles them into the following format of figure 4. The format of the TCP stream can be difficult to follow for new users. It includes the initial HTTP packets that identify each file and follows that with a textual representation of assembled data packets in the TCP stream. For a technically oriented person it provides a lot of useful information, for example: operating systems, web browser, content (HTML or JPEG), language and others. Unless the user possesses a rudimentary understanding of HTTP packet headers, HTML language, CSS language and can understand what an image looks like in byte format, however, it provides little information about what the web page

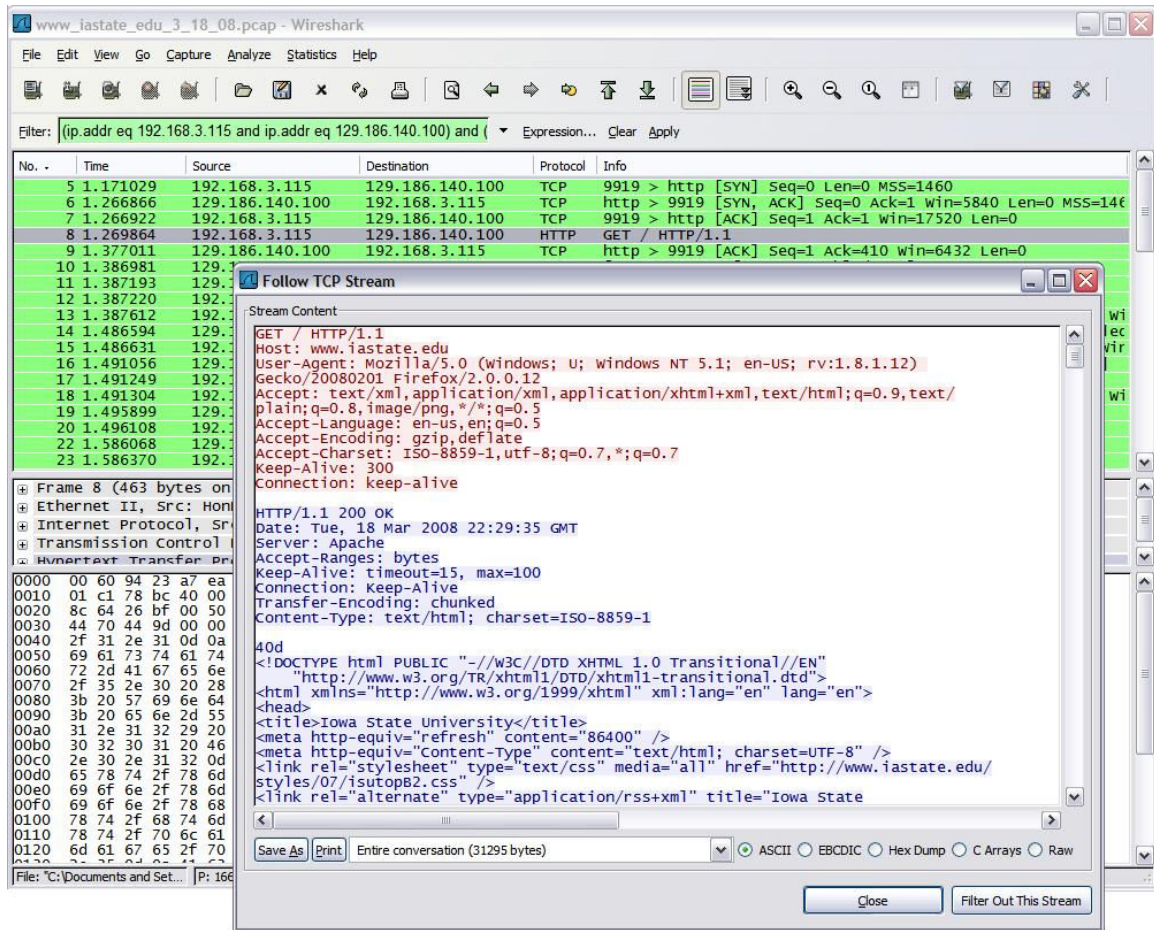


Figure 4: Wireshark TCP Stream of ISU Homepage

viewed actually looks like. There are many other tools that also perform this same functionality, including Capsa [6] and Javvin [7] products, but do not continue on to the next step of extracting the data into separate files. To understand the formats of these data segments, a user must perform a lot of copying and pasting of data to get it into formats that can be understood. This is obviously not an easy or efficient method to view images and other files.

Wireshark provides a variety of benefits, but still leaves much to be desired towards the goal of being able to see what another person saw over the network and being able to read documents that are “right clicked” and “saved as” from online. It did

however provide a useful tool in testing and implementing IseHarvest.

Wireshark was not chosen as a base to implement IseHarvest in for two reasons. The first reason is that IseHarvest is intended to reassemble data packets into full data files. While this could be done with Wireshark, the graphical interface did not bring any real advantage to IseHarvest since the majority of viewing would be operated through a files system explorer or web browser. The second reason is that simplifying the application would allow IseHarvest to run more quickly on stored capture files. Wireshark is a fairly large and complicated application to have running in the background of a computer. Therefore Wireshark was not chosen to be used as a base for IseHarvest.

2.3 Iris[®]

Iris[®] Network Traffic Analyzer is commercial network traffic monitor that provides limited visual web monitoring in addition to a number of functionalities similar to Wireshark. Like Wireshark, Iris[®] provides the ability to interactively browse through individual packets captured from a network and allows a user to search for specific packets, generate statistics, and identify usage of a targeted internet. In addition to these abilities Iris[®] also provides a limited ability for packet decoding. This means Iris[®] can organize captured packets and categorizes them by protocols such as HTTP, PPoE, and SNMP [8]. Once packets are categorized, Iris[®] provides a limited ability to visually see what the stream contained as seen in figure 5. As seen above, Iris[®] does provide limited visualization capabilities for web traffic. When a stream is selected and decoded, Iris[®] reads the TCP streams into the respective files similarly to Wireshark. Once files are reconstructed, Iris[®] then takes the next step and allows physical viewing and visualization

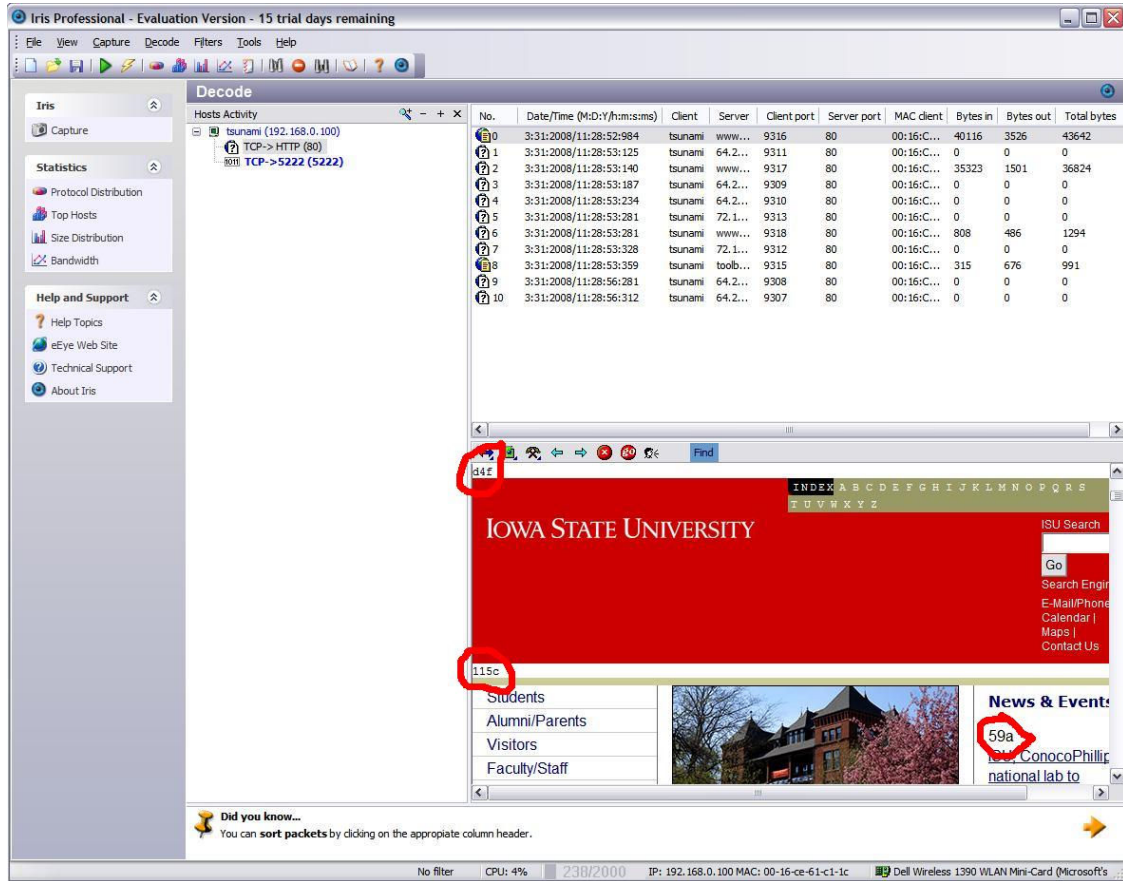


Figure 5: Iris® Capture & Visual of ISU Homepage

of the data. When Iris® visually shows an HTML page, it does not reference other captured files but references back to the hosting web server to re-download the web page. If Iris® has no internet connection, it does not show you the web site as it actually appears, but a text based representation from straight HTML. Additionally, in the above example you can see some out of place text highlighted in red circles. Those pieces of text are values that HTTP uses to segment data and are not handled by this application. In addition to this, from closer look at the web page shows that it is not formatted as the original Iowa State homepage is. IseHarvest attempts to handle many of these issues in implementation.

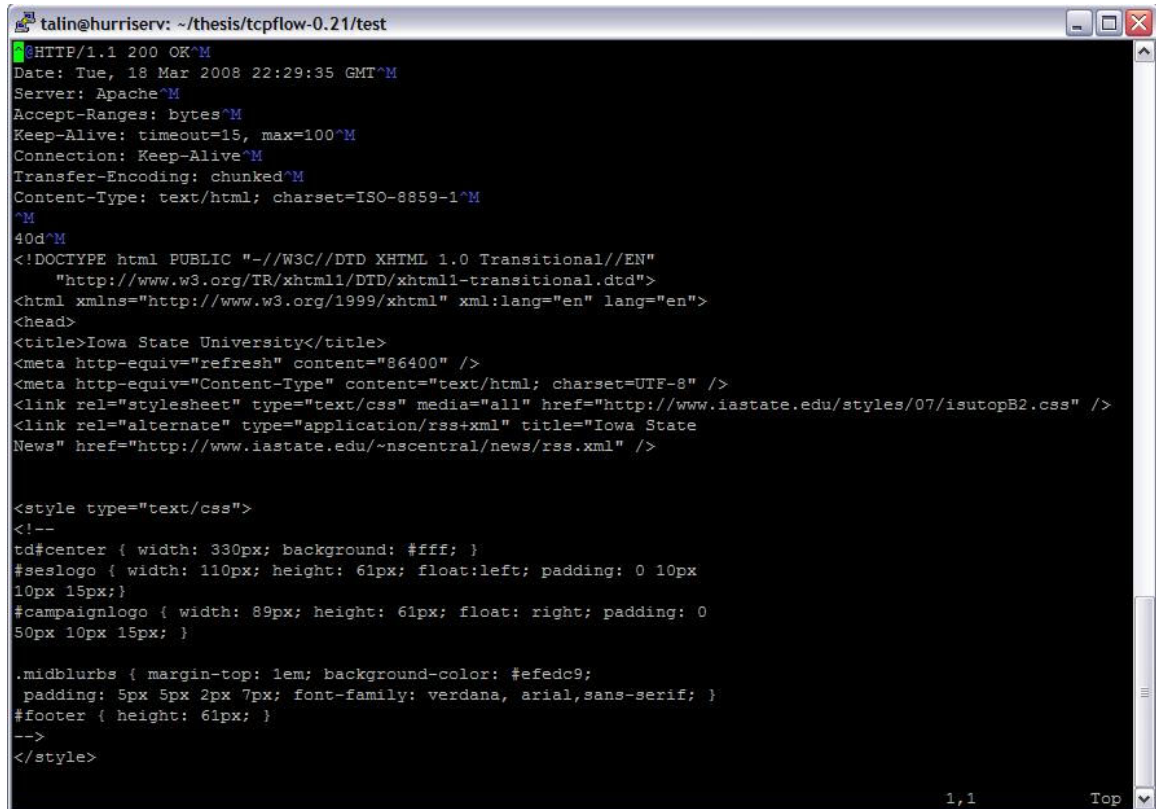
While Iris[®] allows a rudimentary visualization of HTTP and some other protocols; it does have disadvantages of not extracting the data and storing them locally on the computer for easy, separate analysis. Additionally, Iris[®] requires an internet connection to allow a user to visually see what other people saw over the network. Finally, Iris[®] is a commercial based application with a hefty price tag and therefore not easily expanded or modified similar to open source applications, such as Wireshark or IseHarvest. Therefore Iris[®] does not meet the goals of IseHarvest, and leaves an opening for IseHarvest to fulfill.

2.4 Tcpflow

Tcpflow is an open source application designed to provide a capture of traffic going over the viewable network. It differs from Tcpcap in that it reconstructs the actual data streams and stores each flow in a separate file for later analysis [3]. Tcpflow constructs data streams by extracting data from response packets from a given web server to the client. That data is then written to a file based on the IP addresses and ports that are specified in this given TCP data stream.

Tcpflow allows a user to alternatively look at a live capture of traffic or take previously captured data from other applications such as Wireshark or Tcpcap. The main requirement of captured data traffic is that it needs to be in the PCAP format; this allows Tcpflow to read and extract data from PCAP files and convert the packets into the corresponding TCP data streams. Building the captured traffic into data streams allows the data to be organized in a manner similar to the intended purpose of IseHarvest. However, Tcpflow does not take the next step of extracting the data into the specific files

contained in the data stream; by this I mean the images, HTML files, CSS files or even saved documents are combined together into one file instead of saved individually to the hard drive for easy access. The user can only look at this information in its byte or text format. The following is an example of Tcpflow output:



```

talin@hurriserv: ~/thesis/tcpflow-0.21/test
@HTTP/1.1 200 OK^M
Date: Tue, 18 Mar 2008 22:29:35 GMT^M
Server: Apache^M
Accept-Ranges: bytes^M
Keep-Alive: timeout=15, max=100^M
Connection: Keep-Alive^M
Transfer-Encoding: chunked^M
Content-Type: text/html; charset=ISO-8859-1^M
^M
40d^M
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Iowa State University</title>
<meta http-equiv="refresh" content="86400" />
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="stylesheet" type="text/css" media="all" href="http://www.iastate.edu/styles/07/isutopB2.css" />
<link rel="alternate" type="application/rss+xml" title="Iowa State
News" href="http://www.iastate.edu/~nscentral/news/rss.xml" />

<style type="text/css">
<!--
td#center { width: 330px; background: #fff; }
#seslogo { width: 110px; height: 61px; float:left; padding: 0 10px
10px 15px;}
#campaignlogo { width: 89px; height: 61px; float: right; padding: 0
50px 10px 15px; }

.midblurbs { margin-top: 1em; background-color: #efedc9;
padding: 5px 5px 2px 7px; font-family: verdana, arial,sans-serif; }
#footer { height: 61px; }
-->
</style>
1,1 Top

```

Figure 6: Tcpflow Output of ISU Homepage

In the previous data, Tcpflow's output consists of the HTTP header information containing generic information such as type (text/html), date, etc... before any data file. In addition to the header information it contains the re-assembled data, the HTML file, in a text format. This is difficult to view in a browser because it is combined with the header information and other data included in this TCP stream. This output is extremely useful for text files because it lets you view the content, but is almost useless for image or

video files.

As noted, Tcpflow provides a useful functionality, like Wireshark and Iris[®], where it assembles data packets back together into TCP data streams. Unfortunately it does not take the next step of assembling the data into actual data files that will let a web browser or an operating system easily display the data visually. As such, there is still a need for a data extraction tool that takes the process to the next step.

Tcpflow was chosen to use as a base for IseHarvest because it provided a number of useful features that made development of IseHarvest easier, such as packet capture and TCP session assembly. In addition to providing useful features to prevent re-inventing the wheel, Tcpflow is also a light weight, open source application that is adaptable for our purposes.

2.5 Result

There are a number of technologies that allowed a user to examine and analyze captured web traffic. Each of the technologies that I identified allows the user to examine captured traffic at a low, technically oriented level by directly examining bytes or text formats. Other technologies, such as Iris[®], provide a user with a limited visual representation of data, but not the opportunity to examine that data separately. This may present a problem for common users and new administrators if they have not been introduced to these options. As a result, IseHarvest moved forward with its purpose to develop a software application to make it easier for administrators and users to look at and visually analyze their captured network traffic.

CHAPTER 3. IMPLEMENTATION

This chapter focuses on the implementation details of various aspects of IseHarvest. It begins by looking at the implementation of Tcpflow and what it brought to IseHarvest. The next section is devoted to what the major aspects of IseHarvest were that needed to be developed and implemented in order to create a functioning application with the following subsections: packet filtering, data extraction, data reconstruction, re-linking, and compiling of the tool for the end user. The chapter ends with a brief explanation of operating the tool to perform network monitoring.

3.1 *Tcpflow*

As previously mentioned, Tcpflow is an application designed to provide network analysis by following specific TCP streams. Tcpflow operates by examining a provided PCAP file or live capture. Tcpflow was chosen to be the framework for IseHarvest because it already provides a number of features that will be required for IseHarvest to function. Tcpflow possesses the ability to: 1) use live or previously captured traffic, 2) organize captured TCP traffic into sessions that are organized by IP addresses and ports and 3) assemble corresponding data segments in respective sessions.

The first ability, retrieving packets from a live or saved capture, is implemented using the libpcap library [9]. Tcpflow initializes a handler depending on whether Tcpflow is using a specified input file or using an Ethernet device:

```
pd = pcap_open_offline(infile, error);
handler = find_handler(dlt, infile);
```

Or:

```
pd = pcap_open_live(device, SNAPLEN, !no_promisc, 1000, error);
handler = find_handler(dlt, device);
```

dlt represents the data link type that will be used and *infile* or *device* is the input file or network device used for retrieving packets. Once a handler is created, Tcpflow calls *pcap_loop* to initialize packet reading:

```
pcap_loop(pd, -1, handler, NULL);
```

pcap_loop is a function provided by the libpcap library, it uses pointer *pd* and *handler* to access either the capture file or network device to continually read packets from either location. Once a packet is read, the first step is to look at and strip the IP header from the packet to allow easier access to the next layer of data. Tcpflow is designed to only handle TCP traffic, so if the packet contains any other protocols, such as User Datagram Protocol (UDP) or Address Resolution Protocols (ARP), it currently disregards those packets since it is not interested in them. The next step for Tcpflow is to process the TCP headers. Once the TCP header is read from the data packet, a “flow” object is created to correspond to this packet. This object keeps track of source and destination IP addresses and corresponding port numbers for each address. The flow structure is provided as follows:

```
Typedef struct {
    u_int32_t src;
    u_int32_t dst;
    u_int16_t sport;
    u_int16_t dport;
} flow_t;
```

Once the flow is created, it is used to create a *flow_state_struct* that will keep track of additional information for individual TCP streams, such as what file is used to save the outputted stream. The *flow_state_struct* is an object created to identify necessary information to maintain and keep track of active streams. This object is then placed into

a list of all open connections that are currently active. Whenever a new stream is detected, a *flow* object is created as well as a corresponding *flow_state_struct*. Once created, they are placed into a list to enable Tcpflow to identify any data packets that will belong to the existing streams. The *flow_state_struct* keeps track of the flow, associated output file and various other pieces of information. The *flow_state_struct* object is structured as follows:

```
typedef struct flow_state_struct {
    struct flow_state_struct *next;
    flow_t flow;
    tcp_seq isn;
    FILE *fp;
    long pos;
    int flags;
    int last_access;
} flow_state_struct;
```

Now that a *flow_state_struct* has been created, all packets that match the *flow* designated by this state (meaning possess the same source IP address, destination IP address, source port and destination port) will be outputted to the file designated by their corresponding *flow_state_struct*. Additionally, since TCP packet sequence numbers are incremented according to the size of each packet in a specific TCP stream, they are used to chronologically output all data to the specified output file such that data will be placed in correct order when written to files along with its HTTP header information. Each output file is named and identified based on the *flow* object, meaning by IP addresses and ports. These output files are designated as follows:

```
<source_IP>.<source_port>-<destinatiaon_IP>.<destination_port>
192.168.1.1.54321-192.168.1.2.54322
```

When a TCP stream is active, it is common for multiple files to be transferred over the stream. Therefore once a stream is closed, there will often be multiple data files

stored in the specified output file. From this point, Tcpflow was modified to enable it to function in a manner that met the goals for IseHarvest (more on this below). IseHarvest extension replaced the current TCP stream reconstruction and rewrote it to reconstruct individual files instead multiple files together for a stream, with capacity for future protocols to be added.

3.2 IseHarvest Extension

As stated previously, IseHarvest is intended to be a software framework that provides users with a visual method of analyzing network traffic to determine web content has been transferred over the network. It is additionally implemented to be easy expanded to handle further protocols as necessary.

The following subsections discuss the components and modifications that were incorporated into Tcpflow to expand it to the needed level.

3.2.1 Packet Filtering

IseHarvest's initial functionality is to be able to parse and identify web files, such as webpage HyperText Markup Language (HTML), Cascading Style Sheets (CSS), images and as many other file formats that maybe be commonly placed on web pages. This necessitates the need to identify when files are being sent over the wire. At the point in the application where IseHarvest actually sees packets received by Tcpflow functionality, we have stripped off the TCP and IP packet headers, leaving only HTTP or any other protocol that TCP packets may contain. [11] Since IseHarvest is intended to primarily reconstruct web traffic from over the Internet, it focuses on the HTTP protocol,

until other protocols are developed.

In HTTP, packet headers are important when browsing through the internet. A web browser will commonly send a GET packet to a designated web server when requesting a web page. These GET requests are commonly structured similarly to these examples:

```
GET / HTTP/1.1
Host: www.iastate.edu
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.12) Gecko/20061201 Firefox/2.0.0.12
(Ubuntu-feisty)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Or:

```
GET /styles/07/isutopB2.css HTTP/1.1
Host: www.iastate.edu
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.12) Gecko/20061201 Firefox/2.0.0.12
(Ubuntu-feisty)
Accept: text/css,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.iastate.edu/
```

These packets are identified by a “GET” which is then normally followed directly with either a “/”, in the case of root web pages, or by the name of the file that is about to be transferred over the network. The complete filename is constructed for IseHarvest using both the name identified following the “GET” and the host specified after the field “Host:”, “www.iastate.edu” in this case. Therefore the complete filename are created as “/www.iastate.edu/” for the first packet and “/www.iastate.edu/styles/07/isutopB2.css” for the second packet [12]. Notice, that the first packet is only specified by the directory of

“/www.iastate.edu”. IseHarvest appends “index.html” to the end of the filename in order to be able to write the document to a file. This is consistently followed for main homepages. Therefore the complete filename for the first packet is “/www.iastate.edu/index.html”.

Once the filenames are identified, IseHarvest creates each directory and file listed in the complete filename. This allows IseHarvest the added benefit of simulating the directory structure of a given web server, preventing files with the same name from overwriting each other, and providing insight into how a web server was designed. Concurrently, this method of file naming and association allows multiple websites to exist together without impacting each others files. Thus, the GET request provides a convenient method to identify files being transferred over the wire, determine what to call a file in order when writing it to the hard drive, and keep track of multiple files with the identical names.

Once the GET request is detected traveling from client to server, it is necessary to watch for file data sent from the server to client computer:

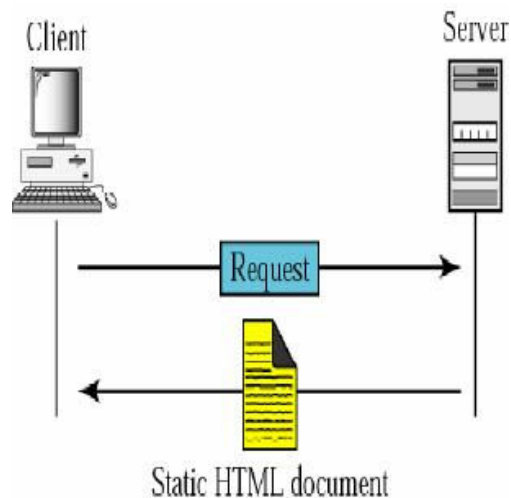


Figure 7: GET Request and Document [12]

Therefore it is necessary to modify *flow_state_struct* in order to account for the reversed direction. The new object, referred to as a *session*, contains an additional *flow* object, so it keeps track of both directions (to and from the client (source) computer and the server (destination)) to make it easier to identify where traffic belongs:

```
typedef struct session {
    struct session *next;
    flow_t src_dest;
    flow_t dest_src;
    tcp_seq isn;
    FILE *fp;
    char filename[NAME_SIZE];
    long pos;
    int flags;
    int last_access;
    int data_type;
    int chunked;
    int gzipped;
    int directories;
} session;
```

Like Tcpflow, most packets have a *session* corresponding to it in the local list of active TCP streams. This list is again used to ultimately determine where received data packets will need to go based on the *flow* object that corresponds to each packet, referring to the source and destination IP addresses and port numbers extracted from each packet. Additionally, there are a number of differences between this object and the original *flow_state_struct* defined by Tcpflow.

First, besides containing two *flow* objects instead of one, each *session* contains the actual filename to write extracted data to in order to make file identification and re-linking (section 3.2.3) easier. The other major changes are that the *session* contains three additional variables; *data_type*, *chunked* and *gzipped*. The variable *data_type* is used to identify what type of file is being transferred, whether HTML, JPEG, etc... The value *chunked* is used to identify if a file being send over the network has been segmented by

the server. This only happens when the server dynamically creates files and does not know how large a file is when sending. The server will flag the data as *chunked* and sends small pieces of the data over at a time. IseHarvest therefore needs to be able to parse out when data is segmented so that it can re-assemble the data when its transfer is complete to make the data readable. The final variable, *gzipped*, is similar to *chunked* in that it does not apply to every packet transferred between computers. For larger files transferred between web servers and client computers, it is prudent to compress a file to a smaller size to improve the speed that pages are loaded by the client's web browser. When this happens, the server flags the data as *gzipped*, or compressed, so that the client computer understands to uncompress the file before attempting to read and display the data to the user. Likewise IseHarvest needs to be aware of compressed data in order to uncompress and write it to a file to readable by the user and any tools that the user may use in conjunction with IseHarvest.

At this point, IseHarvest has identified all the necessary information from the initial HTTP packets. Once a file has been identified with a GET request, IseHarvest begins looking for the data packets. From here on out, any data packets without a corresponding active session are discarded and ignored. Since IseHarvest is a passive application, if there are missing or corrupt packets in the capture, it cannot correctly re-assemble the files. It assembles with the data it has and leaves the files for analysis. Data extraction and reconstruction are discussed in the following section.

3.2.2 Data Extraction and Reconstruction

This section is going to discuss the methods and processes used to implement data

extraction from the corresponding network packets. This includes identifying the data packets, identifying the corresponding output, un-segmenting the data, and decompressing the data files to enable them to be easily read by a user when reconstructed.

After filtering the captured packets for active sessions, the first step in the reconstruction process is to identify the corresponding output file. If you recall from the previous section, upon receiving a GET request that the client browser sent out we extracted necessary information from that packet. This information included the filename, IP addresses (source and destination) and port numbers that are involved in the transmission of this data file. Upon receiving a data packet, the first values checked are the IP addresses and port numbers associated with a given data packet. These values are compared to the locally kept list of active sessions. If an active session contains a *flow* object matching the IP addresses and ports identified with this data packet, the data is stored to the file linked to by the corresponding active *session*. To ensure data segments are written to the correct locations in corresponding files, the sequence number of the packets is used as a write address. When the initial GET request is detected for a data file, the sequence number for that session is identified as the “zero” index for this data packet, and used as the initial place marker for writing data to the output file. Since the sequence numbers correspond to the number of bytes written by previous data packets (sequence number 1275 higher than the initial place marker means 1275 bytes were written before the current packet and 1275 is the current location to start writing), they can be used to conveniently identify what location in a file to write the current data fragment. This ensures data is written to the correct location in the corresponding output

file, regardless of the order a packet has been received. Since IseHarvest is a passive application, and if any packets are corrupt or missing in a traffic capture, the resulting files may be potentially unreadable.

When data files are completely received, there remain small issues of un-segmenting *chunked* files and decompressing data if necessary. Due to the formatting of segmenting data files, “un-chunking” those files is fairly simple. A data file is *chunked* most frequently when a server is dynamically creating files for a client. Files are segmented using simple protocol that is delimited by carriage returns and line feeds (CRLF, two bytes of the value *0x0d0a*). [10] When data is segmented it is broken into fragments of various sizes. It is segmented by using an initial length, in bytes, which is followed by CRLF. Following the initial length is the data segment of the specified length. Once the data segment has reached the specified length, it is followed by another CRLF and the process repeats with the length of the next segment and data of the new specified length. This pattern continues until a length value of zero is parsed followed by two CRLFs to signify the end of the data being sent. To un-segment the data, IseHarvest proceeds by reading each specified length and the corresponding data segment. As IseHarvest reads each length, it writes only the corresponding data to temporary file but not the length value. Once all data has been read and length values removed, the original data file is deleted and the temporary file changed to the old name.

At this point IseHarvest has removed extraneous information from the data and can proceed to decompress the data files if necessary. If a designated file was flagged as being *gzipped* by the initial packet sent from the server to the client, it is a simple matter to decompress the data. IseHarvest executes the application *gzip*, assuming it is an

installed application, and the data file will be automatically decompressed to allow the local computer to easily view and use the received file. The next step for IseHarvest is to re-link HTML and CSS files so they reference files locally.

3.2.3 Re-linking

When all data files are received and re-assembled on the local computer, each image, video, document, etc... can be easily viewed individually. However, it is desirable to view entire web pages that were accessed through the network. Therefore, the next step is to rewrite HTML and CSS documents to link to images and documents locally instead of attempting to connect to the web server. As stated previously, when a website is viewed all necessary pieces (images, documents, etc) are sent to the client's web browser to allow the client to see the web page correctly, unless cached by the client browser. When an HTML document is viewed, all components are linked to as if they were on the web server, as follows:

Example of HTML and CSS original links.

```
</a>
<script src="/js/gamewatcher07.js" type="text/javascript"></script>
<a class="abc" id="u" href="http://www.iastate.edu/index/alpha/U.shtml">U</a>
```

This code represents an HTML document linking to an image, a JavaScript document, and a SHTML page. The Iowa State University homepage was again used as an example, with all of the original HTML and CSS code, looks as follows in figure 8 when viewed from a browser over the internet. As shown, the entire web page is easily viewable using a regular web browser. However, using IseHarvest, the web page should be viewable locally without an internet connection. The following image is how the Iowa State University homepage looks with the HTML and CSS code not re-linked, but not able to

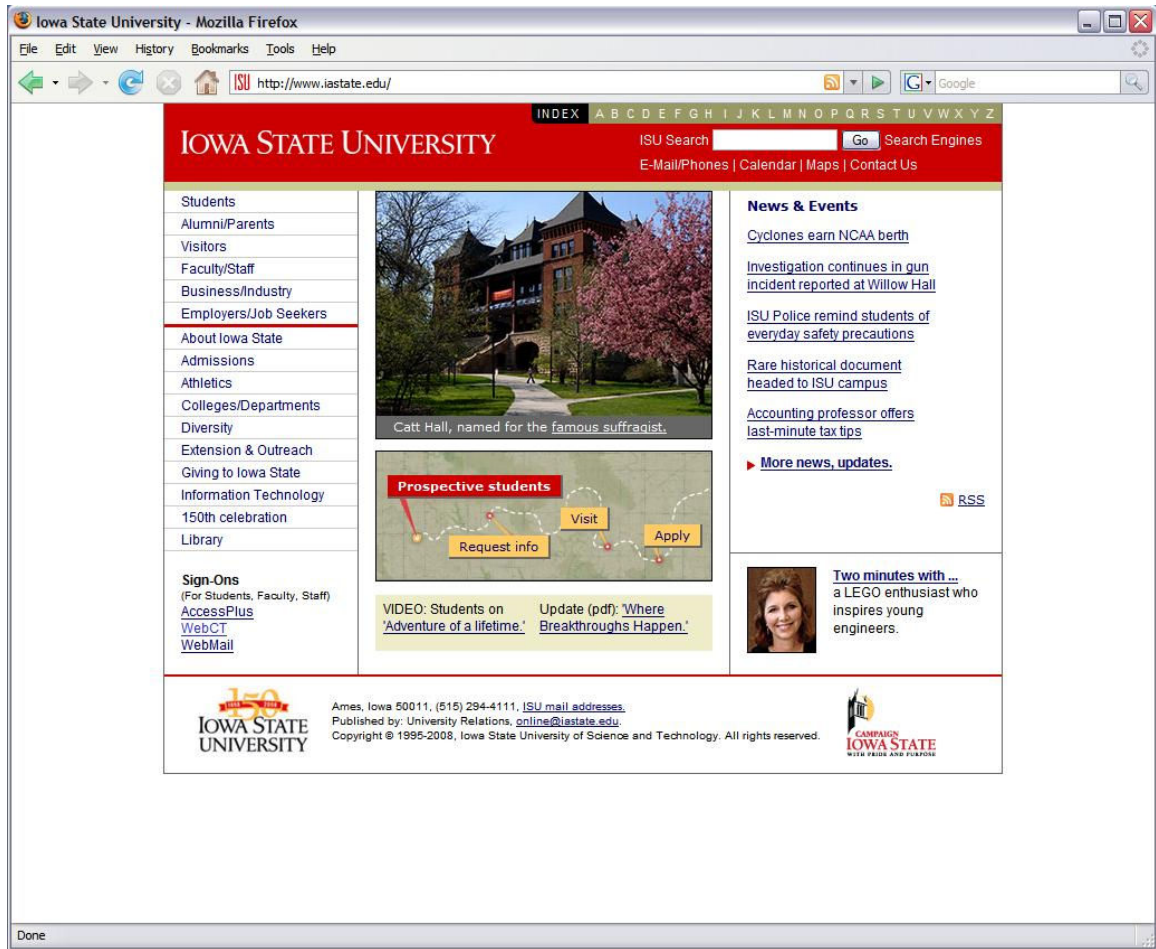


Figure 8: ISU Homepage with web browser

access the a web server in order to display referenced images and documents, in other words how only the HTML looks without an internet connection as in figure 9. This page is viewable and it is possible to determine a lot about the web page from the links and textual information contained. This can represent a problem however, because the intent of IseHarvest is to allow a third party to view as much of a websites as possible without directly communicating with the web server that hosts a specified web page, to see everything that the target sees in a sense. This requires IseHarvest to modify any HTML or CSS files on the fly to allow the complete web page to be viewed. If links were not modified or redirected, then the browser would attempt to download all data

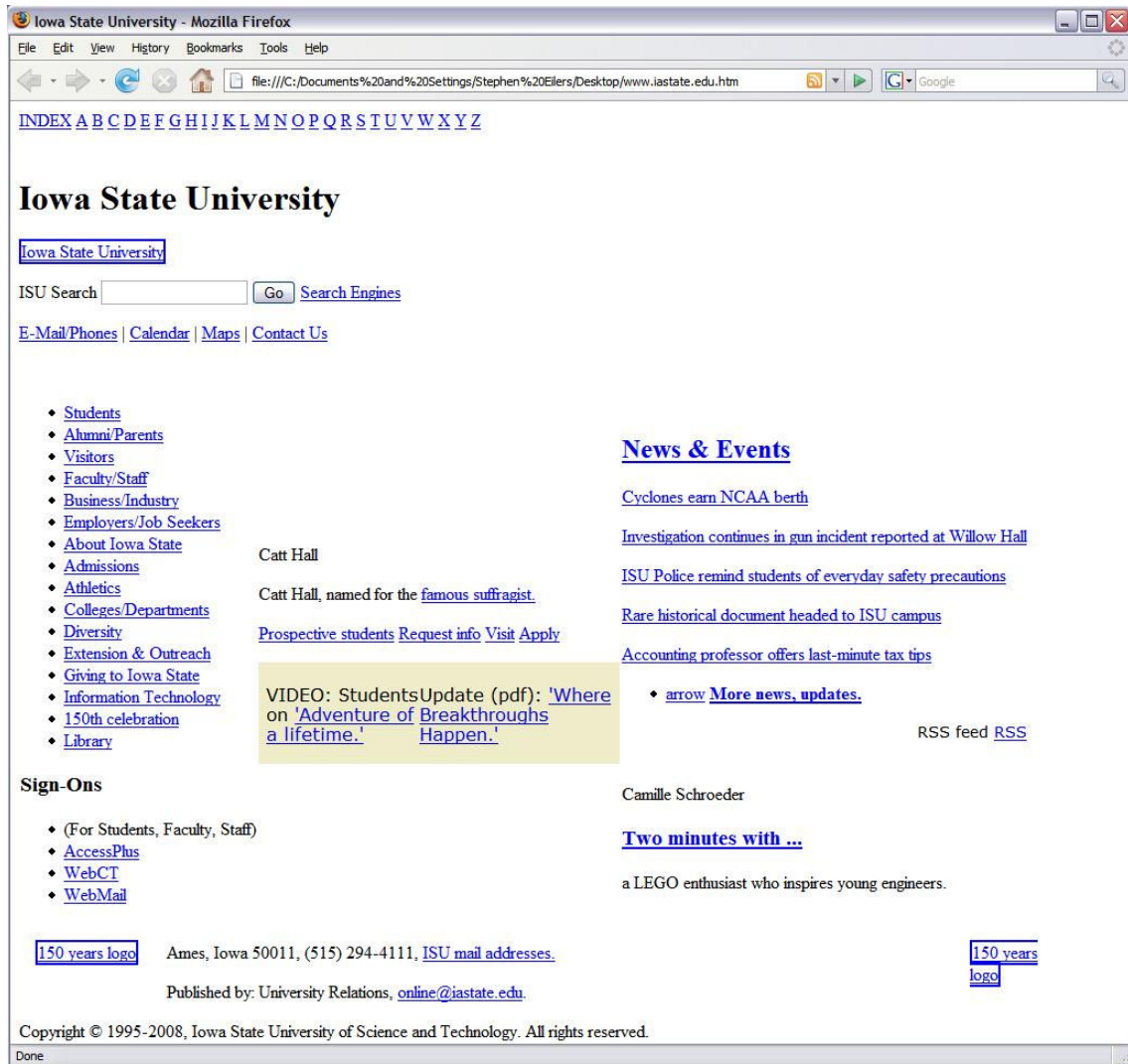


Figure 9: ISU Homepage with IseHarvest without re-linking

from that original web page to view the website. An example of what links should look like after being re-linked, is as follows:

Example of re-linked HTML and CSS links.

```
</a>
```

```
<script src="/js/gamewatcher07.js" type="text/javascript"></script>
```

```
<a class="abc" id="u" href="../../www.iastate.edu/index/alpha/U.shtml">U</a>
```

The first step in re-linking files is to parse for entries that include websites or filenames. The most common examples or “tags” linking to other files in HTML or CSS

code are referenced in the above examples and include: *img src* for image files, *script src* for java related files and *href* for linking to other web pages and documents [12]. Once a specific tag is located, it is often in the format of a complete path name to specific files. An example of this is “*http://home/web/image.jpg*” or “*/home/web/image.jpg*”. These links may reference another website if containing an “*http://*” or are an absolute path from the local root web directory. To re-link properly on the local machine, this path needs to be changed from “*http://home/web/image.jpg*” to simply “*../home/web/image.jpg*”. The “*..*” is necessary because if the link is referencing another website, it needs to go down to a lower directory in order to enter the correct website captured by IseHarvest.

Once a specific link is modified appropriately, it is written to the data file and once all specific links have been re-linked to the local machine, it is now potentially feasible to view the website directly off the local computer. The following webpage, figure 10, is again the Iowa State University homepage, except now run through the entire IseHarvest application.

The shown webpage is identical to the original website when viewed from any computer over an internet connection. This page was displayed from the images and data captured from another computer viewing the Iowa State webpage. Once the captured traffic was entered into IseHarvest, the internet connection was disabled for the viewing computer and viewed locally without any possibility of a connection to the original web server. The originally viewed site and the re-constructed site are identical when viewed through the browsers and, as previously discussed, the only differences between the two are the location of the data corresponding to the webpage; on a remote web server or

locally contained.

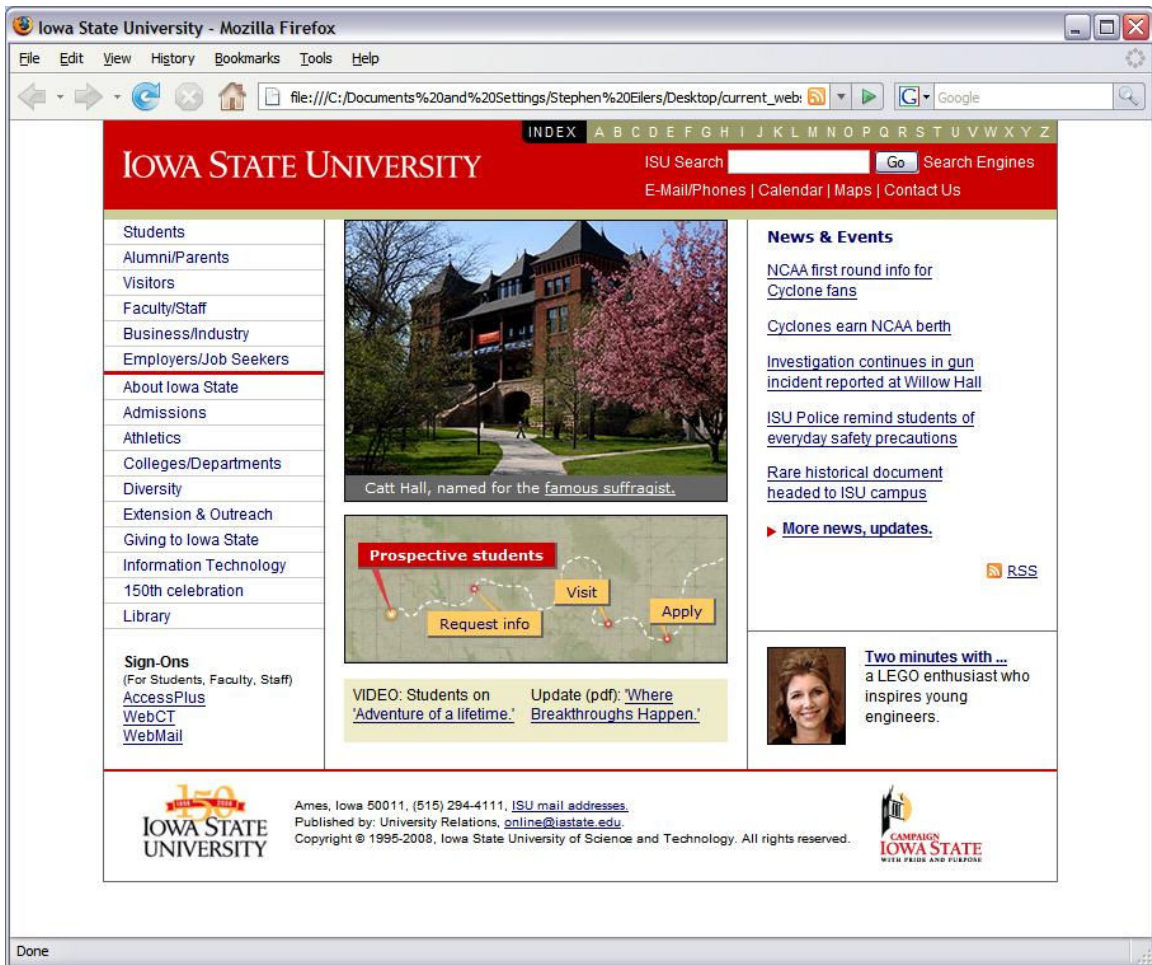


Figure 10: ISU Homepage with IseHarvest with re-linking

3.2.4 Framework

As stated, in addition to performing HTTP reconstruction IseHarvest is intended to provide a framework for easy integration of additional protocol handling. There are multiple layers of extensibility available within IseHarvest. These layers include implementing additional network layer protocols (IP, ARP), transport layer protocols (TCP, UDP), and other application layer protocols (HTTP, FTP, etc...) that may be integrated into IseHarvest.

In order to incorporate additional protocols, IseHarvest is layered to handle the different protocols. Currently, it ignores all packets except for IP packets and handles them with the following function call:

```
process_ip(const u_char *data, u_int32_t length);
```

Parameters to this function include the packet data (*data*) and the length of the packet (*length*). Internally to this function, the IP header is read and stripped away as no longer needed. If the packet is identified as a TCP packet it is passed into the following function, otherwise the packet is not supported and discarded:

```
process_tcp(const u_char *data, u_int32_t length, u_int32_t src, u_int32_t dst);
```

Parameters passed into this function include the data packet (*data*), the length of the TCP packet (*length*), source IP address (*src*), and the destination IP address (*dst*). Similarly to the IP function, the TCP header is read and stripped since it is no longer needed. Finally, the packet is passed into the following function to be processed for HTTP:

```
process_http(flow_flow, const u_char *data, u_int32_t length, u_int32_t seq, u_int32_t ack_seq);
```

Parameters include the data packet, length, source and destination addresses, as well as sequence numbers (*seq* and *ack_seq*) to identify placement of the data packets.

In order to implement an additional protocol, it is necessary to insert new function (*process_protocol*) in the correct layer in the application. This will depend on the protocol: File Transfer Protocol (FTP) would be placed in the transport layer and the function call referenced in the main header file (Tcpflow.h) for example. Including the source modules in this manner should allow additional protocols to be easily inserted into IseHarvest. Compiling the new code will be covered in section 3.2.5.

3.2.5 Compiling

Tcpflow had an extensive and effective system of compiling to ensure that the application could be compiled on a variety of Linux and Unix operating systems. The majority of this compiling system, including makefiles and configuration files, was simplified to make compiling of the application easier to perform. Makefile construction is as follows:

```
http.o: http.c
gcc -DHAVE_CONFIG_H -I. -g -O2 -Wall -Wno-unused -c http.c
(for each source file)
IseHarvest: datalink.o flow.o main.o tcpip.o util.o http.o
Gcc -g -O2 -Wall -Wno-unused -o IseHarvest datalink.o flow.o main.o tcpip.o util.o http.o
(for main executable)
```

In order to compile the IseHarvest application, there is only a need to enter one command from inside the source code directory. The command needed to compile the IseHarvest system is:

```
make
```

This command automatically compiles the source and header files inside the source code directory. Once compiled, an executable file labeled *IseHarvest* is created and may be used to execute the application.

In order to compile in a new source file, it needs to follow the pattern established by the `http.o` and similar compile lines. Then simply add in the `<new>.o` into the IseHarvest compile command and the application should be compile-able.

Operating instructions are documented in the following section.

3.2.6 Operating

IseHarvest executes similarly to Tcpflow since it contains many of the same

functionalities, with the added functionalities of packet re-assembly. Once the program has been compiled, there are a number of parameters that may be entered to enable functionality to be maintained.

The default functionality is data extraction and re-assembly from a specified PCAP formatted file as input. IseHarvest's default functionality is executed as follows:

```
IseHarvest -r <PCAP file>
```

When executed in this manner, IseHarvest reads the specified PCAP file and assembles the data as specified in the implementation section. The data files outputted are placed in the local directory that IseHarvest was called from. Currently all files are outputted and placed into the current directory as detected and assembled. Currently, the application is being expanded to identify what web page is being monitored; all files that are detected to belong to this web page will be placed in a common directory for ease of identification and prevent data files with identical names from overwriting previous files.

It is possible to run IseHarvest on a live capture of data traffic. This mode of operation may have two affects; 1) running on any internet connection which receives a useful amount of internet traffic may cause IseHarvest to slow down due to the large number of read and write accesses performed for each packet and when data files are re-linked, and 2) if IseHarvest is slowed down while attempting to keep up with a live internet connection it could potentially lose packets and end up with incomplete data files. IseHarvest may be operated in the live operation mode with the following command and root privileges:

```
IseHarvest -d <device>
```

An additional useful functionality originally provided by Tcpflow is to limit the

number of files that are opened at a single time. This cuts down on the number of accesses to the hard drive of the local computer but also slows down the local computer because only a limited number of files may be open at one time. This functionality is implemented as follows:

```
IseHarvest -f <#> -r <PCAP file>
```

When a number # is entered, this limits the number of total files that may be open at a single time to #. Otherwise the application attempts to detect the maximum number of open files the operating system can handle. This may improve memory usage, but consequently slow down the application due to continually opening and closing files with large traffic captures.

IseHarvest offers other functionalities based on the original Tcpflow application which can be viewed by looking at the Tcpflow man page or running the following command for a brief help on available commands:

```
IseHarvest -h
```

This displays a limited help menu to explain current options available to IseHarvest.

CHAPTER 4. TESTING

IseHarvest is tested with a variety of testing procedures. Initial tests were operated by performing Wireshark captures of simple browsing to websites such as the Iowa State University homepage (www.iastate.edu), the Slashdot website (slashdot.org) and others.

To run IseHarvest on a live capture, you simply follow the operating instructions in section 3.2.6 with a valid network device and IseHarvest will reconstruct traffic as it

receives it.

To run IseHarvest on a capture file, you must create a capture to run IseHarvest upon. To create the traffic capture, the first step was to clear the local web browser's cache to ensure that the web browser would load the entire web page. If the cache was cleared, it ensures that the entire web page would be transferred over the network. If one or more packets are not transferred, then the web page would be missing pieces on the monitoring computer. The next step is to use Wireshark to begin a data capture and, while Wireshark is running, access the desired web page (Iowa State homepage in this case). Wireshark will capture all the packets being sent to and from the web site and display them to you as follows in figure 11.

The screen shot in figure 11 shows Wireshark displaying an initial GET packet for the Iowa State homepage. The traffic that has been captured is now saved to the local computer, using Wireshark, in a PCAP data format. Once the file is written, IseHarvest is then called with the PCAP file as a parameter. Once operation on the PCAP file is complete, data is outputted to the local computer as displayed in figure 12.

The displayed files in figure 12 can then be accessed and used to open the web page locally. The Iowa State University web page as displayed previously in this document is the output of executing the *www.iastate.edu/index.html* file locally. This operation was successfully used to extract various web pages. Additionally, IseHarvest successfully captured the Iowa State University homepage as well as other web pages in a live capture operation. Testing on web pages such as *Slashdot.org* has successfully extracted and viewed the information with approximately an 80% success rate.

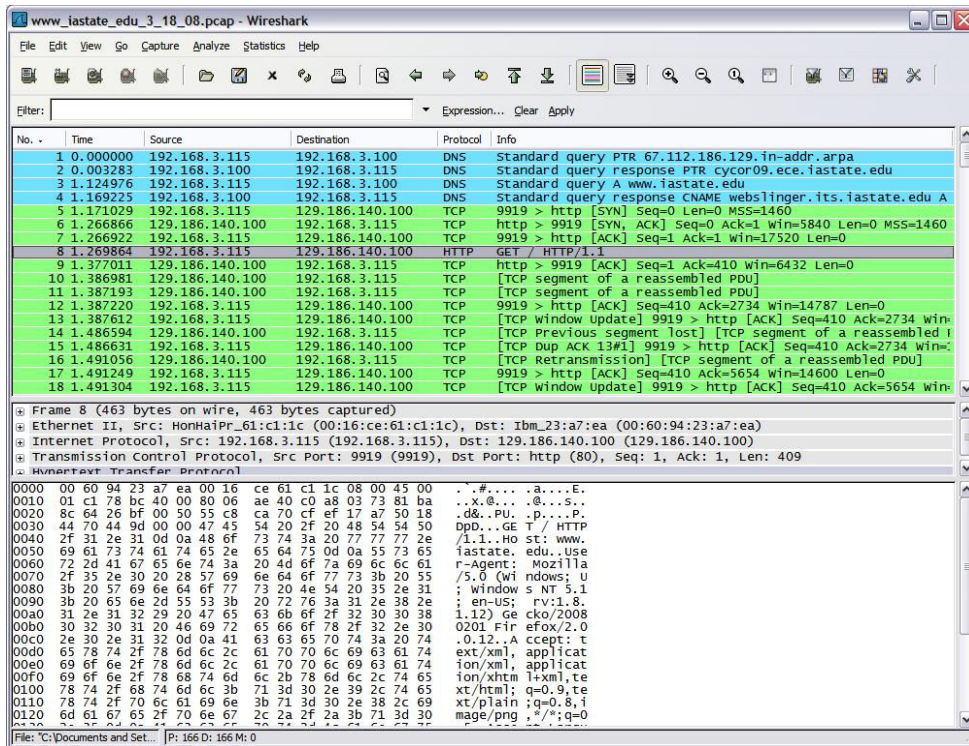


Figure 11: Wireshark Capturing ISU Homepage for testing

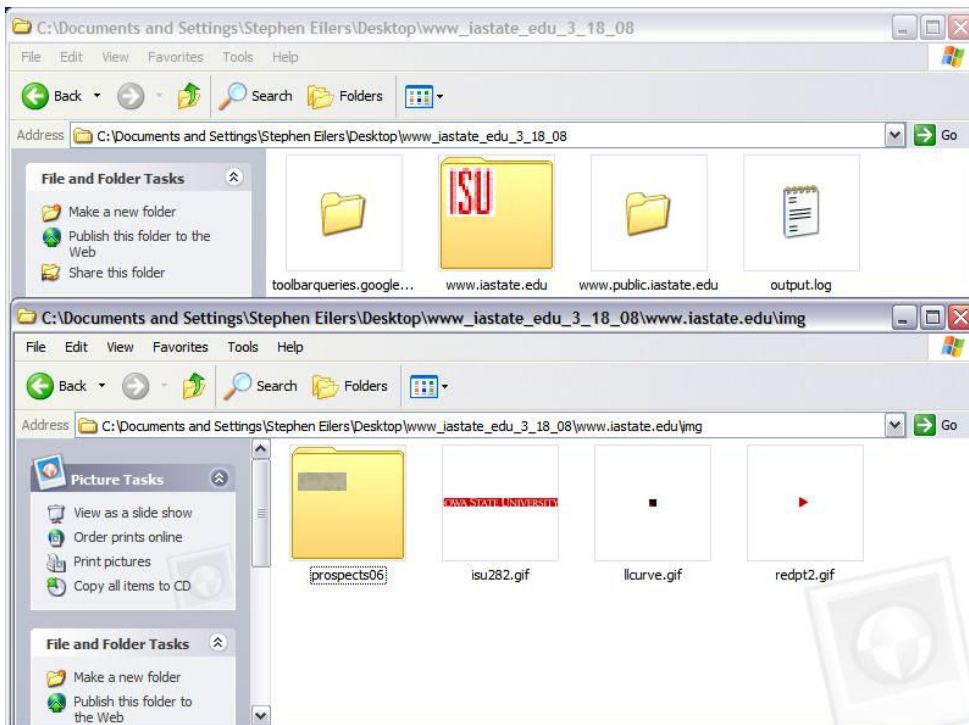


Figure 12: ISU Homepage Example Directory Output with IseHarvest

In figure 13, the circled location in red shows an image that failed to be viewed.

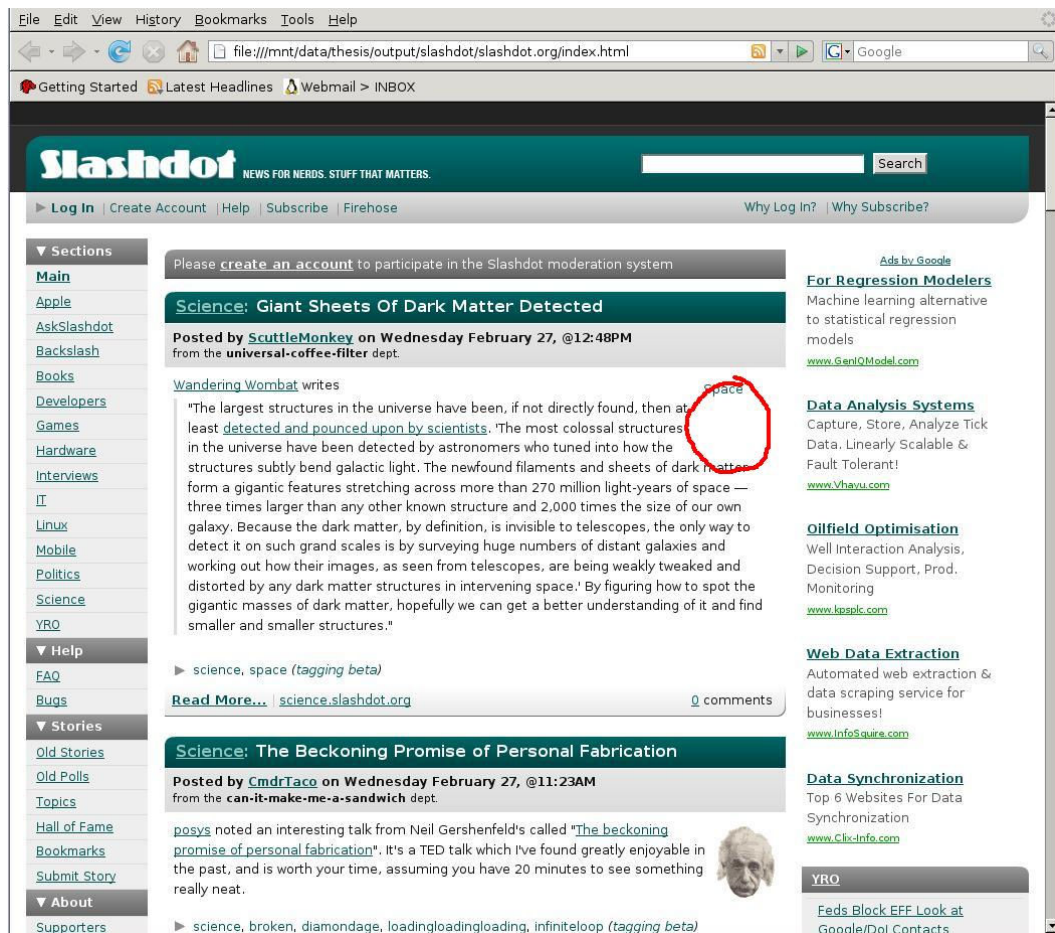


Figure 13: Slashdot with IseHarvest [14]

For continued testing and fine-tuning, captured traffic files from the National Cyber Defense Competition (CDC) held by Iowa State University this school year are being used to continue improving IseHarvest. These tests provided large data files that rigorously test IseHarvest on a much larger scale than simply looking at individual web pages. IseHarvest successfully extracts the majority of the data from the capture files, but due to attacks and other files, IseHarvest can run into issues with re-linking HTML and CSS files from ISEAGE.

IseHarvest has been put under a number of tests that show its effectiveness at extracting HTTP traffic from captured data files. It has successfully extracted the majority of data from captured files with the exception of some dynamic content used in today's internet. It currently extracts and assembles data more accurately than when viewed with the commercial tool by Iris®.

CHAPTER 5. CONCLUSION

IseHarvest utilizes many abilities that were built into Tcpflow, providing a stepping stone from the original functionality to implement data extraction in HTTP. It has been developed in a way that will allow future developers to implement new functionalities with relative ease. This thesis discussed the implementation of its current functionality and how it works to extract and re-assemble HTTP traffic. Following that model, future protocols can be implemented into this framework to allow adaptation and expansion.

I feel IseHarvest successfully meets its goal of providing a network traffic analyzer for web traffic content in HTTP traffic. I feel it successfully meets its goals to read captured TCP traffic, extract HTTP data, reconstruct complete data into viable documents, videos and web pages, re-link HTML and CSS pages to allow local viewing, and provide a framework for future modifications and extensions to be added to IseHarvest.

5.1 Limitations

As mentioned throughout this document, IseHarvest has a few limitations. IseHarvest successfully extracts data from complete streams and can successfully re-link the majority of HTML or CSS files that it intercepts. However, it is currently unable to account for all possibilities. The following are a number of the known limitations IseHarvest may run into.

The first limitation deals with IseHarvest's ability capture traffic. To enable users

to more efficiently browse the Internet, web browsers cache web pages locally to allow faster loading. When a portion of a web page is cached locally, it will not be sent from the web server to the client computer. If a file or image is not sent between the web server and client, then IseHarvest is unable to intercept that data. Therefore, IseHarvest is unable to completely load web pages which contained cached content since it is intended to passively capture and only analyze what it sees.

A second limitation is one that affects all network analysis tools, dropped or corrupted packets. When a captured packet is corrupt or non-existent, it means IseHarvest is unable to complete the file that packet belongs to. A single corrupt segment of data causes an entire file to be corrupt. In normal circumstances, a web browser would potentially request that packet be sent again, but if not then IseHarvest is unable to guarantee a readable data file when it does not see all necessary packets.

An additional limitation of IseHarvest is in re-linking dynamically generated content. As web servers become more intelligent, they keep track information about client web browsers from cookies, databases, etc... An example of this issue is when a web server dynamically generates a web page for a client. It is possible to generate a web page such that two separate links will direct a user to an identical web address or Uniform Resource Locator (URL). The web server is able to understand and differentiate these two URLs, sending the correct information to the client when necessary. IseHarvest, however, is unable to detect these issues. IseHarvest creates one file and then overwrites it with data from the second link. This will cause both links to refer to the same place since IseHarvest is unable to refer back to the web server. This is an issue that, even in the future, IseHarvest will most likely be unable to effectively solve.

5.2 Future Work

As discussed earlier, it is necessary to re-link HTML and CSS files in order to ensure that the local computer can view the web pages correctly. While re-linking works very well on content that is somewhat static for a web site, it is not 100% effective for highly dynamic content on websites. Websites which generate content specifically for each client can have very dynamic pages that are more difficult to keep track of. While IseHarvest can still successfully partially reconstruct those web pages, dynamic links can include symbols or characters that disrupt IseHarvest's ability to successfully re-link and load extracted files. Improving the functionality of IseHarvest to more effectively adapt to dynamic web pages will allow IseHarvest to improve its accuracy when reconstructing captured web pages.

As stated, IseHarvest is intended to be a framework for future protocols to be added to, and a major future goal would be to implement additional protocols and improve on HTTP if/as it changes. There are number of additional protocols and applications which, if implemented in IseHarvest, would be extremely useful. These protocols include, but are not limited to, email (SMTP or POP protocols), chat clients (Jabber or AIM), streaming videos, File Transfer Protocol (FTP), etc...

An additional potential future modification to IseHarvest would be to insert filtering options for capture files. It could be desirable to search through a large amount of traffic for specific streams of data or data files that may include a key word, specific IP address, or any other search variable. This may be inserted at numerous locations in the

application depending on the developer's prerogative.

IseHarvest leaves options for many different ideas to be added and implemented in it. While the stated features would enhance IseHarvest, they should be implemented as the need arises to provide additional functionality.

REFERENCES

- [1] V. Jacobson, C. Leres, and S. McCanne. (2008, Jan.). The Tcpdump Manual Page. Lawrence Berkely Laboratory, Berkeley, CA, Jun 1989.
- [2] Gianluca Varenni, Ulf Lamping, F. Risso, L. Degioanni. (2007, Dec.). PCAP Next Generation Dump File Format. Internet Engineering Task Force. [Online]. Available: <http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html>
- [3] Elson, Jeremy. (2008, Jan.). The Tcpflow Manual Page. Circlemud.org [Online]. Available: <http://www.circlemud.org/~jelson/software/tcpflow/tcpflow.1.html>
- [4] Iowa State University. (2008, Mar.). Iowa State University Homepage. Iowa State University of Science and Technology. [Online]. Available: <http://www.iastate.edu>
- [5] Gerald Combs. (2008, Jan.). Wireshark Network Analyzer Man-pages. Wireshark.org. [Online]. Available: <http://www.wireshark.org/docs/man-pages/wireshark.html>
- [6] Filetransit. (2008, Feb.). Capsa 3.0. Colasoft. [Online]. Available: <http://www.filetransit.com/view.php?id=15885>
- [7] Javvin. (2008, Mar.). Network Packet Analyzer Capsa 6.7. Javvin Network Management & Security. [Online]. Available: <http://www.javvin.com/packet.html>
- [8] eEye Digital Security. (2008, Mar.). Network Security Traffic Analyzer. eEye Inc. [Online]. Available: <http://www.eeye.com/html/products/iris/>
- [9] Sourceforge.net. (2007, Dec.). Manpage of Pcap. TcpDump/Libpcap public repository. [Online]. Available: <http://www.tcpdump.org>
- [10] Dan Connolly. (2008, Jan.). Hypertext Transfer Protocol – HTTP /1.1. World Wide Web Consortium. [Online]. Available: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [11] Alberto Leon-Garcia, Indra Widjaja, “Applications and Layered Architectures,” *Communication Networks: Fundamental Concepts and Key Architectures*, 2nd ed. New York: McGraw Hill, 2004, ch. 2.
- [12] B. A. Forouzan, “World Wide Web: HTTP,” *TCP/IP Protocol Suite*, 3rd ed. New York: McGraw Hill, 2006, ch. 22.
- [13] YouTube. (2008, Mar.). YouTube: Broadcast Yourself. YouTube, LLC. [Online]. Available: <http://www.YouTube.com>
- [14] Slashdot. (2008, Mar.). Slashdot Homepage. Sourceforge, Inc. [Online]. Available: <http://slashdot.org>

ACKNOWLEDGEMENTS

I would like to thank my committee members for agreeing to be my on my committee and helping me in my graduate career. Special thanks to Drs. Jacobson and Licklider who provided feed back, understanding, and assistance when I needed it.

To my family and friends: thank you so very much for your love and support. Without you, my journey would have been long and painful. I hope to continue making you proud.