FRONT PAGE FLYLEAF

ABSTRACT

Keywords: malware, automation, software, OSI, triage

This paper discusses the creation of a software tool called the Suspicious File Triage Tool, or SFTT. SFTT assists in the collection of open source intelligence related to suspicious files. The tool gathers information from the Internet and sources compiled by the user. SFTT is being created in response to the tremendous growth in the amount of malware and also its rapid evolution. It is meant to complement the other software tools used for malware analysis that exist and to free the malware analyst for tasks that require creativity and expertise. SFTT strives to prevent information leakage by not uploading files to external services.

The functionality of the core framework and some example plug-ins are laid out in detail and shown to be implementable. Plug-ins to access the external services VirusTotal and ThreatExpert and a plug-in to perform ssdeep hash comparisons against a list of file hashes are described. Some limitations of the SFTT are also described. One is issues with external services breaking or changing terms of use. Another is the possibility of information leaking even with steps taken to minimize such risks.

SFTT can be enhanced by developing new plug-ins and changing the core structure of the tool to meet new requirements. Designing the tool to be used as a web service and adding alternate output formats like PDF and XML will make the tool easier to access and its data more useful. SFTT can be modified to integrate tightly into an existing automation framework to increase efficiency. Plug-ins to use anti-virus virtual machines and malware analysis software supplied by the user address most information leakage issues.

DEVELOPING A SUSPICIOUS FILE TRIAGE TOOL

By

D. Kevin Stilwell Jr.

A Capstone Project Submitted to the Faculty of

Utica College

March 18, 2012

In Partial Fulfillment of the Requirements for the Degree

Master of Science Cybersecurity – Intelligence and Forensics

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

`

I would like to thank my family, especially my wife Allison, for their understanding and encouragement.

I would also like to thank Professor Randall K. Nichols and Professor Vernon McCandlish for their feedback, ideas, and availability. Their mentorship was instrumental is conceiving and executing this paper and any positive qualities contained within bear their mark.

STATEMENT OF THE PROBLEM

This paper will discuss the creation of a proof of concept (POC) tool called ***Suspicious File Triage Tool (SFTT)*** *which will provide automated file triage to a malware analyst.* **SFTT** will focus on gathering open source intelligence (OSI) related to suspicious files. OSI for this POC tool development includes information from the Internet and information compiled in previous investigations by the tool's user.

SFTT will be a plug-in based tool. These plug-ins will take the form of programmatic library files that implement a specific set of documented functions expected by **SFTT**. Some example plug-ins will be initially developed and discussed in this paper. These plug-ins will be of sufficient technical capability to verify the POC. It is expected that new plug-ins will be added in the future.

SFTT Design and the RegRipper Tool

The author, in creation of **SFTT,** was significantly influenced by the design of the **RegRipper** software tool. Harlan Carvey wrote **RegRipper** in 2008. **RegRipper** can be run from the command line on Windows (Carvey, 2011, p. 176) and on Linux and Mac OS X platforms with minimal modifications (Jeffbryner, 2009). It is easy to run on both 32-bit and 64-bit versions of Windows because a graphical user interface is available and the tool is provided as executable binary files for those platforms (Carvey, 2011, pp. 173-174). The creation of the executable files is accomplished using the Perl2Exe tool from IndigoStar ("Perl2Exe Home Page", n.d.). As of the writing of this paper **RegRipper** can be downloaded from the *winforensicsanalysis* Google Code page ("winforensicsanalysis

Downloads", n.d.) and user contributed plug-ins can be downloaded from the *regripperplugins* Google Code page ("regripperplugins Downloads", n.d.).

The purpose of **RegRipper** is to extract and correlate data from Windows Registry files (Carvey, 2011, pp. 174). The use of the "Parse: Win32Registry" library allows for parsing of registry files. A single plug-in or a custom list of plug-ins can be provided to **RegRipper** to be run against a registry file (Carvey, 2011, pp. 175). Information from each plug-in is output in a plain text format (Carvey, 2011, pp. 173).

The **RegRipper** tool works in a plug-in based framework. It was designed to be interchangeable and upgradeable so that new functions could be written and easily integrated with the core tool (Carvey, 2011, p. 173).

RegRipper was written in the Perl scripting language (Carvey, 2011, p. 175). **SFTT** will also be written in this language. An advantage of using Perl is the existence of the Comprehensive Perl Archive Network that provides access to thousands of modules that make adding functionality easier ("CPAN Frequently Asked Questions", n.d.).

RegRipper provides both a GUI and command line interface to its functionality (Carvey, 2011, pp. 173-174). For the initial release, **SFTT** will only provide a command line interface to access its functionality. Command line tools can easily be integrated into a larger automation command flow system to minimize mistakes and to standardize the performance of certain actions (Ayers, 2009).

One security priority of **SFTT** development is to minimize the risk of information leakage. No possible discovered malware should be released to third parties unless that is the specific intention. A specific real-world example of this issue is the RSA data breach in 2011. A great deal of information was discovered about the specifics of the attack

because the malware used in the compromise was submitted to VirusTotal (Hyponnen, 2011). None of the provided plug-ins will expose files processed by **SFTT** directly to any Internet sites. Information will be gathered based on file metadata and not by uploading the file to any services.

In addition **SFTT** will support the use of proxies. A proxy sits between two computers over a network and routes traffic between them. For the purposes of this tool a proxy would be used to hide the users IP address ("Proxy.org", n.d.). Tools like *whois* can be used to determine the owner of a particular IP address and a general idea of their geographic location ("How to use WHOIS effectively to track spammers," n.d.).

Another priority of **SFTT** development is implementing comprehensive logging of all tool activity. Providing such information to the malware analyst allows for a greater understanding of what specific actions are being performed by the tool (Ayers, 2009).

Source code will be made available under an open source license, specifically the BSD 3-Clause License. Providing **SFTT** as an open source tool will maximize the opportunities for those that might use the tool to contribute back to the functionality of the tool in the form of new plug-ins.

The Malware Problem

According to a report issued by Panda Labs (2011) more than five million new malware strains were created from July to September of 2011. Malware has begun to target mobile phones running the iOS and Android operating systems. Further, malware programs target minority computing platforms like Mac OS X laptops and desktops. Social networks like Twitter and Facebook have become significant malware vectors. A

malicious website is a website hosting or associated with hosting malware, potentially unwanted programs, or phishing sites, according to McAfee (2011). More than 6,500 new malicious websites a day were discovered in the third quarter of 2011. While spam volumes as of 2011 are down to 2007 levels the incidents of targeted spam attacks or "spear-phishing" have risen dramatically. In short, malware can be found in nearly all computing devices, the amount of malware is growing rapidly, and as one malware vector decreases in importance another one rises to take its place.

A report by Cisco (2011) details that global Internet traffic has had an eight-fold increase since 2005 and is predicted to increase an additional four-fold by 2015. A large amount of the new traffic is Internet video, which is predicted to exceed 50% of consumer Internet traffic in 2012. Many of the new devices generating this traffic are non-PC devices. If these trends continued it was predicted that devices would outnumber humans sometime before the end of 2011 (Cisco, 2011). Many of these new devices are vulnerable to malware. Internet router devices that use the MIPS operating system have been targeted by malware that can propagate itself through default passwords and can be used to perform distributed denial of service attacks (Janus, 2011). These sorts of devices are everywhere and are not generally considered as targets of malware by the general population and many system administrators (Janus, 2011).

Quantity and Complexity

This deluge of information and malware forces investigators to have to process large amounts of data in investigations (Schatz, 2007). In many cases this data can reach sizes of multiple terabytes (Schatz, 2007). This is referred to as the Quantity Problem

(Carrier, 2003; Schatz, 2007). This is further complicated by network effects, which can scatter evidence across multiple devices in disparate locations (Schatz, 2007). New sources of data from software and hardware complicate analysis by requiring new techniques for processing (Schatz, 2007). It takes significant effort and knowledge to comprehend this data. Not all computer forensic analysts can be expected to overcome such barriers (Carrier, 2003). This is referred to as the Complexity Problem (Carrier, 2003; Schatz, 2007).

Software Tools & Malware Investigation

According to Carrier (2003), a key part of addressing these challenges is through the use of software tools. Tools can help with the Quantity Problem by assisting in the filtering out of data that is already known or grouping related data into a single unit. Tools can also help with the Complexity Problem by translating data through one or more abstraction layers (Carrier, 2003).

Both the Quantity Problem and the Complexity Problem effect malware investigations. The amount of malware that exists continued to grow exponentially through 2011 and this is likely to continue through 2012 (Cordons, 2011). New malware has shown an ever-increasing amount of sophistication. As an example web malware has begun to recognize connections from anti-malware web bots like Google's and Yahoo's and will not deploy if it identifies those connections (Krebs, 2010). As in the wider field of computer forensics, software tools help to address these issues when investigating malware.

In his introduction to malware analysis Distler (2007) discusses how it is divided into two main types; static analysis and behavioral analysis. Behavioral analysis looks at what the malware does as it is installed and executed on a machine while static analysis is looking at the source code or disassembly of the malware to determine what the malware's function.

Kendall (2007) lists techniques used in static and behavioral analysis of malware in his discussion of how to perform malware analysis. Behavioral analysis of malware includes tools used to monitor file(s), process, and registry activity while testing malware to examine activity in these areas. To examine network related activity due to malware software tools can capture and analyze network traffic.

Acquiring cryptographic hashes of files allows for any modifications made to the file to be detected later and also allows for matches to be found against other files with different names and extensions. Software tools can be used to check if an executable has been compressed, obfuscated, or both and can also extract ASCII and Unicode strings from a binary for analysis. **SFTT** will be a static analysis tool that will use similar techniques in the initial plug-ins developed for it (Kendall, 2007).

The main audience for this paper includes individuals who perform malware analysis. In developing **SFTT** the malware analysis process will become more productive by automating the repetitive parts of an investigation. This will allow the malware analyst to be able to focus on work that requires human creativity and intuition (Zeltser, 2010). In addition this paper may educate malware analysts to new tools.

LITERATURE REVIEW

The purpose of this paper is to develop a proof of concept tool to support the gathering of open source intelligence related to a file, application, or anything that could be construed as malware. This open source intelligence may be gathered from the Internet or from information already compiled by the user of the tool. The tool will be called the Suspicious File Triage Tool (SFTT).

A malware analyst or incident responder would use this tool during the initial part of a malware investigation. The output from this tool will help the analyst to determine if a file is malware or not. This information will help to determine what subsequent steps should be taken in the investigation.

Goals of SFTT

When designing SFTT the first goal was to automate specific actions associated with malware investigations. In doing so, the malware analyst will be able to focus on other parts of the malware analysis process that requires human creativity and skill to perform (Zeltser, 2010).

Another important design goal for this tool is minimizing any difficulty in verification of the tool output. Comprehensive logging of all output generated and actions taken by the tool will be provided. These logs assist the malware analyst in validating the results of the tool (Ayers, 2009).

Lastly, it was important for this tool to minimize the possibility of exposing information during its use that could be detected by adversaries. Any file processed by SFTT will not be directly released to any third parties and only file metadata like its

cryptographic hash or name will be sent over the network. Supporting the use of network proxies will allow SFTT to hide its point of origin on the network.

SFTT Design

SFTT is significantly influenced by the design of the RegRipper tool. RegRipper was developed by Harlan Carvey in 2008 and extracts and correlates data from Windows Registry files (Carvey, 2011, p. 175).

One key feature of RegRipper that is relevant for SFTT is the plug-in based architecture. This makes it possible for additional functionality to be added to the core tool in the future (Carvey, 2011, p. 173). Also like RegRipper, SFTT will have a well-defined command line interface. This minimizes errors and standardizes performance by integrating the tool into a larger automation solution (Ayers, 2009). Having a command line interface also allows for easy integration regardless of how tools being interfaced with it are developed.

SFTT will also have its source code provided under an open source license. The intent of providing the code under this type of license is to increase of likelihood that users of SFTT will contribute plug-ins that provide further functionality for the tool.

Goals of Literature Review

This review will examine malware related background literature. It will then examine the current state of malware and where this tool would be useful in analyzing it. Sources related to development, functionality of the SFTT tools, and creation of automated tools (particularly those related to computer forensic analysis) will also be

explored. In addition the legal implications and issues with using automated tools will be examined.

It should be noted that many of the sources discussed in this review are web log posts and white papers. Generally these types of sources are considered less reliable than academic and printed resources due to a higher instance of bias and a lower barrier of entry to present information, among other factors (Driscoll & Brizee, 2010). Despite the disadvantages, these resources will be used in this paper. During this review it was noted that authors, like Harlan Carvey (Carvey, n.d.) and Lenny Zeltser (Zeltser, n.d.) maintained blogs where they discussed cutting edge development of tools and techniques related to the fields of computer forensics and malware analysis. Information can be found in these references that do not exist in academic sources.

SFTT and Open Source

What is open source? In 1998 the Open Source Definition was created by the Open Source Initiative (Raymond, 2004). The notion of freely sharing code along with an application had existed for a long time before this, but the ideas and goals of doing so had not been written into a standard. Doing so allowed the concept to be protected from intentional or unintentional misinterpretation. According to the Open Source Definition for a program to be considered open source its license has to meet certain criteria.

Open Source Definition criteria. An open source license has to allow free redistribution. This means that the license cannot prevent software from being distributed in a bundle of other software and cannot require a royalty or fee for distribution. It also

has to make the source code available. The available source code cannot make understanding or modification of it difficult ("The Open Source Definition", n.d.).

The creation of new works using source code licensed under an open source license must be allowed and the open source license cannot discriminate against any groups or professions, such as genetic research. Additional licenses not distributed with the software cannot be required. The license cannot restrict use of source code outside of a particular product or distribution and it cannot depend or insist on a particular technology or interface, such as a GUI popup to accept the license ("The Open Source Definition", n.d.).

SFTT license. SFTT will be licensed under the BSD 3-Clause License. This license allows modification, redistribution, and usage of the source and binary of the program. The license must be made available with any source code or binary distribution using the licensed code and the copyright holders and any contributors' names cannot be used in endorsement of products made using the code unless explicit permission is given (Open Source Initiative, n.d.). This license was chosen to maximize the freedom to use and extend the code while also protecting the reputation of those who write and contribute to the code.

Why use an open source license? One reason that SFTT is being offered under an open source license rather than as a commercial product is centered on the development goal for the tool's use. Many of the planned and potential plug-ins for SFTT will interact with some sort of external application programming interface on the Internet. These APIs have policies concerning commercial usage that differ from non-commercial usage. For example the public API of VirusTotal is not intended for commercial use. A

private API is available, but that requires negotiation with VirusTotal to determine pricing ("Virustotal FAQ", n.d.).

Another reason to use an open source license is to allow the tool creator and those who contribute to build a reputation in the field of malware analysis and forensics. Releasing open tools to a community of interest can have a positive impact, such as opening new employment avenues for contributors (Raymond, 2001). The creator of RegRipper, Harlan Carvey, has benefited from his contributions to the computer forensics community through name recognition. It is also possible to profit from payments from third parties to write or modify the tool (Raymond, 2001).

Open source tools also allow money to be made through tool support. Request Tracker is an issue tracking system that is made available under an open source license. The company that developed it, Best Practical, provides services such as support and custom modification of the software at a price ("Best Practical Services", n.d.).

Malware

What is malware? According to Skoudis & Zeltser (2004) malware is malicious code from an attacker that runs on a computing device. This code acts beyond the control of the device's user to perform actions beneficial to the attacker. Some software may be considered malware depending on its usage, such as a remote access tool that could be used legitimately or maliciously (O'Connor, 2010).

How is malware installed? The Organization for Economic Cooperation and Development (OECD) (2008) in their report on malware describe ways that malware can be installed on a particular computer. A user could be fooled into clicking a link in an

email or instant message that installs malware. It is also possible that security flaws in an operating system or other installed software could allow malware to be installed without direct user interaction.

Computers in both government and business are under attack constantly. Many commercial institutions experience over fifty thousand attacks per day and in 2008 the Pentagon experienced six million attempted intrusions in a twenty-four hour period (O'Connor, 2010). Any successful attack could install malicious software.

Table 1 ***Types of Malware***

Type of Malware	**Important Characteristics**
Virus	Infects a particular file on a device. Requires manual intervention to replicate on a device such as launching an executable file
Worm	Can spread across a network without manual intervention.
Trojan horse	Appears to be a legitimate application such as a game but actually does something malicious.
Rootkit	User-level rootkits replace or modify standard system executables to hide malicious activity or create backdoors. Kernel-level rootkits modify the core of the operating system (kernel) for the same purpose.
Malicious Mobile Code	Smalls programs that are downloaded from remote locations and execute without the permission of the user.

Skoudis, E. & Zeltser, L. (2004). Malware: Fighting Malicious Code

Types of malware. Malware is classified into different types based on its characteristics. Skoudis & Zeltser (2004) classify malware as viruses, worms, Trojan

horses, malicious mobile code, or rootkits. Information about these classifications can be found in Table 1 on page 12. Note that malware can be combined to perform a certain task or to increase its effectiveness. A Trojan horse could have an embedded rootkit that is deployed when it is executed by a user.

The OECD report talks about a type of malware called a bot. A bot is a type of malware that is generally deployed to a target computer via one of the other types of malware and is designed to participate in concert with a larger group of computers. This larger group of computers is called a botnet. Botnets allow an attacker to easily control a large group of devices, often for malicious purposes.

Uses of Malware. OECD describes several ways in which malware is used by an attacker. Malware often steals and transmits information from a device to the attacker. This information can include personal data like credit card numbers and passwords. Malware targeted at a specific corporation or government may steal classified or proprietary information. Botnets are often used to commit distributed denial of service attacks in which bots send queries to a target and take the target off the network for an extended period of time.

The Growing Threat of Malware

The growth of malware. The amount of observed malware has grown at an incredible rate. According to Panda Labs (2011) 34% of all malware that has ever existed was created and classified in 2010. This amounts to over twenty million pieces. While in the past malware was active for several months or even years now more than half of new

malware is active for only twenty four hours. Attackers are constantly modifying malware or creating new variations to avoid detection.

The Impact of Malware. A survey of computer users reveals that 89% run security software to protect their computers against malware (G Data, 2011). This shows a high level of awareness to the threat of malware, but despite this awareness many computers have been compromised. According to the Anti-Phishing Working Group (2009), of more than twenty two million computers scanned by Panda Anti-virus Cloud Scanning service, over 48% had some sort of malware present.

Malware also remains a significant threat to businesses. Of security professionals surveyed, 23% reported that their organization had experienced a security breach between April 2009 and April 2010 (Davis, 2010). Of the security professionals that reported a security breach, 86% indicated that it involved some type of malware.

Malware in the News

Some stories of malware incidents help to illustrate how malware is impacting businesses, governments, and regular users.

Operation Aurora: In early 2010 Google reported a targeted attack that affected nearly thirty large multinational companies. It was called Operation Aurora due to the string "aurora" being found in one of the malware Trojans involved in the attack (Panda Labs, 2010). Those targeted were high-level management who received emails inviting them to click a web link that deployed the malware. The malware took advantage of zero-day exploits for the Windows Explorer web browser (Panda Labs, 2010).

The attacks appear to have been targeting corporate intellectual property, especially source code repositories (McAfee, 2010). Google claimed that China was behind the attack due to one of the source servers connected to by the malware being in China. The Chinese government denied these accusations (Panda Labs, 2010).

Stuxnet: In July of 2010 a new worm was discovered that infected computers through USB devices (Panda Labs, 2010). This is not an unusual method of infection, but this worm also used four zero-day exploits in the course of its operation, which was unprecedented. This malware did nothing on infected machines except propagate itself, unless the machine had a Siemens programmable logic controller (PLC) installed, in which case the malware used a vulnerability in the PLC to read and write to it (Panda Labs, 2010). It was named Stuxnet due to keywords found in the code (The Economist, 2010).

The Stuxnet malware infection rate appeared to be particularly high in Iran and several nuclear facilities in Iran use the Siemens controllers (Panda Labs, 2010). The Natanz uranium enrichment centrifuges could have been targeted by Stuxnet by altering the speeds of the centrifuges, possibly causing damage. The sophistication of Stuxnet has led to speculation that a government was behind the creation of the malware (Langner, 2010).

Koobface. Since 2008 the Koobface malware worm has been a significant issue for computer users. The Koobface name is an anagram of Facebook and this malware spreads primarily through social networks, particularly Facebook (Villeneuve, 2010). This was accomplished through links sent from infected users to "friends" in the social network. These links, if clicked on, redirected to a webpage that encouraged the user to

install some malicious executable (Villeneuve, 2010). In many cases the webpage would claim to be a flash video site and the executable would claim to be a plug-in that required watching a video.

Koobface malware causes the infected machine to become part of a botnet. The operators of this botnet had infected users enter in the text for CAPTCHAs to automate the creation of fake websites that helped propagate the malware (Villeneuve, 2010). Koobface would also steal user credentials and allowed its controllers to collect revenue when users of infected machines clicked ads or installed fake security software.

Malware Analysis

Given the quantity and impact of malware that is encountered by private citizens, businesses, and governments; malware analysis has become an important discipline. Malware analysis is the examination of malware in all its forms. Those who perform this job are sometimes called malware analysts or reverse engineers ("Malware Analyst – Job Description", n.d.).

At an anti-virus or network intrusion detection company malware analysis is a full-time job. Other organizations that are not security focused may still hire full-time malware analysts as part of a network security team within the organization ("Malware Analyst – Job Description", n.d.). Skills in the area of malware analysis are also valuable in organizations that do not have the resources to hire someone full-time, but require personnel to perform malware investigation when the need arises ("Malware Analyst – Job Description", n.d.).

As discussed previously, security professionals surveyed indicated that nearly 90% of security breaches involved some sort of malware (G Data, 2011). Malware analysis is often required to understand the details and severity of a security breach. This also extends to determining the best response to a breach and identifying the perpetrators (Zeltser, 2010).

Malware analysis software and techniques are categorized as either static analysis or dynamic analysis (Distler, 2007).

Dynamic Analysis

At a high level, dynamic analysis is determining what happens when a sample is executed. Details of interest include what kind of network traffic is generated and what files are added, modified, or deleted on the system. Dynamic analysis is sometimes referred to as behavioral analysis (Hutcheson, 2006).

Dynamic analysis preparation. It is possible, and in some cases necessary, to perform analysis of malware as it is being executed on a victim machine and is capable of connecting to the attacker's servers on the Internet. However, there are significant risks associated with this approach. These include allowing the compromised machine to continue to propagate or perform malicious actions and communicate with the attacker (Kendall, 2007).

Consequently it is ideal to perform dynamic analysis in some sort of controlled environment. Any machines involved with dynamic analysis should be isolated from both other machines on the local network and the Internet. This can be accomplished through configuration or through having malware analysis machines on a physically separate

network (Brunner et. al., 2010). Network services can be simulated to allow for the malware believing it is connected to a live network. Multiple machines can be used to test malware (Kendall, 2007). Backup images of these machines allow them to be reverted to a clean state when malware testing is completed. In many cases virtual machine software is used either, instead of, or in addition to physical machines (Kendall, 2007).

Using virtual machines in dynamic analysis. Distler (2007) discusses the advantages of using virtual machines in malware investigations. Several virtual machines can be run on a single physical machine, which reduces costs. Virtual machine solutions offer the ability to take snapshots of system state, which can be rolled forward and back to save considerable setup time between tests. Virtual machines can also easily be isolated from each other and from the external network.

There are some drawbacks to using virtual machines. The theoretical possibility of malware exploiting virtual machine software vulnerabilities and escaping to the physical operating system makes using virtual machines slightly less safe than just using physical machines, although this risk can be mitigated by isolating the virtual machine host on the same network as physical machine malware testing (Distler, 2007).

Static Analysis

In general static analysis is any analysis that can be performed on a piece of malware without running it. Specifically there are two main areas of static analysis. One is viewing any source code or scripts of the malware to gain an understanding of malware function (Hutcheson, 2006). In most cases the source code will not be available and binary files will have to be disassembled and run through a debugger (Distler, 2007). This

part of static analysis is sometimes called code analysis (Hutcheson, 2006). The other area of static analysis involves examination of the malware beyond its code or disassembly. This includes details such as malware location and the size of the executable and associated files. This part of static analysis is sometimes called visual analysis (Hutcheson, 2006).

Static analysis is often the first type of analysis performed in a malware investigation. One reason for this is that many types of static analysis are quicker to perform than dynamic analysis and require less preparation. Another reason is that static analysis is considered to be safer than dynamic analysis. Performing static analysis on a different operating system than that of which the malware was intended (such as examining Windows malware on a non-Windows operating system) further minimizes the risk by greatly reducing the chances of accidentally executing the examined malware (Kendall, 2007). Note that there have been cases of vulnerabilities found in static analysis tools that could allow specifically crafted malicious files to install malware on a system or lock the tool up ("Secunia Advisory SA43910", 2011.) No malware analysis should be considered completely safe.

Cryptographic Hashes

Generating cryptographic hashes of malware files is done as part of static analysis. These hashes should be those most commonly used by other malware analysts, which currently are MD5, SHA1, and SHA256 (Kendall, 2007). Cryptographic hashes can be used to verify file integrity or necessarily modified afterwards. Taking hashes of

tools allow for file integrity verification. Cryptographic hashes also allow for file samples to be uniquely identified (Kendall, 2007).

Cryptographic hashing tools. There are several tools that can be used to generate cryptographic hashes. A selection of these tools will be described below.

To generate a hash on a single file, it is the author's experience that the simplest tools to use are from the GNU Coreutils family of tools. The names of these tools are md5sum, sha1sum, and sha256sum ("GNU Coreutils," n.d.). These will generate a MD5, SHA1, or SHA256 cryptographic hash of a provided file, respectively. These tools are generally available by default on Linux systems and can be installed on other operating systems. Unless scripted using a programming language, generating a hash list of a list of files or of files in nested directories will require some effort (Altheide & Carvey, 2011, p. 57).

There are tools developed by Jesse Kornblum with additional capabilities beyond generating a hash for a single file. The tools md5deep, sha1deep, and sha256deep will iterate through provided directories and generate a list of MD5, SHA1 or SHA256 hashes, respectively (Altheide & Carvey, 2011, p.57). The tool hashdeep processes directories similar to the above tools. It also adds the ability to generate multiple types of hashes at the same time and to audit a generated set of hash data. Once a set of hash data has been initially generated it is possible to run hashdeep again with that data and find out if files have been added, removed, moved, changed, or remain the same (Altheide & Carvey, 2011, p. 57).

Kornblum (2006) also created a tool called ssdeep that implements context triggered piecewise hashing. This type of hashing allows files to be associated even when there are

only minor differences between them and is a useful tool for determining whether two malware files are variants of one another.

Table 2 ***AV Comparatives Test Results***

	On-Demand Test February 2011	**Retrospective Test February** 2011	**Performance Test (Suite)** July 2011	**Whole Product Dynamic Test Part 1** (March-June 2011)	**On-Demand Test** August 2011	**Retrospective Test** August 2011	**Performance Test (AV)** November 2011	**Removal Test** November 2011	**Whole Product Dynamic Test Part 2** (August-November 2011)
Avast!	ADV	**N/A**	ADV+	ADV+	ADV+	ADV	ADV+	STD	STD
AVG	STD	**N/A**	ADV+	STD	ADV	**N/A**	ADV+	STD	ADV
AVIRA	ADV+	ADV+	ADV+	ADV	ADV+	ADV+	ADV+	ADV	ADV
Bitdefender	ADV+	ADV	ADV	ADV+	ADV+	ADV+	ADV	ADV+	ADV+
eScan	ADV+	ADV	N/A	N/A	ADV	ADV	ADV	N/A	N/A
ESET NOD32	ADV	ADV	ADV+	ADV+	ADV+	ADV+	ADV+	STD	ADV
F-Secure	ADV+	ADV	ADV+	ADV+	ADV+	ADV+	ADV+	STD	ADV+
G DATA	ADV	ADV	ADV	ADV+	ADV+	ADV+	ADV	STD	ADV+
K7		**N/A**	ADV+			**N/A**	ADV+	STD	STD
Kaspersky	ADV+	ADV+	ADV+	ADV+	ADV+	ADV+	ADV+	ADV+	ADV+
McAfee	ADV+	**N/A**	ADV		ADV+	**N/A**	ADV	STD	ADV
Microsoft	ADV	ADV	N/A	N/A	ADV	ADV	ADV+	ADV	N/A
Panda	ADV	ADV	ADV+	ADV+	ADV+	ADV	ADV+	STD	ADV
PC Tools	STD	**N/A**	STD			**N/A**	STD	ADV+	
Oihoo 360	STD		ADV	ADV	ADV	ADV	STD	STD	ADV+
Sophos	ADV	STD	ADV+	STD	**N/A**	**N/A**	ADV+	STD	STD
Symantec	ADV	**N/A**	ADV+	ADV	ADV	**N/A**	ADV+	ADV+	ADV+
Trend Micro	STD	**N/A**	ADV	ADV+	ADV+	**N/A**	ADV	ADV	ADV
TrustPort	ADV+	ADV	N/A	N/A	ADV	ADV	STD	N/A	N/A
Webroot		**N/A**	STD		**N/A**	**N/A**	ADV+	ADV	

Key	
ADV+	ADVANCED+
ADV	ADVANCED
STD	STANDARD
Grey	TESTED
Grey+N/A	Vendor refused to get evaluated
Black+N/A	Vendor did not take part

AV Comparatives (2011). Summary Report 2011

Virus Scanning

It is possible that the malware being analyzed is detected by anti-virus software. Scanning files against as many anti-virus tools as possible can be useful, since some malware may be detected by only one or two tools (Distler, 2007). Using online virus scanning services such as VirusTotal provides an easier way of scanning against multiple anti-virus solutions (Kendall, 2007).

Virus scanning tools. The not-for-profit AV Comparatives (2011) provides a comparison of different anti-virus solutions annually and their list represents the most commonly used and known anti-virus vendors. Table 2 on page 22 provides a list of all the products compared along with test results.

The primary method in which anti-virus software detects malware is through signature checking. It identifies malware samples through matching them against a database of known malware signatures, which can be cryptographic hashes or algorithms. This is the fastest and most accurate detection, but has the disadvantage that unknown malware will not be detected (Castelli, 2001).

Another method that anti-virus software can detect malware is through heuristic analysis. Heuristic analysis covers a wide range of techniques that go beyond signature matching (Castelli, 2001). It is intended to cover situations where the malware is new enough that no signature exists. The major disadvantage of heuristics is the occurrence of false positives, which is when heuristic analysis matches a file that is not malware (Castelli, 2001).

Static heuristic analysis uses generic signatures that match particular routines or subroutines commonly found in malware. Dynamic heuristic analysis works by running

the code in some sort of virtual machine to examine behavior and check for indicators that a file might be malware (Castelli, 2001).

Malware Analysis Workflow

It is important that an organization have a plan in place for how to respond to computer security incidents. Having a definitive plan helps to mitigate damage and decreases costs associated with an incident. It also helps to keep those involved from going off track and reduces the chances that an important action will be overlooked (Distler, 2007). Generally malware analysis is part of identifying what has caused an incident to occur (Distler, 2007).

A typical malware analysis workflow. When a suspicious file is received for analysis, a copy of the file is archived for safekeeping and a ticket is opened in some sort of tracking system. Cryptographic hashes of the suspicious file are generated and used to determine if any similarity or matches exist with previous known malware files (Bartolomie, 2011).

Some basic static analysis is then performed on the suspicious file. This includes examining header information, determining if the file is compressed or encrypted, and examining strings in the file. Note that even someone who is not proficient at malware analysis could perform all steps up to this point. At the end of this basic analysis it will be determined whether to end the investigation (the file may not be malware) or perform more advanced analysis (Bartolomie, 2011).

If file examination continues dynamic analysis is performed on either a virtual or physical machine. File activity and artifacts generated by the file are examined. Code

analysis is then performed against the file. This includes overcoming any obfuscation or encryption present and then analyzing the disassembly of the file (Bartolomie, 2011).

If the file is determined to be malware and is not detected by the main anti-virus in use by the organization it will be submitted to that vendor. The malware is not submitted to the anti-virus software vendor if it is determined to be targeted malware. Information is entered into the tracking system mentioned above and the ticket is closed (Bartolomie, 2011).

A visual chart of the described malware analysis workflow can be seen in Figure 1 on page 26. Use of this workflow along with associated automation is estimated to save thirty to sixty minutes per malware file analyzed (Bartolomie, 2011).

Figure 1 ***Sample Malware Analysis Workflow***

J. Bartolomie (2011). Operational Security Operations

Malware Anti-Analysis Strategies.

Ollman (2011) discusses advanced automation of malware analysis, including the use of appliances that automatically perform static and behavioral analysis such as signature checking and virtualized execution. Several strategies are used by malware authors to make this analysis difficult.

If the IP address of the malware investigator's network is known to the attackers they could simple ignore requests from that network which would prevent further intelligence from being gathered about the malware based on network traffic. An IP address may become known to attacker simply because considerably more traffic is seen from it than normal (Ollman, 2010).

Malware may make isolated analysis difficult by simply checking to see if sites like yahoo.com or google.com can be connected to before running or requiring that additional components be downloaded from another site before running. Malware may also look for elements in the environment, like installed software that may be present on standard machines, but not on malware testing machines, and may not run if they do not exist.

Ollman (2011) also discusses how virus upload sites share uploaded files with multiple anti-virus vendors as malware samples. In the case of VirusTotal submission files are shared to participating anti-virus vendors if their product does not detect the file and at least one other product does ("VirusTotal FAQ", 2011.) Reports on files that VirusTotal has scanned can be searched for by MD5, SHA1, or SHA256 hash value ("VirusTotal Search", 2011.)

Specific details about the malware used in the RSA hack in 2011 were released to the public by F-Secure due to the malware having been submitted to VirusTotal by an employee of RSA about two weeks after the breach occurred (Hypponen, 2011). This illustrates the risk that uploading files to VirusTotal could reveal to an attacker that their malware has been discovered in the case of a targeted attack (MLE, 2011). Malware authors are also aware of these risks and use services to test new malware against multiple anti-virus, anti-spyware, and firewall products. These services advertise that they do not retain records of files scanned and do not share any information with anti-virus companies (Krebs, 2009).

Malware Analysis Tools and the Daubert Test

Malware analysis is often part of a computer forensic investigation, many of which end up in a court of law. As the output of tools used in malware analysis may be submitted as evidence in these situations it is important that they meet the legal requirements that are expected of all accepted scientific evidence (Carrier, 2003).

In the legal system evidence from malware analysis tools is evaluated under the same standard as that used for any scientific evidence. This standard is often referred to as the Daubert Test (Carrier, 2003). The Daubert Test consists of four categories to evaluate the scientific evidence (the software tool output). The judge uses these categories before the trial begins to determine whether or not the evidence generated meets the standards (Carrier, 2003). The four categories are discussed below.

Testing. It is important to determine whether a software tool produces accurate results. This is done through performing false positive and false negative tests (Carrier, 2003).

False negative tests are intended to verify that a tool provides all possible data for a given input. This is generally the easiest type of test. One way to do so is to introduce data and verify that the tool can acquire the data (Carrier, 2003). False positive tests are intended to verify that a tool does not add data to its output. One way this can be done is by comparing the output against the output of another tool (Carrier, 2003).

Some progress has been made towards creating a standardized test for forensic tools. The National Institute of Standards and Technology with a National Institute of Justice grant created a group dedicated to computer forensic tool testing. This group has published evaluations of several different software and hardware tools and has provided standards for the testing of several types of tools (National Institute of Justice, 2010). However, several tools have not been tested, tool creation is ongoing, and not all categories of tools have standards yet.

Carrier (2003) asserts that the best way to test tools used in computer forensics is by implementing an open method. It is important to understand how a tool works beyond a generic set of test procedures for a type of tool. Tools, in which the source code is available, such as open source tools, are easier to evaluate since the code can be reviewed and tests can be designed based on a greater understanding of the software's execution. Carrier believes than even closed source tools should be required to provide at minimum design specifications to a trusted third party to allow the tool to be more effectively tested.

Error rate. Software tools generally process input data through a series of rules. The error rate of a tool is based on how well the tool implements and follows those rules (Carrier, 2003).

Carrier (2002) identifies two categories of errors that are of interest; abstraction errors and tool implementation errors. Tool implementation errors come from programming or tool design issues. These can be subtle. It could be that a tool correctly follows a specification for the information processed, but what generated the information does not. Abstraction errors come from making decisions when adding an abstraction layer to data. An example of this type of error is an intrusion detection system that shows a network attack by aggregating network packets. Some error is possibly introduced when the intrusion detection system does not know with 100% accuracy which network packets are part of the attack.

Carrier (2002) recommends that the error rate be based on number and severity of bugs found in a given time period. This is difficult to implement with a closed source application, since errors are not necessarily disclosed. With an open source application it is easier since even if bugs are not disclosed a comparison of source code between releases can reveal bug fixes.

Publication. This requirement is meant to verify that a tool has been documented in some public fashion and has been peer reviewed. This is the weakest area for most forensic and malware analysis software tools. Very few tools that can be considered published have specific technical details of the tools operation in these publications (Carrier, 2003).

Based on guidance provided by the Federal Bureau of Investigation for use of imaging technologies in the justice system (1999) software developers should be willing to release source code of tools when they are used in the generation of evidence. This is obviously not an issue with open source software. Both open and closed source software should also contain the details of the operation of the tool documented in a form other than just source code (Carrier, 2003).

Acceptance. According to Carrier (2003) many closed source tool providers assert that the number of users of their particular tool signifies acceptance. The problem with this assertion is that if choices are limited for a type of tool use of the tool has less to do with the tool being known valid and more to do with other factors like ease of use.

In-depth technical details about a tool should be made available so the user community can choose based on those details. Open source tools provide source code, which allows them to be accepted or rejected based on their technical details (Carrier, 2003).

SFTT Alternatives

Counterintelligence and controlling information. Counterintelligence is any activities undertaken against enemy intelligence operations. This can include actions such as identifying that the intelligence operations exist and then deceiving or disrupting them (Department of Defense, 2010). Enemies can include foreign governments, commercial interests, and individual actors (O'Connor, 2011).

An important part of counterintelligence is the protection of friendly assets. This includes activities that identify and neutralize the intention of adversaries and denying

adversaries access to information that might be used against an organization (U.S. Marine Corps, 2000).

Online file scanning services. There are several public services available that can be used to analyze files that may or may not be malware. One type of service scans uploaded files against multiple anti-virus solutions. Some examples of these include VirusTotal, Jotti, and NoVirusThanks. Another type of service runs uploaded files in a controlled environment and records the behavior of the executed file. This includes data like changes made and network traffic generated. Some examples of these include ThreatExpert, CWSandbox, ANUBIS, and JoeBox (Adair, Hale Ligh, Hartstein & Richard, 2011).

Unintentional information disclosure examples. In October of 2011 a hacker group in Germany called the Chaos Computer Club (CCC) released details on a piece of Trojan software that was being used by the German government in criminal investigations (Sullivan, 2011).

A German company called Digitask is the developer and according to F-Secure (2011) they submitted the installer for the malware multiple times to VirusTotal, presumably to determine what anti-virus solutions would detect it. This allowed for additional details to be determined about the malware beyond what was provided by the CCC such as the identity of the installer. The anti-virus company Kaspersky determined additional details from the submissions, such as the existence of a 32 and 64-bit version (Werner, 2011). Most malware authors are aware of this issue and use services that are for-pay only and advertise that they do not submit their files to anti-virus vendors (Krebs, 2010).

In August 2011, additional details about the RSA hack that occurred in March 2011 were released by F-Secure. While RSA provided some details about the breach F-Secure was able to determine the full scope of what happened and provided additional information. This occurred because an employee at EMC (the parent company of RSA) submitted the file to VirusTotal around the time the breach occurred (Hyponnen, 2011). It is likely that RSA would have preferred these details not be released to the public and could have avoided it by not using Virustotal.

Disadvantages of online file scanning services. Submitting files to online services for analysis carries a significant risk of exposing information to other parties that may not be realized by those who do it. Files submitted to these sites are often made available to anti-virus and security companies. Both VirusTotal and NoVirusThanks indicate in their policies that this is done ("VirusTotal FAQ," n.d. "; NoVirusThanks Terms", 2010.) Even if the service does not share submitted files generated reports can often be searched. ThreatExpert allows for generated reports to be searched by text string or by cryptographic hash. If a targeted malware is submitted the author of said malware could find out by searching for the hash value using the ThreatExpert site or Google. SFTT avoids these issues by only using hash values of files with these services and not submitting the files themselves.

Conclusions

SFTT will provide the capacity to use open source intelligence gathered by a malware analyst and through the Internet to investigate suspicious files. Using these sources allows for both dynamic and static analysis information to be gathered with a

minimum of manual effort, allowing the malware analyst to focus on tasks that require creativity. It will accomplish this while minimizing exposure of information about the user by supporting the use of proxies and not releasing files to third party services. SFTT will support a plug-in based system for adding capabilities to the tool and encourage contributions of additional plug-ins through an open source license.

In the next section, the discussion of the findings, SFTT, and its features will be presented in the context of this literature review. The design and initial plug-ins for SFTT will also be discussed. Lastly, limitations and issues with the SFTT tool will be mentioned.

DISCUSSION OF THE FINDINGS

Problem Synopsis

There has been tremendous growth in the amount of malware. In 2010 more than one third of all malware in existence was created (Panda Labs, 2010).

Malware continues to change and evolve at a rapid pace. For less than four thousand U.S. dollars the ZeuS Trojan kit can be purchased in the criminal underground. As of version 1.4 it supports the ability to re-encrypt itself and alter file names for each infection, making automated detection very difficult (Stevens & Jackson, 2010). The availability of off-the-shelf malware with this type of *capability means that over half of new malware exists for no more than 24 hours (Panda Labs, 2010).*

Malware is also adapting to other types of devices beyond personal computers. The number of networked devices was predicted by Cisco (2011) to exceed the number of people by the end of 2011. These devices include smartphones. Malware has begun to target phones running the iOS and Android operating systems. A variant of the ZeuS Trojan called Zitmo can infect Android phones and access user bank accounts to intercept one-time transmission passwords. This allows the owner of the malware to run transactions against those bank accounts (Panda Labs, 2011).

The quantity of devices and data that could potentially be involved in a malware investigation can make analysis very difficult. This is referred to as the Quantity Problem (Carrier, 2003; Schatz, 2007). As new types of malware and obfuscation techniques are created new techniques may need to be developed to process this data (Schatz, 2007). It can require a significant amount of effort and knowledge to comprehend this new data

(Carrier, 2003). This is referred to as the Complexity Problem (Carrier, 2003; Schatz, 2007).

Literature Review Synopsis

The use of an open source software license for SFTT has already been discussed. The principles that make a software license open source were enumerated. A description of the BSD 3-Clause License, which was chosen for SFTT, was given. Key reasons for using the BSD 3-Clause License and for using an open source license were described.

The literature review provided background on malware. This included an expanded definition of malware and types of malware that are known to exist. Details as to the extent to which malware influence the behavior of individuals, businesses, and governments were provided. Specific examples of malware such as Stuxnet were discussed to provide real-world examples to illustrate the impact of malware.

The function and processes of malware analysis were identified. This included a description of what malware analysis is and who might perform it. Malware analysis was discussed within the purview of computer security. Its importance was emphasized in the resolution and understanding of most computer security incidents.

The next topic discussed was the static analysis of malware. Static analysis is defined as any analysis performed on malware that does not involve actually running it digitally. Cryptographic hashes were discussed, along with common tools used to generate them and the different types of cryptographic hashes that exist. Virus scanning methodologies were also discussed. The use of virus scanning tools was described along

with specific types of functionality that virus-scanning tools have, such as signature checking and heuristic checking.

Dynamic analysis of malware was also covered. It is defined as running malware, observing its behavior, and how it modifies the compromised system. The advantages and disadvantages of performing dynamic analysis on physical and virtual machines were explored.

The importance of having an incident response plan and how malware analysis fits into incident response were discussed. An example of a malware analysis workflow was presented.

Lastly was a discussion of the main competition to SFTT. These are online services that perform some combination of scanning files using multiple anti-virus solutions and running the files inside a controlled environment to observe their behavior. Discussions of counterintelligence concepts, and specifically the need to protect information, were used to show the disadvantages of using these online services. Some examples of *unintended leakage* of information related to use of online anti-virus and file scanning services were discussed.

SFTT in Relation to the Literature Review

SFTT was influenced in several key ways by the RegRipper tool. These include the use a plug-in framework that will make it easier to add and remove specific functionality and a command line interface with similar options. The design of SFTT described in the next section will assume implementation in the Perl scripting language,

which is also used to implement RegRipper. Specific design choices made in RegRipper will be carried over to SFTT where appropriate.

A main priority of SFTT is to minimize information that could be discerned by an adversary based on the usage of the tool. Doing so adheres to one of the main principles of counterintelligence, which is denying the enemy access to information that might be used against an organization. Many SFTT plug-ins could use online services that either release uploaded files to third parties like anti-virus companies or make it possible to others to see what files have been processed by the tool.

Releasing malware files to online services could be a source of embarrassment or reveal damaging information. A specific concern with malware is that it could be targeted or unique malware. Writers of this kind of malware could use a search engine to look for its cryptographic hash and see if any services have processed it. This would indicate that the malware has been discovered and allow them to react accordingly. SFTT avoids these issues by just searching for existing reports on these services using cryptographic hash values.

SFTT will be an open source tool. Releasing SFTT under this type of license is intended to encourage individuals to contribute plug-ins to the tool and to contribute to the malware analysis and computer forensic communities. Providing the source code of SFTT and its plug-ins will simplify tool validation. This is important if information gathered by SFTT is used in a courtroom where it will have to meet the four categories of the Daubert Test. Those are accuracy of operation, understanding of the error rate of the tool, information about how the tool operates, and acceptance of the tool in the forensic community.

SFTT will be a static analysis malware tool because it will not execute the file that it is processing. In the context of the malware analysis workflow that was shown in the literature review SFTT is part of basic static analysis that is performed on any files that have not been identified previously (see Figure 1 on page 26). Note that the types of information collected during dynamic analysis of malware, like processes generated when malware is launched, may be collected from sources by SFTT.

SFTT Core Framework Design

SFTT prerequisites. SFTT will require version 5.10.0 or newer of the Perl scripting language to be installed for its execution. Most distributions of the Linux and Mac OS X operating systems have Perl installed by default. On Windows ActivePerl Community Edition or Strawberry Perl can be downloaded and installed to meet this requirement ("ActivePerl Community Edition," n.d. "; Strawberry Perl for Windows", n.d.).

SFTT plug-ins may also have separate required modules. Modules required by the core framework and the initial plug-ins will be described later. Most SFTT operations will require a connection to the Internet to function correctly. Specific plug-ins may have other prerequisites that will be described later.

SFTT structure. SFTT will be distributable as a directory that can be dropped into the file system wherever desired. The contents of that directory are listed in Table 3 on page 40. The core framework of SFTT will consist of a single executable file called sftt.pl, which will be passed command line arguments for execution. The core framework

will coordinate execution of SFTT by processing command line arguments and executing the plug-ins.

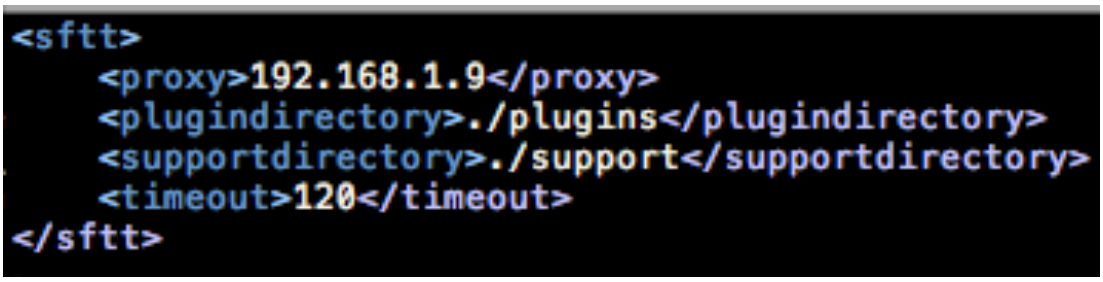
Table 3 ***SFTT File Layout***

File/Directory	Description
sftt.pl	Main executable file for SFTT
sftt_config.xml	Framework configuration file for SFTT
sftt_logging.conf	Logging configuration file for SFTT
sftt.log	SFTT log file; only exists if tool has been run at least once (default configuration)
plugins/	Directory where SFTT plug-ins are placed (default configuration)
support/	Directory for files required by SFTT plug-ins (default configuration)
README	Instructions for using SFTT

Author implementation of CONOPS (2012, February 16)

Configuration of the SFTT core framework will be supported through the file called sftt_config.xml. This file will be XML based with a structure similar to that shown in Figure 2 on page 41. Configuration options set here will be accessible by specific plug-ins as well. By default sftt_config.xml will allow for the expected location of plug-in files and plug-in support files to be set. These by default will be ./plugins and ./support. Relative and absolute directory paths will be supported for these values and they will not be able to be overridden by plug-in configuration files. Both UNIX and Windows style directory paths will be supported.

Figure 2 ***SFTT sftt_config.xml***



Author implementation of CONOPS (2012, February 16)

A server IP address will also be configurable through file_processing.xml to configure a SOCKet Secure (SOCKS) proxy. A proxy sits between two computers over a network and routes traffic between them and allows for the users IP address to be hidden. All network-based plug-ins described in this paper will support this proxy configuration. It will be possible for specific plug-ins to be configured to override the main proxy settings.

The timeout value is the amount of time in seconds that a SFTT plug-in will wait for information to be returned from the Internet before returning a failure message to the core framework. This is only used for plug-ins that access the Internet and can be overridden in a plug-ins configuration file.

The file sftt_logging.conf will allow for configuration of comprehensive logging for SFTT as shown in Figure 3 on page 42. By default two loggers are created, one for the SFTT core and one for the SFTT plug-ins. Log messages can be classified from high

to low in order of importance as FATAL, ERROR, WARN, INFO, DEBUG, and TRACE. Both loggers are set to record the debug level DEBUG by default.

Figure 3 ***SFTT sftt_logging.conf***

```
log4perl.logger.sftt.core              = DEBUG, SFTTFileAppender
log4perl.logger.sftt.plugins           = DEBUG, SFTTFileAppender

log4perl.appender.SFFTFileAppender          = Log::Log4perl::Appender::File
log4perl.appender.SFFTFileAppender.filename = ./fp_log.log
log4perl.appender.SFFTFileAppender.layout   = Log::Log4perl::Layout::PatternLayout
log4perl.appender.SFFTFileAppender.layout.ConversionPattern = [%d{DATE}] %F:<C-D-4>%L - %m%n
```

Author implementation of CONOPS (2012, February 16)

Also by default log messages will be written to the file ./sftt.log. Based on configuration a typical log entry will look like the following:

[06 Nov 1994 15:49:37,459] sftt.pl:88 – Beginning to call plug-ins

The log entry begins with its inception date, provides the file name and line where the entry was generated, and then prints the log message.

The Perl modules used by the SFTT core framework are listed in Table 4 on page 43. These modules enable the XML based configuration files used by the SFTT core framework to be loaded and accessed and also support proxy configuration and logging. These modules are also available to any plug-ins.

SFTT functionality. As stated previously, the initial version of SFTT will be limited to a command line tool. The list of available command line flags can be found in Appendix A.

SFTT can list command line functions and plug-ins available. SFTT selects plug-ins to run by either providing the name of a specific plug-in or by providing a file that has a list of plug-ins to run. SFTT can run plug-ins against an MD5, SHA1, or SHA256 cryptographic hash or a binary file. If a binary file is provided the SFTT core framework will generate a SHA1 cryptographic hash to pass to plug-ins that accept hash values. Some examples of how these command line flags would be used in practice are shown in Appendix B.

Table 4 ***Perl Modules Used in SFTT Core Framework***

Perl Module	**Description**
XML::Simple	A application programming interface for XML that makes it easy to access information in the files
LWP::UserAgent	Implements a web user agent that can handle web requests and responses in HTTP style communication
LWP::Protocol::socks	Adds support for the SOCKS protocol for LWP::UserAgent
Log::Log4perl	Provides a standard interface for logging and the ability to customize logging messages and logging destinations

Author implementation of CONOPS (2012, February 17)

SFTT Plug-ins

SFTT implements its functionality using plug-ins. Each plug-in in SFTT is a Perl library that is loaded from the configured directory when SFTT executes it. This directory

is by default ./plugins. If a plug-in is named exampleplugin.pl, then the name of the plug-in is exampleplugin.

Each plug-in can have a XML based configuration file that is loaded from a plug-in support file directory. By default this directory is ./support. If the name of the plug-in is testplugin the name of the configuration file would be testplugin.xml. Any additional files required by a plug-in will also reside in the plug-in support file directory.

Plug-in API. Each SFTT is required to implement a specific set of functions expected by the core framework. These functions are listed in Appendix C.

Each plug-in can return its name and version number. It will accept either a file or a hash value and the core framework will determine what parameters to pass by calling getInput (see Appendix C). Each plug-in can be executed and will return its output, which will be in plain text.

SFTT Initial Plug-in Descriptions

Three plug-ins that will be implemented in the initial version of SFTT are described below.

Figure 4 ***ssdeep Plug-in Configuration***

```
<sftt>
    <ssdeep>
        <executable>ssdeep</executable>
        <hash_list>ssdeep_hash_list</hash_list>
        <match_threshold>80</match_threshold>
    </ssdeep>
</sftt>
```

Author implementation of CONOPS (2012, February 16)

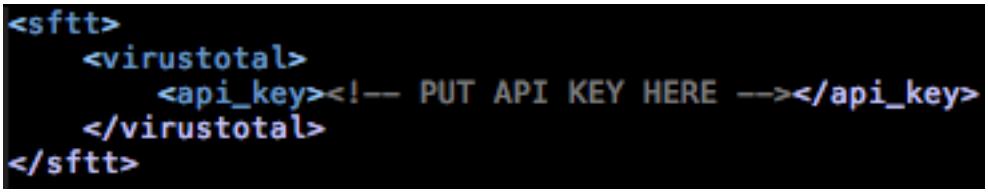
Ssdeep plug-in. The ***ssdeep plug-in*** will be used to compare the ssdeep cryptographic hash of a file against a list of ssdeep cryptographic hashes provided by the user of the plug-in. The default configuration file for ssdeep is shown in Figure 4 on page 44. The ssdeep binary and ssdeep hash list have to be installed by the user in the plug-in configuration support directory and are expected by default to be named ssdeep and ssdeep_hash_list, respectively. The match value at which ssdeep reports a possible match can also be set. The match value ranges from the lowest match value of 0 to a highest match value of 100. This value will by default be set to 80 ("Fuzzy Hashing and ssdeep," n.d.).

The ***ssdeep plug-in*** will accept two arguments. One is a file that is provided to the plug-in for comparison. The second is the list of ssdeep hashes from the plug-in configuration support directory. The plug-in will not be executed if only a hash value is provided to SFTT or if the hash list does not exist. The ***ssdeep plug-in*** will return either a list of files that matched the provided file, a message indicating that no matches were found, or an error message if the plug-in failed.

Virustotal plug-in. The ***virustotal plug-in*** will be used to send a cryptographic hash value to the VirusTotal web service. If the file has been submitted to VirusTotal previously anti-virus scanning results for the file that match that hash will be returned. If the file has not been submitted a result indicating this will be returned. This plug-in will accept MD5, SHA1, or SHA256 cryptographic hashes. Values that can be returned also include a message indicating that no anti-virus scanning values were found for a file with the hash provided or an error message if the plug-in does not function correctly.

The default configuration file for the plug-in is shown in Figure 5. The ***virustotal plug-in*** will require a public API key to be acquired from VirusTotal by creating an account on their web site. This key will then have to be pasted in the configuration before the ***virustotal plug-in*** will function ("VirusTotal FAQ", n.d.).

Figure 5 v*irustotal Plug-in Configuration*



Author implementation of CONOPS (2012, February 16)

Threatexpert plug-in. The ***threatexpert plug-in*** will send a MD5 cryptographic hash to the ThreatExpert web service and return information about the file that matches the hash if it exists. This plug-in will only accept an MD5 hash. Values that can be returned include a message indicating that no information about the files corresponding to the provided hash was found or an error message if the plug-in failed. This plug-in has no specific configuration values associated with it ("ThreatExpert", n.d.).

SFTT Limitations

Two of the three SFTT plug-ins discussed use external Internet services to perform their functions, as will most future plug-ins. This characteristic of SFTT carries some limitations with it. These services have terms of use that can change or expand.

This can make plug-in operation against the service legally questionable and create complications. Changes to the terms of use of a service can change legal usage of a service to illegal or unlicensed usage. Companies and government organizations should be particularly concerned about these issues.

Technical limitations and issues are also a concern when using external services. Using the service in an authorized fashion may impose some limitations that can be bothersome. An example would be the public API for VirusTotal limits the user to four queries per minute. If SFTT is running frequently, this throttling could lead to delays. If a service is being used in an unofficial or unsupported way, such as scraping and parsing a web page, changes to the design of the page or how the service operates could break a plug-in without warning. Lastly, a service could become unavailable because of maintenance or simply be discontinued, breaking a plug-in.

Despite the efforts taken to limit information leakage in SFTT when using external services, there are still risks. Someone monitoring incoming network traffic to a service could determine the origin of requests and that could expose intelligence about a SFTT user, even if it is just that they are using a particular proxy service. While it is not currently possible for anyone but the owners of these services to check whether a cryptographic hash value has been searched for, this could be changed by a service or a service could be compromised and the information released or exploited by a particular group.

In the last section of this paper conclusions will be drawn about SFTT based on prior discussion. Recommendations will also be provided for ways that SFTT could be expanded and possible new areas of tool functionality that could be explored.

RECOMMENDATIONS AND CONCLUSIONS

Recommendations

Based on the investigation and research performed the scope of SFTT is expandable. There are two parts of SFTT. One part accepts input, executes the appropriate plug-ins, and then provides the output. The second part is the plug-in framework that allows users to develop new plug-ins or modify existing plug-ins. The recommendations discussed below are summarized in Table 5 on page 50.

User interface improvements. Adding a graphical user interface to SFTT will increase productivity and discoverability. Plug-ins and file processing would become more user friendly. A web-based interface and modifying SFTT into a web service would make multiple concurrent user access possible.

Standardized output formats. SFTT can be improved by providing multiple output formats. This would include structured data formats like HTML or XML. Adding PDF output would make printing easier and provide portability. Providing the ability to email reports to users after they are generated by SFTT would enhance the user experience.

Automated processing. It may be useful to integrate SFTT into a larger, fully automated sequence. SFTT could check the contents of a directory and process any files that appear. Based on the SFTT results the automated process engine would decide whether or not to engage in further analysis. An example of a tool that could be integrated with is the Cuckoo Sandbox. This tool uses the virtualization software VirtualBox to perform automated dynamic analysis of files (“Cuckoo Sandbox”, n.d.).

SFTT would triage files to determine whether or not they need to be processed by Cuckoo Sandbox, based on some automated criteria.

New SFTT plug-ins. Other malware static analysis actions could be automated by adding new plug-ins to SFTT. One way to address the concern of information leakage when using external services is to set up services within an organization's infrastructure. There are companies like FireEye that develop network appliances to perform automated dynamic analysis of suspected malware files ("FireEye Malware Analysis System", n.d.). SFTT could send files to these appliances and then process the information that is returned.

Anti-virus software could be installed on multiple virtual machines and then used to test suspicious files. SFTT would transmit files to the virtual machines or to a directory that is mounted by these virtual machines and then process the returning results.

Implement SFTT. A limitation of this report is that the tool was not implemented and tested. Writing a new paper that describes the creation of SFTT as planned and any unexpected challenges or benefits that were encountered would provide additional useful information.

In addition to discussing the implementation of SFTT this paper should address providing a test version to some malware analysts and incident responders and discuss feedback. It is also important to test the tool in a controlled environment to evaluate accuracy and usability issues. Load testing would provide insight into the performance of SFTT. This could be as simple as running a number of files through the tool multiple times and monitoring computer resource usage and execution times.

Expanding the scope of study around SFTT and looking at other types of tools and plug-ins that could be developed using the same concepts will help to develop a greater understanding of what can be accomplished. This area of malware analysis is open to the continued creation and refinement of tools.

Table 5 ***Summary of Recommendations***

Recommendations

Add a web-based interface to allow access by multiple users at the same time
Provide multiple output formats for results, like XML, PDF, and HTML
Integrate into one or more tools to allow for automation to be expanded across multiple steps of malware analysis
Add a plug-in to support sending files and receiving reports from proprietary dynamic analysis engines like FireEye
Add a plug-in to support send files and receiving results from an intranet anti-virus farm
Write a report in the future about implementing SFTT and the complications and benefits encountered

Author recommendations (2012, February 21)

Conclusions

The purpose of this paper was to develop a tool to support the gathering of open source intelligence related to files that could be malware related. This open source intelligence could be gathered from the Internet or from information already compiled by the user of the tool. The tool is called the Suspicious File Triage Tool, or SFTT. A malware analyst or incident responder can use this tool during the initial part of a

malware investigation or it may be implemented to automatically preprocess samples prior to being submitted for manual analysis. The information obtained will help to determine what subsequent steps should be taken in an investigation.

Malware analysis has become an important part of investigating security breaches due to the high likelihood that malware is involved. This extends to identifying the perpetrators of the breach and determining the best response. Manual analysis of malware is very time consuming and any method that decreases the time required is highly desirable.

Part of addressing the challenge of processing and understanding the large quantities of data involved with malware investigations is the use of software tools. Automating malware analysis tasks with tools like SFTT frees the time of the malware analyst to focus on tasks that require expertise and creativity. It also allows for samples that are not malware or have already been analyzed to be handled earlier in the process, which increases efficiency. Performing this automation in the context of a malware investigation workflow allows for more time to be saved and reduces the chance than an important step will be overlooked.

Since many investigations that involve analysis of malware may end up in a court of law, it is important that SFTT meet legal requirements. Software tool output is treated like scientific evidence and a set of four criteria called the Daubert Test are used to evaluate it. These criteria are whether or not the tool produces accurate results, what the error rate of the tool is, whether the tool has been peer reviewed and documented in some public fashion, and acceptance of the tool in the forensic community. The fact that SFTT

will have its source code available under an open source license will allow its technical merits to be evaluated openly and its functionality to be discoverable.

There are several online services like VirusTotal and ThreatExpert that will accept uploaded files, analyze them against anti-virus software and other tools, and then display that information to the user. A principle of counterintelligence is the protection of friendly assets and intelligence. Uploading files to these services violates that principle by exposing damaging or embarrassing information to adversaries or third parties. SFTT can avoid these issues by only querying information from these services and not submitting files to them.

Table 6 ***Summary of Conclusions***

Conclusions

The growth of malware has made malware analysis more difficult and more important
Software tools, automation and workflows help to meet the challenges of malware growth
Tools like SFTT have to meet the criteria of the Daubert Test to be used in a legal investigation
Uploading files to services like VirusTotal make information leakage an issue; SFTT lessens this leakage considerably by using these services in different ways
SFTT can be built to meet its requirements
SFTT has limitations related to using external services such as availability and terms of use changes
SFTT can be expanded and evolved to add new functionality and interfaces

Author conclusions (2012, February 21)

SFTT will be a plug-in based tool with a core framework inspired by the Windows Registry analysis tool RegRipper. As long as plug-ins are written to a specific standard expected by the core framework they can be used by the tool. An examination of how SFTT can be implemented shows that it is entirely feasible to design a tool that meets these requirements. There are some possible limitations with this approach. External services used by SFTT plug-ins can malfunction or shutdown. Interfaces to these services can change, with little or no notice, also causing plug-ins to break. Despite efforts to limit information leakage changes to an external service policy or compromise of an external service could still allow this to occur.

SFTT has the potential to be expanded and evolve in many new ways that still fit its core focus. New plug-ins can add new capabilities, such as being able to integrate with commercial malware analysis tools. Adding new methods to interact with SFTT, such as a web interface, would reduce the learning curve for using SFTT.

Despite the growing threat posed by malware, there are positive steps that can be taken to address these challenges. By building new tools like SFTT and expanding the range of tools available the difficulties of malware analysis can be mitigated and less skilled personnel can perform portions of the investigation process, allowing the more experienced analysts to focus on areas that require manual analysis.

Appendix A

SFTT Command Line Options

Option	Description
-f, --file <FILE>	Specify file named <FILE> to process (cannot be used with –m, -s, or –s256)
-m, --md5	Specify MD5 cryptographic hash value to process (cannot be used with –f, -s, or –s256)
-s, --sha1	Specify SHA1 cryptographic hash value to process (cannot be used with –f, -m, or –s256)
-s256, --sha256	Specify SHA256 cryptographic hash value to process (cannot be used with –f, -m, or –s)
-p <PLUG-IN>, --plugin <PLUG-IN>	Run plug-in <PLUG-IN> (cannot be used with –L)
-L <LIST>, --pluginlist <LIST>	Runs all plug-ins in <LIST> (cannot be used with –p)
-l, --listplugins	List available plug-ins (only used by itself)
-h, -?, --help	Help (print this list) (only used by itself)
-v, --version	List the version of SFTT and all plug-in versions

Author implementation of CONOPS (2012, February 17)

Appendix B

SFTT Examples of Usage

Usage Example	Command Line
Printing help information	./sftt.pl --help
List all available plug-ins	./sftt.pl --listplugins
Run SFTT against SHA1 hash using the plug-in list plugins_used.txt	./sftt.pl --sha1 8c9d61d849a71c7d56f0424f1ada4420b9aedd39 --pluginlist plugins_used.txt
Run SFTT against file example.exe with the plug-ins named ssdeep and threatexpert	./sftt.pl --file example.exe --plugin ssdeep --plugin threatexpert

Author implementation of CONOPS (2012, February 17)

Appendix C

SFTT Plug-in API

Function Name	Description
getName	Returns name of the plug-in
getVersion	Returns version number of plug-in (e.g. “1.0”)
getInput	Returns type of input value expected by the plug-in. Possible values are “md5”, “sha1”, “sha256” and “file”
pluginMain	Function that is called to execute the plug-in. Returns output from executing plug-in

Author implementation of CONOPS (2012, February 17)

Appendix D
Glossary

Anti-virus Software – Software used to scan potential malware sources against signatures and to perform actions against data that matches. This can include quarantining or removing the data (Harris, 2010).

API – Acronym for application programming interface. It is a set of instructions and standards for accessing particular software (Roos, n.d.).

ASCII – Acronym for American Standard Code for Information Interchange. ASCII is a character encoding format that converts binary into readable characters (Harris, 2010).

Bot – A type of malware designed to operate in concert with a larger group of computers, generally for malicious purposes (OECD, 2008).

Command Line Interface – Interacting with a system in real time via typed commands. Sometimes shortened to CLI (Stephenson, 1999).

Computer Forensics – The recovery, authentication, and analysis of electronic data using specialized techniques, often for the purposes of a criminal investigation (Harris, 2010).

CONOPS – Acronym for concept of operations. A statement of what is intended to be accomplished with available resources (Department of Defense, 2010).

Context Triggered Piecewise Hash – A cryptographic hash created by hashing a file in pieces which size are determined by monitoring the results of generating a rolling hash at the same time. This is used by the ssdeep tool (Kornblum, 2006).

Counterintelligence– Activities undertaken against enemy intelligence operations such as identifying that the intelligence operations exist and then deceiving or disrupting them (Department of Defense, 2010).

CPAN – Acronym for Comprehensive Perl Archive Network. It is a collection of software and documentation and software for the Perl scripting language ("The CPAN Frequently Asked Questions", n.d.).

Cryptographic Hash - A fixed-length value produced mathematically from a variable-length string. The fixed-length value cannot be reversed to produce the variable –length string (Harris, 2010).

Daubert Test – A set of criteria to determine whether scientific evidence is admissible in a court of law. Named after the Supreme Court ruling in Daubert vs. Merrill Dow Pharmaceuticals in 1993 (Carrier, 2003).

Distributed Denial of Service Attack – A network attack where many computers from several different locations overwhelm a computer with traffic (Harris, 2010)

Dynamic Analysis (Malware) – The process of determining what happens when a malware file is executed (Hutcheson, 2006).

Encryption – A mechanism for protecting information from unintended disclosure by making data unreadable to all but the intended destination (Krutz & Vines, 2001).

False Negative – When a condition, such as software vulnerability, exists but is not detected by a tool ("What is a False Negative?" n.d.).

False Positive – When a condition, such as software vulnerability, does not exist but is detected by a tool ("What is a False Positive?" n.d.).

Firewall – Software or hardware used to restrict access from one network to another network (Harris, 2010).

Graphical User Interface – Interacting with a system via visual abstractions such as icons, windows, and a mouse pointer. Sometimes shortened to GUI (Stephenson, 1999).

Heuristic Analysis – Uses reaching a threshold of indicators that might be present to determine whether or not data might be malware (Castelli, 2001).

HTML – A markup language used to format web pages (Harris, 2010).

Intrusion Detection System – Software used to monitor network activity and look for behaviors that vary from the expected activity of the network (Harris, 2010).

IP Address – A numerical value that identifies a device on a network. Is 32 bit in IPV4 and 128 bit in IPV6 (Harris, 2010).

Malicious Mobile Code – A type of malware that is a small program that is downloaded from remote locations and executed without the permission of the user (Skoudis & Zeltser, 2004).

Malware – Malicious code from an attacker that runs on a computing device (Skoudis & Zeltser, 2004).

Malware Analysis – The examination of malware in all its forms ("Malware Analyst – Job Description", n.d.).

Metadata – Data that describes data (Harris, 2010).

Open Source Intelligence – Information that is gathered from publically available sources such as newspapers and web logs (George & Bruce, 2008).

Open Source Software – Software in which the source code is made available with the binary. The license the software is under allows for the source code to be changed, redistributed, and used in other projects ("Open Source Initiative", n.d.).

PDF – A document container format. Allows for content to look the same on a screen and on the printed page (U.S. Census Bureau, 2011).

Perl – A programming language in use for over 23 years. Often used for scripting and web infrastructure ("About Perl", n.d.).

Phishing - A social engineering attack intended to obtain data such as credit cards and personal information (Harris, 2010).

Process – A set of instructions that is running on a computer that carry out several types of functionality (Harris, 2010).

Proxy – Sits in the middle of two endpoint connections and routes traffic between them (Harris, 2010).

RegRipper – A plug-in based tool written in Perl for processing and parsing data from Windows registry hives (Carvey, 2011).

Rootkit – User-level rootkits replace or modify standard system executables to hide malicious activity or create backdoors. Kernel-level rootkits modify the core of the operating system (kernel) for the same purpose (Skoudis & Zeltser, 2004).

Signature Analysis – Using a signature database of known malware files to determine whether data being scanned is malware (Castelli, 2001).

Static Analysis (Malware) – Any examination of malware that can be done without executing it (Hutcheson, 2006).

ThreatExpert – An online service that analyzes and then reports the behavior of malware ("ThreatExpert: Introduction", n.d.)

Trojan Horse – A type of malware that appears to be a legitimate application such as a game but actually does something malicious (Skoudis & Zeltser, 2004).

Unicode – A character encoding format that can represent the all textual characters in the world as a standard format (Harris, 2010).

Virtual Machine – An instance of an operating system running in a virtual environment. Allows a single computer to run multiple operating systems at the same time (Harris, 2010).

Virus – A type of malware that infects a particular file on a device. Requires manual intervention to replicate such as launching an executable file (Skoudis & Zeltser, 2004).

VirusTotal – An online service that analyzes files and web links for malicious content ("About VirusTotal", n.d.).

Windows Registry – A database of configuration information that is binary and hierarchical on modern Windows systems (Carvey, 2011).

Workflow – A series of connected steps, such as the process for requesting a new user account (Harris, 2010)

Worm – A type of malware whose main characteristic is that it can spread across a network without manual intervention (Skoudis & Zeltser, 2004)

XML – Acronym for Extensible Markup Language. A markup language used to create other markup languages. A markup language allows for text to be constructed and viewed in a certain way (Harris, 2010).

Appendix E

BSD 3-Clause License Template

Copyright (c) <YEAR>, <OWNER>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

(“Open Source Initiative”, n.d.)

REFERENCES

About Perl (n.d.). Retrieved February 22, 2012 from http://www.perl.org/about.html

About VirusTotal (n.d.). Retrieved February 22, 2012 from https://www.virustotal.com/about/

ActiveState Community Edition (n.d.). Retrieved February 18, 2012 from http://www.activestate.com/activeperl/downloads

Adair, S. & Hale Ligh, M. & Hartstein, B. & Richard, M. (2011). Malware Analyst's Cookbook and DVD. In C. Long, M. Spears, M. Gregg, K. Wisor & N. Rappaport (Eds.). Indianapolis, IN: Wiley Publishing, Inc.

APWG (2009). Phishing Activities Trends Report 3rd Quarter 2009 [White Paper] Retrieved from http://www.antiphishing.org/reports/apwg_report_Q3_2009.pdf

AV Comparatives (2011). *Summary Report 2011* [White Paper]. Retrieved from http://www.av-comparatives.org/images/stories/test/summary/summary2011.pdf

Ayers, D. (2009). A second generation computer forensic analysis system. *Digital Investigation, 6*, S34–S42. doi:10.1016/j.diin.2009.06.013

Altheide, C. & Carvey, H. (2011). Digital Forensics with Open Source Tools. In R. Davidson, A. Ward, & H. Scherer (Eds.), Disk and File System Analysis (pp. 39-67). Waltham, MA: Syngress Publishing, Inc.

Bartolomie, J. (2011, October 10). Operational Security Operations. ITT ICS-01, Approved for Public Release 10-11, PR2011-35

Best Practical Services (n.d.). Retrieved February 11, 2012 from http://bestpractical.com/services/

Brunner, M., Epah M., Hofinger, H., Roblee, C., Schoo, P., Todt, S. (2010). Establishing a Secured, Honeynet-based Cyber Threat Analysis, and Research Environment. *The Fraunhofer AISEC Malware Analysis Report Technical Report*. Retrieved from http://www.aisec.fraunhofer.de/content/dam/aisec/en/pdf/TechReports /AISEC_MalwareLab.pdf

Carrier, B. (2002). Defining Digital Forensic Examination and Analysis Tools. Paper presented at the Digital Forensics Research Workshop 2002, Syracuse, NY. Retrieved from http://www.digital-evidence.org/papers/dfrws_define.pdf

Carrier, B. (2003). Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers. *International Journal of Digital Evidence, 1(4),* 1–12. Retrieved from http://citeseerx.ist.psu.edu/viewdoc /download?doi=10.1.1.14.9813&rep=rep1&type=pdf

Carrier, B. (2003). Open Source Digital Forensics Tools The Legal Argument. Retrieved from http://www.digital-evidence.org/papers/opensrc_legal.pdf

Carvey, H. (2011). Windows Forensic Analysis. In A. Ward, E. Casey & G. Byrne (Eds.), *Registry Analysis* (pp. 157–252). Burlington, MA: Syngress Publishing, Inc.

Carvey H. (n.d.). Windows Incident Response [Web log]. Retrieved February 11, 2012 from http://windowsir.blogspot.com

Castelli, J. (2001). Choosing Your Anti-Virus Software. *SANS Institute Information Security Reading Room*. Retrieved from http://www.sans.org/reading_room/whitepapers/commerical /choosing-anti-virus-software_784

Cisco (2011). *Entering the Zettabyte Era* [White Paper]. Retrieved from http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/VNI_Hyperconnectivity_WP.pdf

Corrons, L. (2011, December 15). 2012 Security Trends [Web log post]. Retrieved from http://pandalabs.pandasecurity.com/2012-security-trends/

CPAN Frequently Asked Questions (n.d.). Retrieved December 30, 2011 from http://www.cpan.org/misc/cpan-faq.html

Cuckoo Sandbox (n.d.). Retrieved February 20, 2012 from http://cuckoobox.org/

Davis, M. (2010). *Global Threat, Local Pain: 2010 Strategic Security Survey* [White Paper]. Retrieved from http://reports.informationweek.com/abstract/21/3018/Security/research-2010-strategic-security-survey.html

Department of Defense (2010, November 8). *Department of Defense Dictionary of Military and Associated Terms* (Joint Publication 1-02). Retrieved from http://www.dtic.mil/doctrine/new_pubs/jp1_02.pdf

Distler, D. (2007). Malware Analysis: An Introduction. *SANS Institute Information Security Reading Room*. Retrieved from http://www.sans.org/reading_room/whitepapers/malicious/malware-analysis-introduction_2103

Driscoll, D. & Brizee, A. (2010). Evaluating Print vs. Internet Sources. Retrieved from http://owl.english.purdue.edu/owl/resource/553/04/

FireEye Malware Analysis System (n.d.). Retrieved February 20, 2012 from http://www.fireeye.com/products-and-solutions/malware-analysis.html

Fuzzy Hashing and ssdeep (n.d.). Retrieved February 19, 2012 from http://ssdeep.sourceforge.net/

George, Roger Z. and Bruce, James B., eds. Analyzing Intelligence. Washington, D.C.: Georgetown University Press, 2008

G Data (2011). *Security Survey 2011: How do users assess threats from the Internet?* [White paper]. Retrieved from http://www.gdata-software.com /wp-content/uploads/GData_SecuritySurvey_2011_EN2.pdf

GNU Coreutils (n.d.). Retrieved January 24, 2012 from http://www.gnu.org/software/coreutils/manual/html_node/index.html

Harris, S. (2010). *All in One CISSP Exam Guide Fifth Edition*. New York, NY: McGraw-Hill

How to use WHOIS effectively to tracker spammers (n.d.). Retrieved January 1, 2012 from http://www.netdemon.net/tutorials/whois.txt

Hutcheson, L. (2006, July 9). Malware Analysis: the Basics. Retrieved from http://isc.sans.org/presentations/cookie.pdf

Hypponen, M. (2011, August 26). How We Found the File That Was Used to Hack RSA [Web log post]. Retrieved from http://www.f-/secure.com/weblog /archives/00002226.html

Janus, M. (2011, August 16). Heads of the Hydra. Malware for Network Devices [Web log post]. Retrieved from http://www.securelist.com/en/analysis/204792187 /Heads_of_the_Hydra_Malware_for_Network_Devices

Jeffbryner (2009, August 7). RegRipper on Linux? Message posted to http://www.forensicfocus.com/index.php?name=Forums&file=viewtopic&t=4315

Kendall, K. (2007). *Practical Malware Analysis*. Retrieved from http://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan /Paper/bh-dc-07-Kendall_McMillan-WP.pdf

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation 3S*, S91–S97. doi:10.1016/j.diin.2006.06.015

Krebs, B. (2009, December 31). Virus Scanners for Virus Authors [Web log post]. Retrieved from http://krebsonsecurity.com/2009/12 /virus-scanners-for-virus-authors/

Krebs, B. (2010, April 23). Hiding from Anti-Malware Search Bots [Web log post]. Retrieved from http://krebsonsecurity.com/2010/04 /hiding-from-anti-malware-search-bots/

Krutz, R. & Vines, R. (2001). *The CISSP Prep Guide*. New York, NY: John Wiley & Sons, Inc.

Langner, R. (2010, September 16). Stuxnet logbook, Sep 16 2010, 1200 hours MESZ [Web log post]. Retrieved from http://www.langner.com/en/2010/09/16 /stuxnet-logbook-sep-16-2010-1200-hours-mesz/

McAfee (2010). *Protecting Your Critical Assets: Lessons Learned from "Operation Aurora"* [White paper]. Retrieved from http://www.wired.com/images_blogs /threatlevel/2010/03/operationaurora_wp_0310_fnl.pdf

McAfee Labs (2011). *McAfee Threats Report: Third Quarter 2011* [White paper]. Retrieved from http://www.mcafee.com/us/resources/reports /rp-quarterly-threat-q3-2011.pdf

Malware Analyst – Job Description (n.d.). Retrieved January 21, 2012 from http://zeltser.com/reverse-malware/malware-analyst-job.html

Marrs, T. (2010, September 30). The worm in the centrifuge. Retrieved from http://www.economist.com/node/17147818

MLE (2011). Getting Started with Basic Malware Analysis [Web log post]. Retrieved from http://www.zonbi.org/?p=495

National Institute of Justice (2010, November 5). The Computer Forensics Tool Testing Program. Retrieved from http://www.nij.gov/topics/forensics/evidence/digital/standards/cftt.htm

NoVirusThanks (2010, July 1). Terms of Service and Privacy Policy. Retrieved from http://www.novirusthanks.org/terms/

O'Connor, T. (2010, August 15). In *Privacy and Information Operations, MegaLinks in Criminal Justice*. Retrieved from http://drtomoconnor.com/3100/3100lect03b.htm

O'Connor, T. (2010, November 7). In *Malware, AdWare, RiskWare, & Spam, Megalinks in Criminal Justice*. Retrieved from http://www.drtomoconnor.com/3100/3100lect07a.htm

O'Connor, T. (2011, July 6). In Counterintelligence, Megalinks in Criminal Justice. Retrieved from http://www.drtomoconnor.com/4125/4125lect03a.htm

Ollman, G. (2011). *Automated In-Network Malware Analysis* [White Paper]. Retrieved from http://www.damballa.com/downloads/r_pubs/WP_MalwareVM_pitfalls.pdf

Open Source Initiative (n.d.). Licenses by Name. In *Open Source Initiative*. Retrieved February 11, 2012 from http://www.opensource.org/licenses/alphabetical

Open Source Initiative (n.d.). The BSD 3-Clause License. In *Open Source Initiative*. Retrieved December 2, 2011 from http://www.opensource.org /licenses/BSD-3-Clause

Open Source Initiative (n.d.). The Open Source Definition. In *Open Source Initiative*. Retrieved February 2, 2011 from http://www.opensource.org/docs/osd

Organization for Economic Cooperation and Development (2008). *Malicious Software (Malware): A Security Threat to the Internet Economy*. Retrieved from http://www.oecd.org/dataoecd/53/34/40724457.pdf

Panda Labs (2011). *Panda Labs Quarter Report July – September 2011* [White Paper]. Retrieved from http://press.pandasecurity.com/wp-content/uploads /2011/10/PandaLabs-Report-Q3-2011.pdf

Panda Labs (2010). *Annual Report PandaLabs 2010* [White Paper]. Retrieved from http://press.pandasecurity.com/wp-content/uploads/2010/05 /PandaLabs-Annual-Report-2010.pdf

Perl2Exe Home Page (n.d.). Retrieved January 8, 2012 from http://www.indigostar.com/perl2exe.php

Proxy.org (n.d.). Retrieved January 1, 2012 from http://proxy.org

Raymond, E. (2001). The Cathedral and the Bazaar. In O'Reily, T. & Shangraw, S. [Eds.]. Sebastopol, CA: O'Reily Media, Inc.

Raymond, E. (2004). The Art of UNIX Programming. In Kernighan, B. [Ed.]. Boston, MA: Pearson Education, Inc.

Regripperplugins Downloads (n.d.). Retrieved January 8, 2012 from http://code.google.com/p/regripperplugins/downloads/list

Roos, D. (n.d.). How to Leverage an API for Conferencing [Web log post]. Retrieved February 22, 2012 from http://money.howstuffworks.com /business-communications/how-to-leverage-an-api-for-conferencing1.htm

Schatz, B. (2007). *Digital Evidence: Representation and Assurance* (Doctoral thesis, Queensland University of Technology, Brisbane, Australia). Retrieved from http://eprints.qut.edu.au/16507/1/Bradley_Schatz_Thesis.pdf

Scientific Working Group on Imaging Technologies (1999, October). Definitions and Guidelines for the Use of Imaging Technologies in the Criminal Justice System. *Forensic Science Communications*, 1(3). Retrieved from http://www.fbi.gov /about-us/lab/forensic-science-communications/fsc/april1999/swgit1.htm/

Secunia Advisory SA43910 (2011, April 13). Retrieved January 22, 2010 from http://secunia.com/advisories/43190

Skoudis, E. & Zeltser, L. (2004). Malware: Fighting Malicious Code. Franz, M. [Ed.]. Upper Saddle River, NJ: Prentice Hall PTR.

Strawberry Perl for Windows (n.d.). Retrieved February 18, 2012 from http://strawberryperl.com/

Stephenson, N. (1999). In the Beginning was the Command Line. Retrieved from http://www.cryptonomicon.com/beginning.html

Sullivan, S. (2011, October 11).More Info on German State Backdoor: Case R2D2 [Web log post]. Retrieved from http://www.f-secure.com/weblog /archives/00002250.html

ThreatExpert (n.d.). Retrieved February 19, 2012 from http://threatexpert.com/

ThreatExpert: Introduction (n.d.). Retrieved February 22, 2012 from http://www.threatexpert.com/introduction.aspx

Villeneuve, N. (2010, November 12). *Koobface: Inside a Crimeware Network*. Retrieved from http://www.infowar-monitor.net/reports/iwm-koobface.pdf

U.S. Census Bureau (2011, October 26). Portable Document Format (PDF). Retrieved from http://www.census.gov/main/www/pdf.html

U.S. Marine Corps (2000, 5 September). *Counterintelligence* (MCWP 2-14). Retrieved from http://www.fas.org/irp/doddir/usmc/mcwp2-14.pdf

VirusTotal FAQ (n.d.). Retrieved December 20, 2011 from http://www.virustotal.com/faq.html

VirusTotal Search (n.d.). Retrieved December 20, 2011 from http://www.virustotal.com/search.html

What is a False Negative? (n.d.) Retrieved February 22, 2011 from http://www.cgisecurity.com/questions/falsenegative.shtml

What is a False Positive? (n.d.). Retrieved February 22, 2011 from http://www.cgisecurity.com/questions/falsepositive.shtml

Werner, T. (2011, October 18). Federal Trojan's got a "Big Brother" [Web log post]. Retrieved from http://www.securelist.com/en/blog/208193167 /Federal_Trojan_s_got_a_Big_Brother

Winforensicanalysis Downloads (n.d.) Retrieved January 8, 2012 from http://code.google.com/p/winforensicaanalysis/downloads/list

Zeltser, L. (2010). Introduction to Malware Analysis. SANS Institute. Retrieved from http://zeltser.com/reverse-malware/intro-to-malware-analysis.pdf

Zeltser, L. (2010, October 9). Free Toolkits for Automating Malware Analysis [Web log]. Retrieved from http://blog.zeltser.com/post/1284687696/malware-analysis-tool-frameworks

Zeltser, L. (n.d.). Lenny Zeltser on Information Security [Web log]. Retrieved February 11, 2012 from http://blog.zeltser.com

INDEX

P

Q

R

S

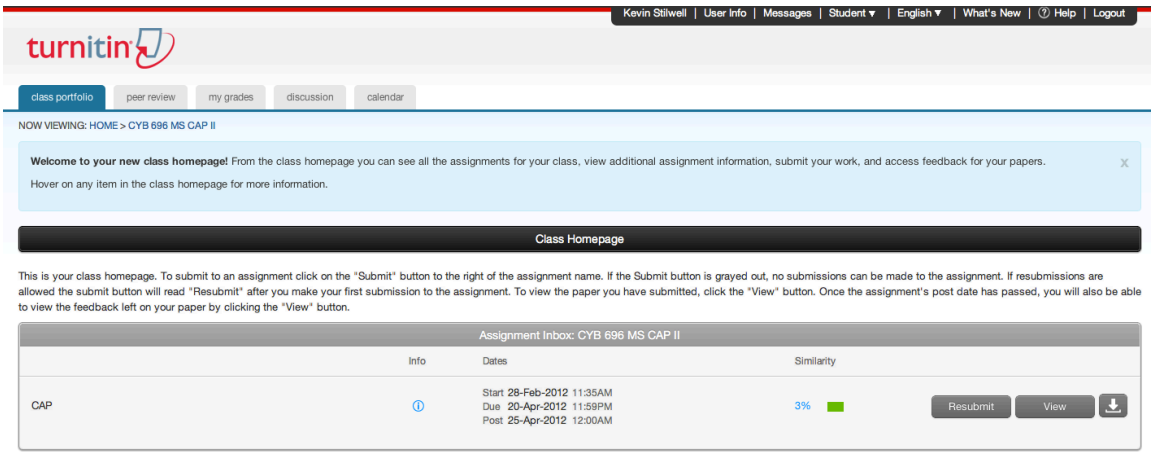
T

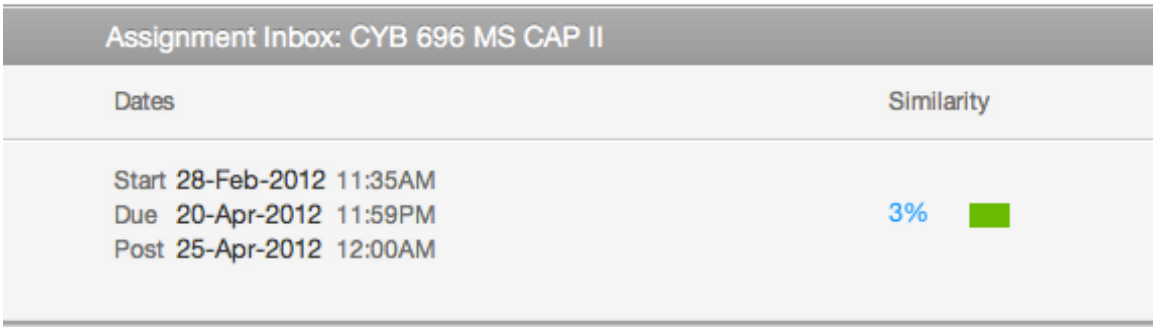
V

X

Z

TURNITIN RESULTS





END FLYLEAF PAGE