



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at <http://www.giac.org/registration/gcia>

Extracting Files from Network Packet Captures

GIAC (GCIA) Gold Certification

Author: Rebecca Deck, sdsecurityacct@hotmail.com

Advisor: Hamed Khiabani

Accepted: December 17th, 2015

Abstract

Extracting files from full packet captures can save security analysts a great deal of time. Time-consuming procedures, such as performing a complete forensic analysis on suspect machines, can often be avoided if analysts are able to extract files from the network traffic. There are several tools to perform this function, but they all have shortcomings. In order to make an informed assessment of packet captures, analysts must familiarize themselves with these limitations. This paper compares the capabilities of currently available tools which automate this task, explores the process of manually extracting artifacts from packet captures, and offers a script to extend the functionality of TShark to include file extraction. This will familiarize new security analysts with current tools as well as establish a baseline knowledge of how these tools function.

1. Introduction

Full content packet captures provide analysts with the ability to review exactly what has transpired on a network. Analysts neither have to rely on questionable logs nor perform guesswork when determining what data have been transferred. One of the benefits to utilizing full packet captures is the ability to extract specific files that are transferred between hosts on the network. These files can then be used for more in-depth analysis to determine the true nature of malware (Bejtlich, 2012). Many tools can remove raw files from network packet captures, but as with most forensic technologies, these tools have their limitations. An understanding of packet capture structures and how to access the packets' raw data allows analysts to answer many questions without resorting to time-consuming forensic analysis of systems.

Based on research from Simson Garfinkel, traditional computer forensics suffers from several issues. Traditional computer forensic methods require that a system be imaged before analysis can begin. Due to the large amount of data storage on current computer systems, this process often requires several hours. Moreover, some modern malware is capable of existing solely in the memory of a system; writing nothing to disk (Garfinkel, 2010). Log messages on compromised systems certainly help investigations, but can potentially be altered by an attacker (Allen, 2001, p. 238).

In contrast to host-based forensics, analysis of network packet captures can begin to provide answers within minutes, does not impact hosts, and is unlikely to be manipulated by an attacker. Properly placed network packet capture systems capture the exact data that traverses the network. This provides a record of the files used to initially infect a system, command and control traffic, and files that may have been exfiltrated by an attacker. An analyst can often reconstruct specific exploits and malware payloads; allowing the investigator to determine what software versions were targeted by an attacker and the purpose of the persistent malware delivered to the victim computer.

Capturing packets on modern computer networks is not without its own challenges. Chief among these are the increasing speed of networks and the massive amount of data that traverses them. This is especially significant in enterprise networks. Fortunately, not only have the purveyors of packet capture systems invested significant

Rebecca Deck, sdsecurityacct@hotmail.com

time and effort to address these issues, but approaches for open source network captures have also been developed (Banks, 2013). Therefore, this paper does not address these concerns.

2. File Extraction Basics

An understanding of the format in which packets are stored is necessary in order to understand file extraction techniques. The most common and well-supported format is the packet capture, or PCAP format. These files have a very simple structure containing a single global header followed by multiple groups, which consist of a packet header and packet data. This generic structure is easily seen in Figure 1 (Harris, 2015).



Figure 1. PCAP file format header and data layout (Harris, 2015).

These headers identify the generic PCAP format through the “Magic Number,” ensure that accurate time information is stored for each capture, and permit length checks to accommodate snaplen limits (Harris, 2015). Within most network communication, several layers of additional information are present within the raw network data. Specific protocol decoders are required to interpret this information and reconstruct files that are embedded within a packet capture. Figure 2 illustrates how this protocol data is layered using a screenshot from Wireshark.

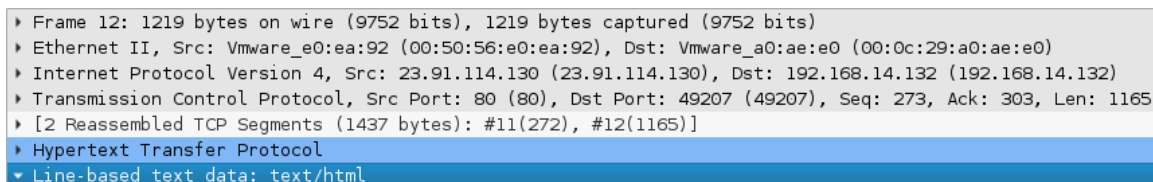


Figure 2. Wireshark displaying the various protocol layers on an HTTP request.

Standard file carving tools function by searching for various file format identifiers within a larger file. As is discussed in Davidoff and Ham’s book on network forensics, bytes that are identified as part of a file are then extracted. In packet captures, the additional bytes associated with the packet capture file format and protocols used during the capture are intermingled with the bytes that need to be extracted. Using normal file

Rebecca Deck, sdsecurityacct@hotmail.com

carving tools results in the various headers being embedded in the resulting files. Analysts can use hex editors to manually remove any extraneous protocol information from extracted files or from a packet capture themselves, but this is a painstaking process (Davidoff & Ham, 2012).

To compound the problem, network traffic does not always arrive in a predictable manner. Packets can potentially arrive out of order or be lost entirely and are subject to retransmission (Dharmapurikar & Paxson, 2005). Moreover, packet header lengths can vary due to TCP options, so simply assuming constant header lengths is impractical. Therefore, any tool to extract files from a packet capture must have the ability to not only remove packet capture headers from the raw file data, but also interpret networking protocols and reassemble data streams. This makes creation of a new file extraction tool a daunting task.

3. File Transfer Protocols

Since files may be transferred over myriad protocols, any application that extracts files from network traffic must support at least the most common of these protocols. Four common unencrypted file transfer protocols are Hypertext Transfer Protocol (HTTP), Server Message Block (SMB), File Transfer Protocol (FTP), and Trivial File Transfer Protocol (TFTP). Encrypted protocols, such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Secure Shell (SSH), must first be decrypted before any files can be extracted. Decrypting these protocols is a significant task and is beyond the scope of this paper. Therefore, this paper covers only the ability to extract files transferred using the HTTP, SMB, FTP, and TFTP protocols that are contained within packet captures.

The HTTP protocol is of particular interest during analysis. Research shows that 90 percent of initial malware infections take place over the HTTP protocol (Palo Alto, 2013). As of June 2015, all of the major exploit kits served their exploits over an unencrypted protocol (Duncan, 2015). When analyzing HTTP communication, one factor to consider is the use of gzip compression for transferring content. If tools are not able to cope with this compression, analysts must manually perform this procedure.

Rebecca Deck, sdsecurityacct@hotmail.com

SMB is frequently used in internal corporate networks for transferring files and system management. Most modern networks block the SMB protocol between internal systems and the internet (Brenton, 2006). Therefore, SMB will rarely be the initial source of malware. However, SMB packets frequently contain evidence of system misuse, pivoting to adjacent systems, or data exfiltration.

Although it represents a security risk, FTP is still used to transfer files today (Allman & Ostermann, 1999). Recently, more secure protocols have supplanted it for the transfer of sensitive files. Many systems that are deemed to be internal and have little chance of having their network traffic intercepted by an attacker still rely on this protocol. Several operating systems also include FTP clients by default. These reasons make FTP a viable method of removing data from compromised systems.

The TFTP protocol is used in much the same way as the FTP protocol. It is often installed by default on systems and is capable of data exfiltration. The feature set of TFTP is much smaller than that of FTP; it allows only downloading or uploading of files. TFTP has neither the ability to list folder contents nor to manage directory structures. Another advantage of the TFTP protocol is that it uses the UDP protocol for communication. This is useful because firewall configurations that implement egress filtering may be configured to limit TCP traffic, but leave UDP traffic unregulated.

4. File Extraction Tools

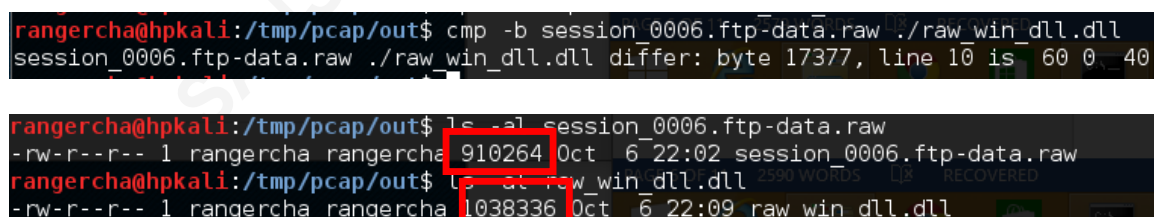
Several tools exist that support extraction of files from packet captures. These tools focus heavily on the HTTP protocol and often favor bulk extraction. To provide a comparison of these tools, each one was used to process packet captures containing HTTP, SMB, FTP or TFTP file transfers. The packet captures were either taken between test machines or downloaded from a data set of example recordings of malware infections (Parkour, 2015). After extraction, the file hashes and sizes of the output files were compared. Standard Linux tools and the Bless hex editor were then used to examine any differences between files.

Rebecca Deck, sdsecurityacct@hotmail.com

4.1. ChaosReader

ChaosReader is a tool designed to produce easily readable metadata and extract files from packet data. This tool produces HTML output detailing the connection information between all hosts contained within a packet capture. ChaosReader is also designed to extract files that have been transferred over HTTP and FTP (Gregg, 2004). Although simple to use, ChaosReader results in a large number of output files. This often makes it difficult to find pertinent data. In order to extract files, ChaosReader must be run with the `-raw` switch to extract raw files. This produces `.raw` files that should contain the exact files transferred during the packet capture being analyzed.

During testing, this tool sometimes resulted in the extraction of incomplete files. Following extraction, output files were compared using the Linux `cmp` command-line tool. This was further confirmed through the use of a hex editor and a check of the file size for the extracted `.dll` file compared to the one that was actually transferred. This is detailed in Figure 3.



```
rangercha@hpkali:/tmp/pcap/out$ cmp -b session_0006.ftp-data.raw ./raw_win_dll.dll
session_0006.ftp-data.raw ./raw_win_dll.dll differ: byte 17377, line 10 is 60 0 40

rangercha@hpkali:/tmp/pcap/out$ ls -al session_0006.ftp-data.raw
-rw-r--r-- 1 rangercha rangercha 910264 Oct  6 22:02 session_0006.ftp-data.raw
rangercha@hpkali:/tmp/pcap/out$ ls -al raw_win_dll.dll
-rw-r--r-- 1 rangercha rangercha 1038336 Oct  6 22:09 raw_win_dll.dll
```

Figure 3. Comparison of files extracted with ChaosReader to the actual files transferred. This failure was confirmed with several other file transfers using both HTTP and FTP protocols. Therefore, ChaosReader is ill-suited for file extraction and should not be used for this purpose.

ChaosReader is effective at generating connection information for packet capture files, but delivers unreliable results when extracting files. The application reports that it successfully extracts files, but testing shows that this is not the case. Since these files may not be usable for further analysis, other tools should be used by analysts when attempting to extract files.

4.2. Foremost and Tcpflow

Several sources suggest a combination of using Tcpflow and Foremost to extract files from network packet captures (Soderberg, 2010). Foremost is a file carving tool originally designed to extract files from disk images. Ordinarily, this would make it unsuitable for processing network packet captures for all the reasons described earlier in this paper. However, Tcpflow is able to parse a network packet capture, handling protocol headers, fragments, and out of order packet delivery. Once the data streams are reassembled, then standard data carving tools – such as Foremost – can retrieve the raw files. Since Foremost is a carving utility, its ability to extract files is limited to formats for which it has been programmed. While this list does contain 18 common file formats, several are missing (Kendall & Kornblum, n.d.). This includes file formats that are often included in exploit kits such as Java archives and Flash files.

The proposed use of Tcpflow is to export data streams into individual files (Soderberg, 2010). The individual files are concatenated into a single file, and then parsed with Foremost. This process is detailed in Figure 4.

```
rangercha@hpkali:/tmp/tcpflow$ tcpflow -r smb_dll.pcap -o ./output/
rangercha@hpkali:/tmp/tcpflow$ cat ./output/* > ./output/dump
rangercha@hpkali:/tmp/tcpflow$ foremost -i ./output/dump -o ./output/foremost_out/
Processing: ./output/dump
|*|
```

Figure 4. Illustration of combining Tcpflow and Foremost to process a packet capture.

In practice, Tcpflow was found to not properly process several protocols above layer four of the OSI model. The extracted files were found to be different than expected. The reason becomes obvious upon closer inspection with a hex editor. In this example the SMB protocol was used to copy files from a computer to a network share. As is shown in Figure 5 the SMB protocol data is embedded in the executable file as extracted by Foremost.

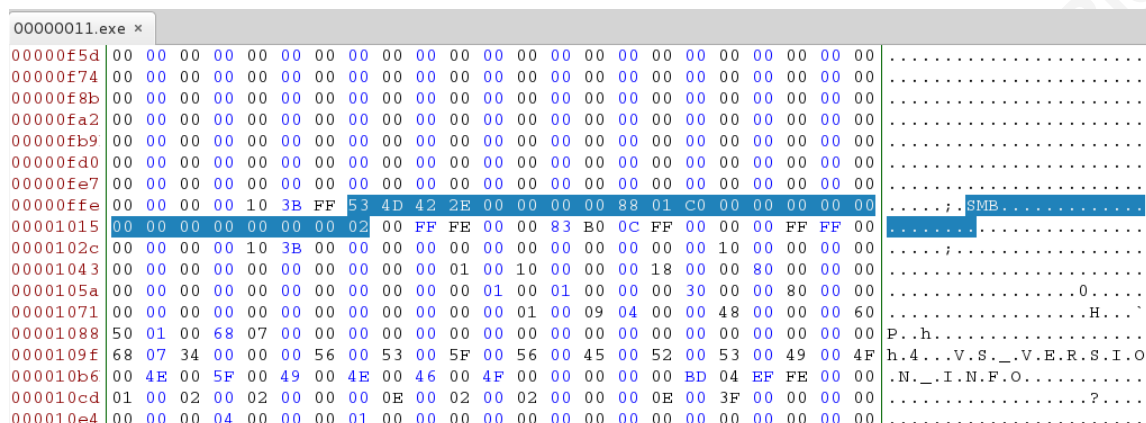


Figure 5. Residual SMB (Server Message Block) protocol data embedded in a file extracted with Tcpflow and Foremost.

However, some protocols are unaffected by this lack of interpretation. One example of this is the FTP protocol. When files are transferred over FTP, the protocol does not insert any additional data into the stream. In these cases, Tcpflow itself is sufficient to reconstruct the file. This can be seen in Figure 6.

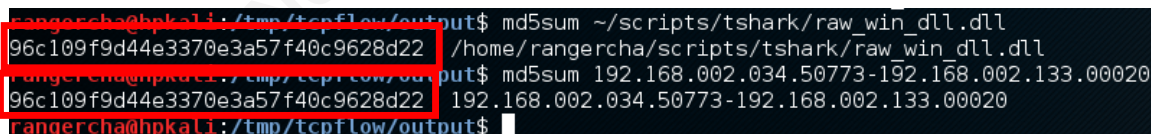


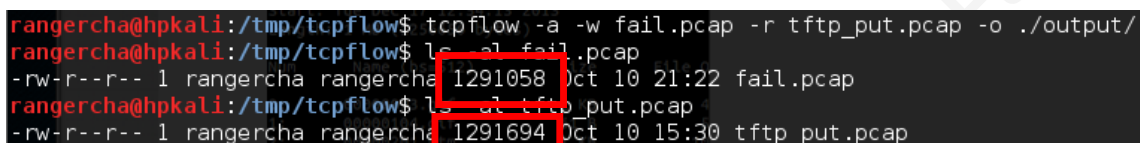
Figure 6. Comparison of the MD5 (message digest) hash of a file extracted from an FTP (file transfer protocol) data stream using Tcpflow and the actual file.

A second exception to the rule of higher level protocol parsing with Tcpflow is the HTTP protocol. When Tcpflow is invoked with the -a switch to enable all post-processing, it is capable of stripping HTTP protocol headers from reassembled streams and unzipping server responses that are compressed with gzip (Elson, n.d.). Tcpflow is able to properly process this content, so Foremost provides no additional benefit.

A second shortcoming to the approach of using Tcpflow is that it only is capable of processing TCP packets. While this may be obvious from the name of the software, it can still be an important point when analyzing UDP-based communication. For these situations, Tcpflow has the -w switch. Using this option causes any packets that cannot be properly processed by Tcpflow to be exported to a separate packet capture. This behavior can be seen in Figure 7 where files transferred using TFTP, which is a UDP-

Rebecca Deck, sdsecurityacct@hotmail.com

based protocol, are entered into the failure capture. The failure packet capture contains nearly all packets from the initial capture.



```
rangercha@hpkali:/tmp/tcpflow$ tcpflow -a -w fail.pcap -r tftp_put.pcap -o ./output/
rangercha@hpkali:/tmp/tcpflow$ ls -al fail.pcap
-rw-r--r-- 1 rangercha rangercha 1291058 Oct 10 21:22 fail.pcap
rangercha@hpkali:/tmp/tcpflow$ ls -al tftp_put.pcap
-rw-r--r-- 1 rangercha rangercha 1291694 Oct 10 15:30 tftp_put.pcap
```

Figure 7. Tcpflow placing unprocessed UDP (User Datagram Protocol) packets into a failure capture file.

Tcpflow is an effective method for extracting files from both the HTTP and FTP protocols. Unfortunately, it is incapable of working with either UDP-based protocols or the SMB protocol. The use of Foremost on the streams extracted by Tcpflow does not resolve any shortcomings of Tcpflow. While Foremost may be useful when extracting files during many forensics investigations, it does not add any value when parsing packet captures.

4.3. Tcpxtract

Tcpxtract is designed to be a file carving tool for packet captures. This is in contrast to several other tools tested, which focus on exporting streams. Tcpxtract instead reassembles streams and looks for markers that denote the beginning and end of files. In this way, it is very similar to the Foremost tool, except Tcpxtract functions on packet captures. Tcpxtract supports 26 file formats and is designed to be extensible, so as long as a file format's markers can be found, Tcpxtract can support it (Padres & Harbour, 2005).

Unfortunately, not all file formats contain clear start and end markers. A notable exception is the frequently malicious portable executable format. This format is difficult to carve, because it lacks an end of file marker. The zip file compression format is also used for a variety of other file types including Java JAR files and Android APK archives. Tcpxtract classifies all these file types are extracted as zip compressed files. These compressed files do not properly decompress and are sometimes completely missed by Tcpxtract. In these cases, other tools that extract entire streams were found to be far more reliable.

Rebecca Deck, sdsecurityacct@hotmail.com

Tcpextract only offers support for TCP packets with no support for UDP datagrams. Therefore, during testing of the TFTP protocol it was found that no files were extracted even for supported file formats. If Tcpextract is selected for performing file extraction, then analysts should ensure that packet captures do not contain any UDP data that could be material to an investigation.

Due to the lack of consistent behavior, analysts should be wary of using Tcpextract during investigations. Some files are correctly extracted, but this is not always the case. Extensions for file formats that have multiple purposes, such as compressed archives, are frequently misidentified by Tcpextract. Even with these shortcomings, Tcpextract can be of use. Since this tool functions by reassembling streams and then looking for markers to identify file boundaries, it produces different results than tools that simply extract streams. In cases where other tools fail to provide adequate results, Tcpextract can be used to search for additional files. In these instances, analysts should anticipate that the files extracted may be flawed.

4.4. Tcpextract

Tcpextract is another file extraction solution similar to Tcpflow. Although similarly named, this is not the same tool as Tcpextract. This is a command line tool that extracts all files that it can from either a packet capture or listening interface. One advantage to using Tcpextract is that it extracts files with their original names. Most other tools use an index to name extracted files, but this does little to help an analyst who may be attempting to identify files that were served from a specific site. However, this becomes one of the greatest failings of the Tcpextract tool. Long file names often cause the application to crash, even if the protocols are fully supported. One instance of this is depicted in Figure 8.

```
rangercha@hpkali: /tmp/tcpextract$ tcpextract -f bing_test_curl2.pcap
Exception in thread Thread-2:
Traceback (most recent call last):
  File "/usr/lib/python2.7/threading.py", line 810, in __bootstrap_inner
    self.run()
  File "/usr/local/lib/python2.7/dist-packages/tcpextract-1.1-py2.7.egg/TcpExtract/Parser.py", line 38, in run
    fd=open(self.directory+"/"+f[i], "rb")
IOError: [Errno 36] File name too long: './output/www.bing.com fd ls l?IG=83cf09e65af3401d947a750c3159fcef&CID=3F05F64F543A645805CF&L713580055307type=Event.CPT&DATA={%22pp%22:{%22S%22:%22L%22,%22FC%22:-1,%22BC%22:-1,%22H%22:47,%22BP%22:52,%22CT%22:55,%22IL%22:1},%22ad%22:[-1,-1,1103,746,1103,746,1],%22w3c%22:%2217fdf0,14,,6,,13,,47,1,37,,5,,1,-17%22,%22nav%22:0}%P=SERP&DA=ATAv2'
```

ypextract failing to extract a file from an HTTP (Hypertext Transfer Protocol) stream due to file name length.

Tcpextract has no way to override this behavior without modifying the application; so if a capture has even one long file name, no files can be extracted. Even more troubling than the overt extraction failures due to filename length are silent failures resulting in extraction of only a subset of the files contained in a packet capture. As can be seen when analyzing an infection from the Angler exploit kit, several files are indeed extracted. When the ngrep tool is used to view HTTP requests contained within the packet capture, it is clear that data from several requests is never extracted. This is shown in Figure 9.

```
rangercha@hpkali:~/tmp/tcpextract$ ngrep -qI 2015-06-03-Angler-EK-traffic.pcap | grep GET
GET /js/view.js HTTP/1.1..Accept: /**..Referer: [[[[[[[[[[[ redacted ]]]]]]]]]] ..Accept-Language: en-U
GET /whinnied swivel languor legibly/126044985793117700 HTTP/1.1..Accept: application/x-ms-application, imag
GET /9trkcewr0LU5WruECPQ_jJz6W0Z8TCrt4-1q0b1kmBx1GT07.aspxnet HTTP/1.1..Connection: Keep-Alive..Host: piscato
GET /20i8bgZdM1nAppbiiIzKbJzJ6gALK2d5rVLGwez8kwSUuXib.jsscript HTTP/1.1..Accept: /**..Accept-Language: en-US
GET /timezone-1.1/-54.60046/25.71254 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml
GET /stats/eurofxref/eurofxref-hist-90d.xml/746590099265aee91ccb0954b2efadb50 HTTP/1.1..Connection: Keep-Alive
GET /domain/xemuxziznq6u.com HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, *
GET /45bdee6cbdacfe5ad284221e3e867fdae HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /r.php?ts=8465c25c2d68144f02b604bf4219d79 HTTP/1.1..Accept: /**..Referer: http://calbuleheruse.com/searc
GET /7c=0c25f3555619cbffffffffff0000077d2f3512942adlad55a61c1b7ca2da7f37 HTTP/1.1..Accept: /**..Referer: htt
GET /r.php?ts=1d2509a6d8f384931e325d5f7b2139 HTTP/1.1..Accept: /**..Referer: http://kertacolser.com..Accep
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /r.php?ts=be8ade06311bfe503c09d666aabbddedb HTTP/1.1..Accept: /**..Referer: http://heddoleidenfo.com/search
GET /ads.php?sid=1917 HTTP/1.1..Connection: Keep-Alive..Accept: text/html, application/xhtmlml+xml, /**..Accep
GET /click?side=3a0d18146f210b13ff9ce57672fb58a3fc12c0a6cid=0 HTTP/1.1..Accept: /**..Referer: http://kertaco
GET /click?node=56time=1433342100&id=18161&pid=346&sid=188877&rank=0 HTTP/1.1..Accept: /**..Referer: http://cc
rangercha@hpkali:~/tmp/tcpextract$ rm -rf output/
rangercha@hpkali:~/tmp/tcpextract$ tcpextract -f 2015-06-03-Angler-EK-traffic.pcap
rangercha@hpkali:~/tmp/tcpextract$ ls output/
com.custermid.com_js_view.js
jerorefast.com_ads.php?sid=1917
jertadopoeremo.com_ads.php?sid=1917
jiuijitsukarate.com_ads.php?sid=1917
lopjertamoper.com_ads.php?sid=1917
piscatorialist.newevecollection.com_9trkcewr0LU5WruECPQ_jJz6W0Z8TCrt4-1q0b1kmBx1GT07.aspxnet
piscatorialist.newevecollection.com_whinnied swivel languor legibly/126044985793117700
www.earthtools.org_timezone-1.1_-54.60046_25.71254_-
www.ecb.europa.eu_stats_eurofxref_eurofxref-hist-90d.xml/746590099265aee91ccb0954b2efadb50
xxso.xemuxziznq6u.com_45bdee6cbdacfe5ad284221e3e867fdae
rangercha@hpkali:~/tmp/tcpextract$
```

Comparison of the number of files found by ngrep

Another shortcoming of the Tcpxtract tool is that it supports only the HTTP protocol. It silently discards packets from any other protocols. Moreover, content compressed using gzip as part of HTTP communication is not automatically decompressed. Analysts must manually complete this step after processing is complete.

Tcpxtract, while simple to use, is a dangerous tool for analysts. It can successfully extract files from packet captures, but not in a reliable and predictable fashion. It supports only the HTTP protocol, overtly fails on some file names, and silently fails to extract others. Given these limitations, it is not recommended to rely on Tcpxtract when analyzing packet captures.

4.5. Network Miner

Network Miner is a graphical packet capture analysis tool that is written for the .NET framework. Although .NET applications traditionally run only in Windows, applications such as Wine and Mono allow Network Miner to function on Linux as well. Network Miner is extremely simple to use, requiring only that an analyst open a packet capture file. Support for twenty protocols is provided including HTTP, TFTP, and FTP.

As soon as a packet capture is opened in Network Miner, several tabs are populated with various pieces of extracted information and Network Miner extracts all the files it can. Files contained within the packet capture are placed in the “AssembledFiles” folder, which is automatically created in the folder containing the Network Miner executable file. A list of the extracted files is also available on the “Files” tab (Davidoff & Ham, 2012).

Network Miner is an exceptionally easy tool to use and it is quite effective at extracting files from its supported protocols. Unfortunately, Network Miner is not easily extended to support additional protocols if it does not already support them. During testing, it was confirmed that the SMB protocol is unsupported and files transferred with this protocol are silently ignored. Additionally, the time required to load Network Miner is substantially more than other extraction tools. During testing of a moderately-sized packet capture of 42 megabytes, Network Miner took more than four times as long to extract files as comparable command-line tools.

Rebecca Deck, sdsecurityacct@hotmail.com

The ease of Network Miner makes it very attractive for new analysts. It supports file extraction from several protocols, provides large amounts of data for further analysis, and has a graphical interface. However, the abundance of information generated often adds to the amount of time required for analysis. Since every file is extracted, the analyst must then determine whether or not each file is material to the investigation. Network Miner also takes the longest time to run of any tool considered during this experiment. This makes it unsuitable for longer packet captures. Provided that packet captures are of a limited size, this is a very effective tool for extracting files.

4.6. Wireshark

Wireshark is an incredibly versatile packet analysis tool. It interprets an extremely large number of protocols and easily extracts files from the HTTP, DICOM, and SMB protocols. With some manual processing, Wireshark can also be used to extract files from other protocols.

Extraction from the HTTP, DICOM, and SMB protocols is trivial. Unlike all other applications discussed previously, Wireshark allows for files to be selectively extracted. The protocol is selected from Wireshark's pull-down menus as depicted in Figure 10.

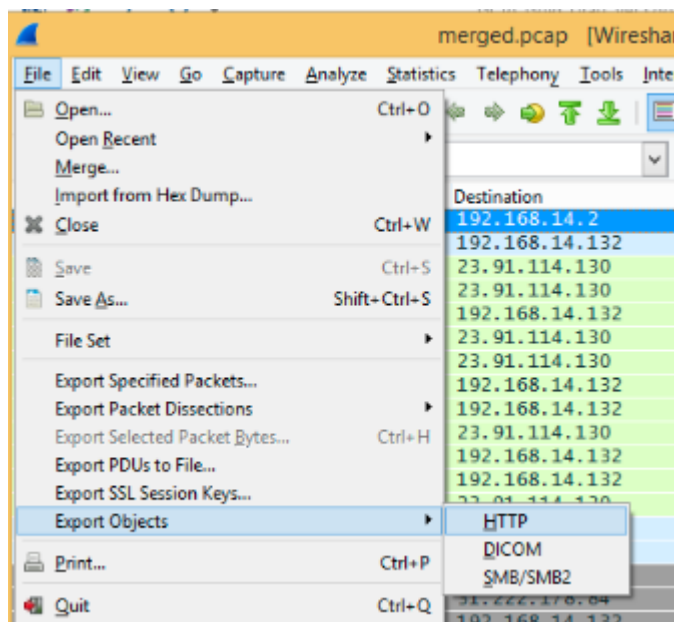


Figure 10. Location of Wireshark's "Export Objects" function.

Rebecca Deck, sdsecurityacct@hotmail.com

Wireshark then presents a list of files it has identified for extraction. An analyst can choose specific files of interest or save all files to a folder. The procedure is virtually the same for HTTP, DICOM, and SMB objects.

Although simple to use, the “Export Objects” interface of Wireshark leaves much to be desired. Wireshark’s incredibly useful display filters do not apply to this functionality. While this is immaterial in smaller packet captures, if a capture contains thousands of files it becomes extremely difficult to locate the desired objects. Another shortcoming of this interface is the inability to sort the list of objects by any field other than packet number. For HTTP requests this is the number of the packet containing the HTTP response code, as depicted in Figure 11.

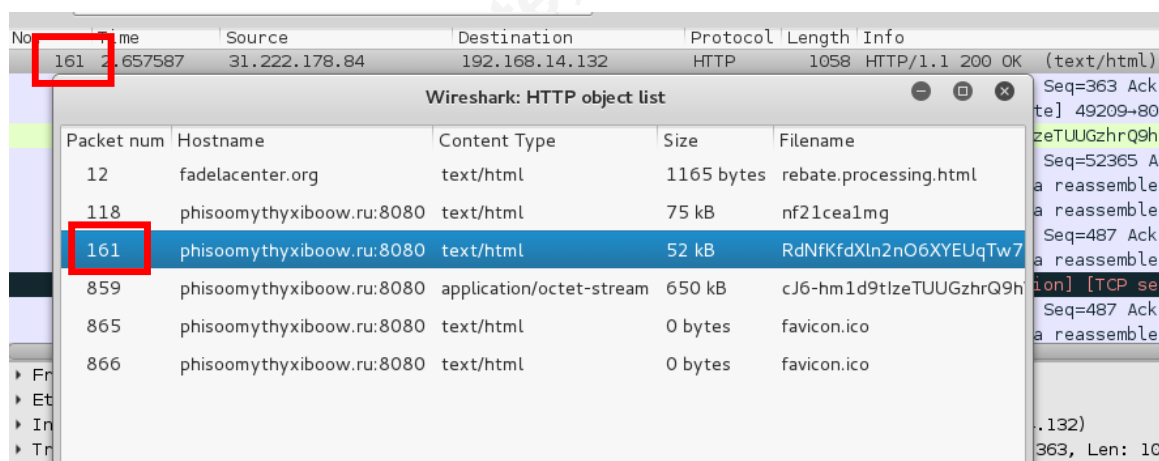


Figure 11. Wireshark’s “Export Objects” interface, showing how to locate a specific object based on packet number.

Wireshark can still be used to extract files through protocols that are not supported by the “Export Objects” function. The analyst must first determine which packets hold the desired file. Using “Follow TCP Stream” from the right-click menu as shown in Figure 12, an ASCII interpretation of the raw file is presented.

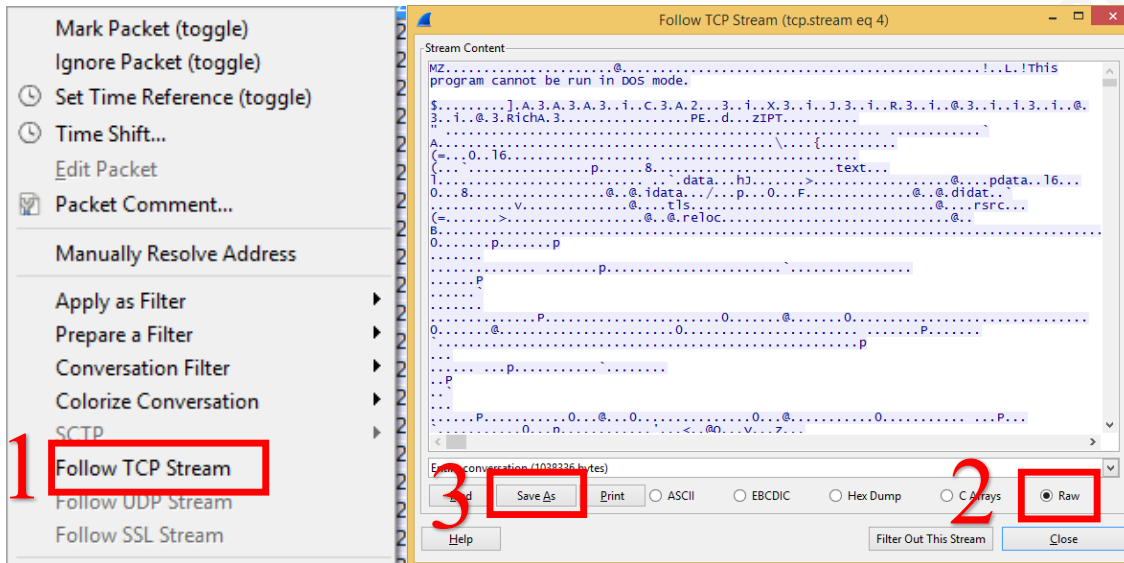


Figure 12. Extracting files from a stream in Wireshark without using “Export Objects.”

The raw file can then be saved to disk without the “Export Objects” functionality. This works well for protocols that transfer files without intermingling protocol data with the file’s raw data. Protocols such as FTP and TFTP are good candidates for this method. If protocols do insert their own data then this information must be manually removed after extraction. This is a painstaking task involving the use of a hex editor.

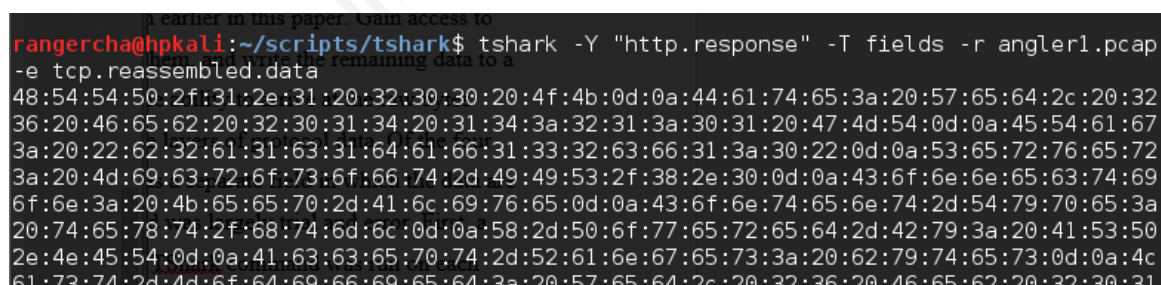
Wireshark is an extremely reliable and versatile tool. However, its file extraction abilities are unwieldy to use. Sorting through the extraction interface is time consuming and decisions often must be made with limited information. For protocols other than HTTP, DICOM, and SMB, analysts must manually write the data streams to disk. Even though these limitations exist, analysts can consistently and accurately extract files from packet captures using Wireshark.

5. TShark Extractor

TShark is a companion tool to Wireshark that works from the command line rather than a graphical interface. By using TShark it is possible to leverage Wireshark’s considerable protocol analysis capabilities in a fashion that is easily scripted. Unfortunately for those attempting to extract files from packet captures, TShark does not possess Wireshark’s file extraction capabilities. Since TShark allows access to the raw

bytes from a packet capture, it is possible to not only replicate, but also extend Wireshark's extraction functionality.

The process is the same that is set forth earlier in this paper: gain access to the raw bytes, strip the protocol information from them, and write the remaining data to a file. TShark's protocol decoding ability provides the ability to retrieve the raw bytes transferred without having to contend with multiple layers of protocol data. Each of the four protocols examined in this experiment has a separate field in which the raw bytes are stored. The process of discovering an appropriate field is largely trial and error. First, a packet capture containing the protocol with the embedded files must be selected. Then a TShark command is run on each field available within Wireshark. When a command is found that returns the desired data, usually in a hex-encoded format, then this is selected as the appropriate field for extraction. An example of the TShark command used to select the correct field for the HTTP protocol is shown in Figure 13.



```
rangercha@hpkali:~/scripts/tshark$ tshark -Y "http.response" -T fields -r angler1.pcap
-e tcp.reassembled.data
48:54:54:50:2f:31:2e:31:20:32:30:30:20:4f:4b:0d:0a:44:61:74:65:3a:20:57:65:64:2c:20:32
36:20:46:65:62:20:32:30:31:34:20:31:34:3a:32:31:3a:30:31:20:47:4d:54:0d:0a:45:54:61:67
3a:20:22:62:32:61:31:63:31:64:61:66:31:33:32:63:66:31:3a:30:22:0d:0a:53:65:72:76:65:72
3a:20:4d:69:63:72:6f:73:6f:66:74:2d:49:49:53:2f:38:2e:30:0d:0a:43:6f:6e:6e:65:63:74:69
6f:6e:3a:20:4b:65:65:70:2d:41:6c:69:76:65:0d:0a:43:6f:6e:74:65:6e:74:2d:54:79:70:65:3a
20:74:65:78:74:2f:68:74:6d:6c:0d:0a:58:2d:50:6f:77:65:72:65:64:2d:42:79:3a:20:41:53:50
2e:4e:45:54:0d:0a:41:63:63:65:70:74:2d:52:61:6e:67:65:73:3a:20:62:79:74:65:73:0d:0a:4c
61:73:74:2d:4d:6f:64:69:66:69:65:64:3a:20:57:65:64:2c:20:32:36:20:46:65:62:20:32:30:31
```

Figure 13. TShark displaying colon-separated, hex-encoded, raw bytes extracted from a packet capture.

The second task is to find fields that can be used to create a unique name for each file to be extracted from a packet capture. Each of the fields necessary to support both the file name and data are added to a list of fields. TShark is invoked to select each field in this list from a packet capture, filtering for protocols that might result in file extraction. In order to support extraction of HTTP, SMB, FTP, and TFTP the fields in Figure 14 are used.

```
#used to determine protocol
#[0]:_ws.col.Protocol
#used by HTTP
#[1]:tcp.reassembled.data
#used by HTTP and FTP
#[2]:tcp.stream
#used by SMB
#[3]:smb.fid
#[4]:smb.file_data
#used by TFTP
#[5]:data
#[6]:udp.stream
```

Figure 14. Listing of TShark fields used by the TShark Extractor script to generate unique file names and retrieve raw bytes.

This results in an unwieldy TShark command as depicted in Figure 15.

```
rangercha@hpkali:/tmp/tshark$ tshark -r merged.pcap -Y "http.content_type
contains shockwave && (http.content_length > 0 || (smb.file_data && smb.re
maining==0) || ftp-data || tftp.opcode==3)" -T fields -e _ws.col.Protocol
-e tcp.reassembled.data -e tcp.stream -e smb.fid -e smb.file_data -e data
-e udp.stream -E quote=d -E occurrence=a -E separator=|
"HTTP"| "48:54:54:50:2f:31:2e:31:20:32:30:30:20:4f:4b:0d:0a:44:61:74:65:3a:
20:46:72:69:2c:20:32:34:20:4a:75:6c:20:32:30:31:35:20:32:33:3a:31:33:3a:33
```

Figure 15. The TShark command used by the TShark Extractor script.

In reality, this command has only three parts: a filter, a set of extraction fields, and formatting for extracted information. The filter has a user-provided section, which can be any display filter supported by Wireshark, and a group of filters designed to ensure that only valid files are extracted. A separate filter is used for each file transfer protocol to ensure that blank and invalid files are not processed. The fields specified in the next segment are those denoted in Figure 14. The formatting segment ensures that each returned value is enclosed in double quotes and separated by the “|” symbol. This ensures that TShark’s output can be easily parsed.

Following the execution of TShark, data is parsed into a Python list. Based on the contents of the “_ws.col.Protocol” field a specific protocol parser is selected. Each parser has the responsibility of converting TShark’s output to a file name and raw bytes. There

Rebecca Deck, sdsecurityacct@hotmail.com

are two requirements for the file name selected. The first is that it is unique to the extraction so as not to overwrite other files that have already been extracted or mistakenly merge two separate files into one. The second requirement is that the file name should not contain the character used to separate columns in the TShark query. The file bytes are presented as colon-separated hexadecimal values enclosed in double quotes. To convert this into raw bytes using Python is a trivial process. If there is any post-processing to be done on these bytes, then this can also be handled by the parsing function. This is illustrated by the HTTP protocol parser. This parser contains HTTP protocol information and potentially has content compressed using gzip.

The FTP protocol does not have a simple method of accessing the bytes of transferred files. Instead, TShark must be rerun to extract each FTP data stream. This must be performed once per stream. This can be accomplished using TShark's "-z" argument with the "follow, tcp, raw" arguments, once per stream containing FTP data. This method is considerably slower using a specific protocol field to extract the data, as is done with the HTTP, SMB, and TFTP protocols. With these three protocols a single TShark query can extract all the files in the packet capture as opposed to having a separate pass for each stream to be extracted.

This format allows simple extension to other protocols and the addition of post-processing rules. TShark should theoretically be able to extract files from any protocol as long as TShark can decode the protocol and access the raw bytes. Leveraging TShark also has the advantage of allowing an analyst to use Wireshark display filters to select which files to extract. Use of command line tools allows a scripted approach to file extraction and aids in the development of repeatable processes. To demonstrate this process and provide another tool to augment the analyst's arsenal, the TShark Extractor script is available at https://github.com/rangercha/tshark_extractor.

6. Comparison of Tools

None of the tools tested during this experiment will successfully extract files in all situations. Analysts must be cognizant of each tool's abilities and limitations to ensure that accurate information is used during investigations. Therefore, consideration must be

Rebecca Deck, sdsecurityacct@hotmail.com

taken for the protocols contained within packet captures and each tool's support for these protocols.

Protocol support must also be considered when selecting a tool for extracting files. Analysts should first conduct a quick assessment of which protocols are contained within a packet capture to ensure the appropriate tool is used. Even if all protocols are not supported by a tool, the analyst will at least be aware of what a tool may miss. Testing of each tool with the HTTP, SMB, FTP, and TFTP protocols revealed support according to Table 1.

Table 1 File extraction support of tested tools by protocol				
Tool	HTTP	SMB	FTP	TFTP
ChaosReader	Partial			
Tcpflow	Full		Full	
Tcpextract	Partial	Partial	Partial	
Tcpextract	Partial		Full	
NetworkMiner	Full		Full	Full
Wireshark	Full	Full	Manual	Manual
TShark Extractor	Full	Full	Full	Full

Table 1. File extraction support of tested tools by protocol.

“Partial” support indicates that a tool sometimes either missed files or extracted files with incorrect contents. “Manual” support means that while the tool does not have an automated method of extracting files from this protocol, files still can be extracted with a few manual steps. Tools that always correctly extracted files from a protocol are listed as providing “Full” support.

The most dangerous behavior exhibited by tools is providing incorrect results. This can manifest as either expected files not being extracted or extracted files with incorrect contents. Tcpextract, Tcpextract, and ChaosReader all exhibit these symptoms. Use of these tools is only advised if the expected results cannot be obtained with other more reliable tools. In this case, protocol bytes may have to be

Rebecca Deck, sdsecurityacct@hotmail.com

manually removed from the extracted files. Analysts should not expect sandboxing tool results or hashes of these files to be accurate.

If an appropriate tool is selected, analysts can avoid wasting time on failed extractions or analyzing malformed files. Not included in this experiment is the analysis of protocols other than HTTP, SMB, FTP, and TFTP. Network Miner claims support for several additional protocols and is a good choice when there are no other clear options. Whenever protocols are encountered for which there is no support in any tool, analysts should be prepared to follow the steps outlined in this paper's introduction to manually retrieve the original files.

7. Conclusion

Network packet captures are a valuable tool when analyzing incidents. These captures are often the only means to obtain precise records of what has gone into and out of a system. For many common exploit kits the specific exploits used during attacks can be discovered by analyzing files that are pulled from network packet captures. A suitably positioned sensor can also give an analyst insight into the exact data that has been exfiltrated from a network during a compromise.

Standard forensics tools that are designed for use on file system images cannot be used to analyze network traffic. The structure of packet capture files adds raw packet header information and other metadata into the raw data of the files contained within a capture. Tools that are not designed for analyzing packet captures are unable to separate this protocol data from the files that were actually transmitted over the network.

Fortunately, there are many tools that can extract files from network packet captures. Unfortunately, no tool performs perfectly in all situations. Some of these applications occasionally do not accurately reconstruct all files from packet captures, even if the files are contained in a supported protocol. This makes these tools extremely risky to use, because they can cause analysts to incorrectly discard malicious files or not even have the opportunity to examine them.

Rebecca Deck, sdsecurityacct@hotmail.com

Protocol support is another area of difficulty for each tool. Of the four common file transfer protocols tested, none of the previous tools were able to process all of them in an automated fashion. Protocol coverage by tool varied according to Table 1.

Most tools do not have the ability to perform selective extraction of files. Only Wireshark possesses the ability to extract only specific files. All other tools are only capable of extracting every file in a packet capture. Analysts must then perform post-processing to determine which extracted files are material to the investigation. Wireshark is able to select specific files for extraction, albeit with a limited interface. This allows an analyst to narrow the scope of the files they must process without using several other command-line tools.

TShark Extractor allows the extraction of specific content in a file capture utilizing Wireshark display filters. With other tools an analyst must apply a filter, extract the applicable traffic to a new capture file, and then use a file extraction tool. Automating extraction from various protocols minimizes the number of different tools that must be used during analysis. Additionally, the script is short enough to demonstrate the concept of file extraction to allow analysts to easily examine the process.

The specific capabilities of a tool are secondary to an analyst's ability to understand the process used by the tool. Knowing when a tool will return accurate data saves analysts from wasting time and avoids incorrect conclusions. Even when new protocols are encountered, a skilled analyst can extend existing tools or manually reproduce the process.

8. References

- Allen, J. (2001). *The CERT Guide to System and Network Security Practices*. Upper Saddle River, NJ: Addison-Wesley.
- Allman, M. & Ostermann, S. (1999). FTP Security Considerations. RFC 2577. Retrieved 4 November, 2015, from <https://tools.ietf.org/html/rfc2577>
- Banks, D. (2013). Custom Full Packet Capture System. Retrieved 21 October, 2015, from <https://www.sans.org/reading-room/whitepapers/logging/custom-full-packet-capture-system-34177>
- Brenton, C. (2006). Egress Filtering FAQ. Retrieved 3 November, 2015, from <https://www.sans.org/reading-room/whitepapers/firewalls/egress-filtering-faq-1059>
- Bejtlich, R. (2012). *The Practice of Network Security Monitoring* [Kindle Fire HD version]. Retrieved from Amazon.com
- Davidoff, S., Ham, J. (2012). *Network Forensics: Tracking Hackers through Cyberspace*. Upper Saddle River, NJ: Prentice Hall.
- Dharmapurikar, S. & Paxson, V. (2005). Robust TCP Stream Reassembly In the Presence of Adversaries. *14th USENIX Security Symposium*. Retrieved 3 November, 2015, from https://www.usenix.org/legacy/event/sec05/tech/full_papers/dharmapurikar/dharmapurikar.pdf
- Duncan, B. (2015). Exploit Kit Roundup – early June 2015. Retrieved 21 October, 2015, from <https://isc.sans.edu/diary/Exploit+kit+roundup+-+early+June+2015/19763>
- Elson, J. & Garfinkel, S. (n.d.). Tcpflow. Retrieved 21 October, 2015, from <https://www.mankier.com/1/tcpflow#>
- Garfinkel, S. (2010). Digital Forensics Research: The Next 10 Years. *Digital Investigation*, 7. S64-S73. doi:10.1016/j.diin.2010.05.009

Rebecca Deck, sdsecurityacct@hotmail.com

Garfinkel, S. & Shick, M. (2013). Passive TCP Reconstruction and Forensic Analysis with tcpflow. Retrieved 21 October, 2015, from <http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA585499>

Gregg, B. (2004). Chaosreader. Retrieved 21 October, 2015, from <http://www.brendangregg.com/chaosreader.html>

Harris, G. (2015). Libpcap File Format. Retrieved 21 October, 2015, from <https://wiki.wireshark.org/Development/LibpcapFileFormat>

Kendall, K. & Kornblum, J. (2006). Foremost Man Page. Retrieved 21 October, 2015, from <http://foremost.sourceforge.net/foremost.html>

Padres, B. & Harbour, N. (2005). Tcpxtract Home Page. Retrieved 31 October, 2015, from <http://tcpxtract.sourceforge.net/>

Palo Alto. (2013). The Modern Malware Review. Retrieved 21 October, 2015, from <http://media.paloaltonetworks.com/documents/The-Modern-Malware-Review-March-2013.pdf>

Parkour, M. (2015). Collection of Pcap files from malware analysis. Retrieved 31 October, 2015, from <http://contagiodump.blogspot.com/2013/04/collection-of-pcap-files-from-malware.html>

Soderberg, W. (2010). Extracting Files from a Capture aka Intercepting Files. Retrieved 21 October, 2015, from <https://wh1sk3yj4ck.wordpress.com/2010/08/12/extracting-files-from-a-capture-file-aka-intercepting-files/>