EXFILTRATION TECHNIQUES: AN EXAMINATION AND EMULATION

by

Ryan C. Van Antwerp

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Spring 2011

EXFILTRATION TECHNIQUES: AN EXAMINATION AND EMULATION

by

Ryan C. Van Antwerp

Approved: ____
Fouad Kiamilev, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: ____
Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: ____
Michael J. Chajes, Ph.D.
Dean of the College of Engineering

Approved: ____
Charles G. Riordan, Ph.D.
Vice Provost for Graduate and Professional Education

ACKNOWLEDGEMENTS

To Dave Sincoskie: Dr. Sincoskie gave me the amazing opportunity to continue my education under his supervision. I would not be where I am today without his guidance and teachings. He provided great insight into the field of cybersecurity as well as the obstacles I faced in academia. He taught me to not give up at the first sign of hardship and provided direction when I had hit a dead end. Not a day goes by where he is not missed and he will always be remembered for his wisdom, compassion, and his willingness to help.

To Chase Cotton: Dr. Cotton has provided great guidance into this field of research. His practical experience and knowledge have proved to be invaluable. I could always rely on him to provide assistance and to brainstorm. I never left his office without having a solution to the problem I came to him with.

To Fouad Kiamilev: Dr. Kiamilev has been incredibly generous in providing a lab for me to work in. This environment has been an incredible place to work, learn, and build. The support he has provided me throughout this research has been exceptionally helpful.

To my fiancée, Hye-Jin Gehring: I cannot wait to start my life with Hye-Jin. She has always been kind, caring, and supportive despite my late nights at the lab and stressful moods. She is my best friend and I could not imagine life without her.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Data exfiltration is the process of transmitting data from an infected or attacker-controlled machine back to the attacker while attempting to minimize detection. In current attack scenarios, an attacker will attempt to break into a network, achieve control of a target machine and steal sensitive data. Current network defense mechanisms are largely implemented to prevent attackers from entering a network, however there are typically few defenses implemented which prevent sensitive data from leaving a network. In addition, a major obstacle is the inability of researchers to know exactly how data will be exfiltrated from a machine. Currently, detection suites focus on attributes of the sensitive data being stolen such as file names and keywords. However, simple modification by the attacker of the data or the exfiltration channel can bypass these defense mechanisms. In order to better understand how to defend against this type of activity, the attack surface must be examined.

In this research, we examine the attack surface of data exfiltration by characterizing different exfiltration methods and observing common characteristics between them. By exploring the taxonomy of exfiltration techniques, we hope to help the research community improve existing detection algorithms and identify patterns that can be used to create new detection algorithms. After examining each method, a test bench suite was designed and implemented which emulates the data exfiltration process. This plug-in based framework allows a researcher to test common exfiltration methods on any given data. The framework is also extendable in that plug-ins can be quickly implemented using a wide array of existing libraries. The results from this research show that there is a set of common characteristics among all methods that can be used to help further research of detection algorithms. Features such as exfiltration timing, destination determination and traffic symmetry can be used to construct a stronger detection suite.

Chapter 1

BACKGROUND

One important but often overlooked aspect of any computer network is the security of the network. The loss of sensitive data can cause an immense amount of damage, regardless of the organization the network is serving. For consumers, networks can contain the online identities of individuals, allowing attackers to steal credit card numbers, passwords, and other data that can be used for a multitude of different damaging acts. For corporations and organizations, the damage can be much worse. Attacks will not simply harm the individual, but rather the organization as a whole. Attackers can steal trade secrets, financial records, and other information that can result in the abrupt downfall of that corporation. Even more damaging, attacks targeted at governmental and military networks can result in severe consequences, specifically when military[18] or political[20] data is stolen. These attacks happen frequently and it is becoming clear that new actions must be taken to prevent data theft.

The main focus of recent research in cybersecurity and the protection of government, corporate, and consumer computer systems and information has been on attempting to prevent attackers from breaking into a system. Most defensive approaches involve using an Intrusion Detection System (IDS) coupled with least privilege security policies to ensure that attackers cannot break in without overcoming several security layers. While this aspect of defense is necessary, it is important to look at other protective techniques. Intrusion detection tools are useful, yet cannot analyze all network traffic on large networks without substantial and expensive hardware or high overhead resulting in network slowdowns. Another important aspect of network breaches is that even with extensive defenses focused on preventing network intrusion, the most ardent and persistent hackers will eventually breach a targeted network[6]. Several layers of defenses may stop an attacker who is targeting large ranges for any vulnerable target, however attackers will not give up at the sign of a strong defense for targets such as government or military networks. Breaching these networks can provide much greater gains, particularly in times of war and political activities. For these reasons, it is important to not only examine incoming traffic for anomalies that could signify a network breach, but also to examine outgoing traffic for the results of a network breach. In order to gain from a network breach, an attacker must find a way to transmit data from the infiltrated machine back to an attacker-controlled machine. Commercial tools do exist that attempt to detect sensitive data from leaving a network. These tools are known as Data Leak Prevention (DLP) suites[16]. However, as their name implies, they are targeted at preventing accidental data leakage and only the most primitive attacks. The majority of these tools focus on expression or keyword based algorithms, which an attacker can overcome rather simply.

The current state of exfiltration detection appears to be focused on detecting the attributes and data of sensitive files rather than the actual method of transmission and behavioral aspects. Most exfiltration detection suites enumerate files based on file names and hashes[16]. These signatures are examined when inspecting outgoing traffic. The alternate approach that exists is the use of regular expressions that match common data formats such as credit card numbers and social security numbers. This approach is similar to the approach taken by antivirus companies and has similar flaws. The most glaring issue with this approach is that a simple modification of the data or file can cause it to be transmitted unnoticed. Furthermore, most exfiltration methods can simply compress or encrypt the data, thus making the examination of the data impossible. Since a user will commonly and increasingly communicate to servers using encrypted traffic[12], the encrypted or compressed exfiltrated traffic will be lost among the noise. To demonstrate the obstacles that current detection suites face, consider the traffic sample outlined in figure 1.1. This figure represents an hour of traffic on a single machine that utilizies a variety of different protocols. Each point is a packet that corresponds to a port number. The first obstacle is the sheer amount of traffic and breadth of protocols that must be understood and examined for exfiltrated data. Data is being exfiltrated in that traffic sample, yet there is nothing alarming that can be gleaned from examination of the traffic. In figure 1.2, the data exfiltration is highlighted. In this case, the traffic is on port 443 and encrypted, rendering packet inspection suites useless. Also, due to the amount of legitimate traffic on that port, it is nearly impossible to pick out the exfiltrated data as suspicious.

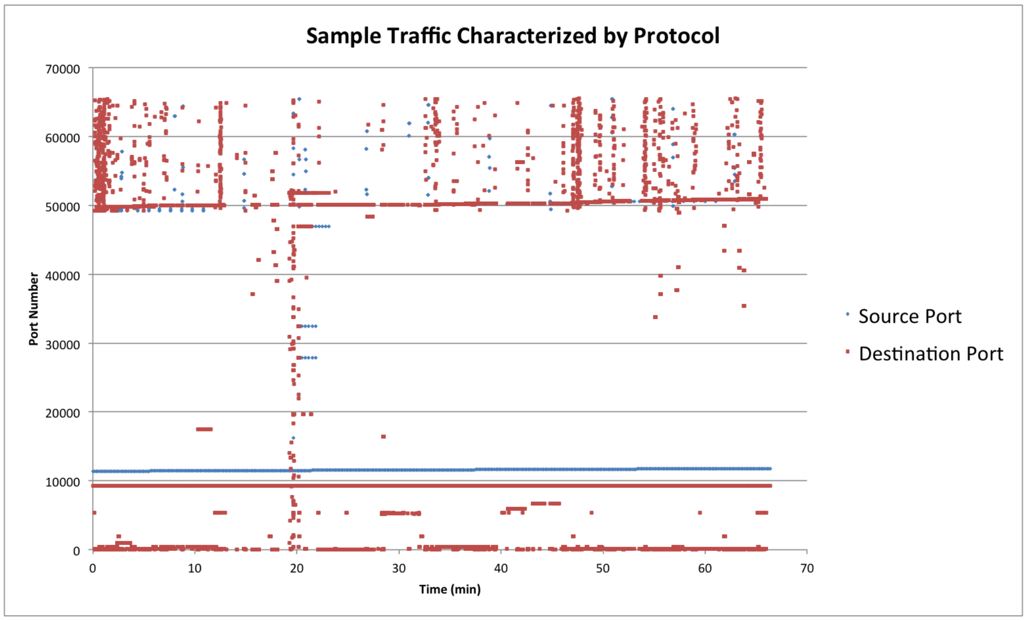

Figure 1.1: Data exfiltration hidden in sample traffic

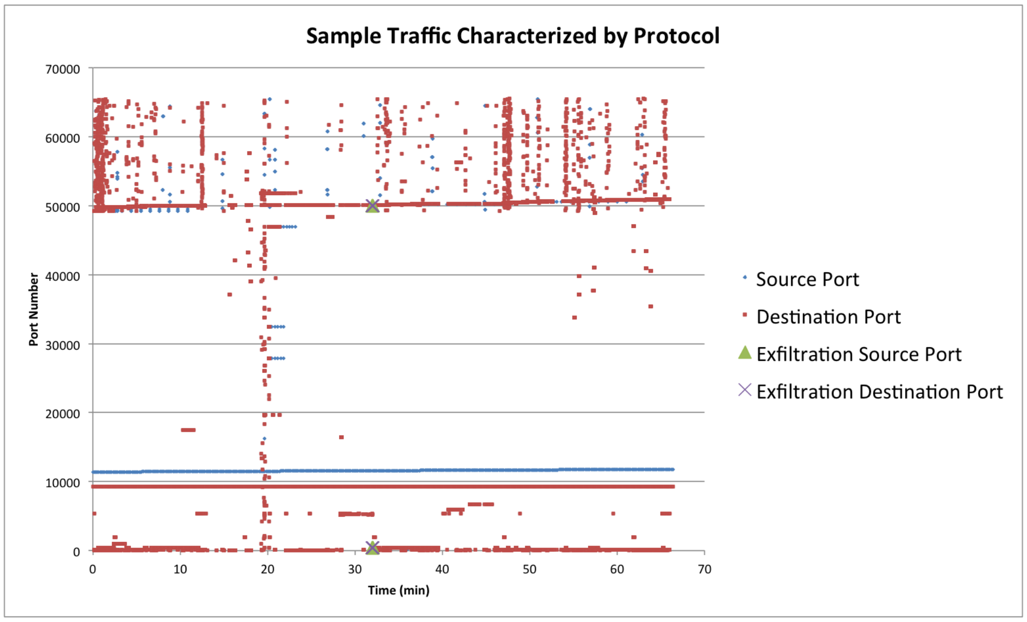

Figure 1.2: Data exfiltration highlighted among sample traffic

There is a small but growing movement towards behavior analysis that will lead to more accurate anomaly detection. A behavioral algorithm will examine all traffic from a host for a period of time that it is assumed to be known as safe. After this learning period, a host-based agent will know which traffic is expected (e.g. webmail, general browsing) and which traffic is an anomaly (e.g. SSH to previously unknown server.) This approach is not perfect and will likely result in many false positives, however it is a step in the right direction. It is clear that the focus of exfiltration detection must shift focus from the data being stolen to other behavioral aspects such as the method of data theft and typical user actions.

One of the main reasons that more complicated or specific algorithms do not exist is due to the lack of exfiltration traffic samples and the difficulty of malware analysis. In this research, we attempt to categorize exfiltration techniques and create an emulation tool corresponding to the most popular methods. This tool will provide a flexible way of creating exfiltration traffic samples that can be used to design more advanced and efficient detection algorithms.

Chapter 2

INTRODUCTION

2.1 Motivation

There are a number of reasons an attacker would attempt to infiltrate a private network. The most common goals are to block the service from use by legitimate individuals, to take over systems in order to send spam messages, and to steal information. The first case, known as a denial of service attack (DoS) usually occurs as a primitive way of temporarily disabling a service. This type of attack can be accomplished without infiltrating the network by simply saturating the normal means of accessing the network. The other goals typically require an attacker breaking into a system or network. While sending spam messages can be profitable, there are many guards in place in mail servers and effective systems that nearly eliminate spam exposure to the average user[5]. For these reasons, the most profitable and most likely activities once a network has been infiltrated is the theft of sensitive information such as passwords, credit card information, and other personally identifiable data that can be used to provide financial gain.

2.2 The Exfiltration Process

There is currently a relatively small amount of information known about existing exfiltration techinques. There are certain steps, however, that an attacker must go through in order succesfully exfiltrate data. The attacker must first breach a private network. Consider the network in figure 2.1. Most private networks have a network gateway and firewall in which there may be significant defensive tools that limit incoming traffic. Conversely, there are likely to be very few, if any, defensive mechanisms examining outgoing traffic. Attacks such as cross-site scripting (XSS) and SQL injections will not be hampered by a defensive firewall and after the attacker gains control of the system, he or she will have little trouble transmitting traffic out of the network. After exfiltration, a sophisticated attacker may remove traces of his or her existence by the modification of system logs.

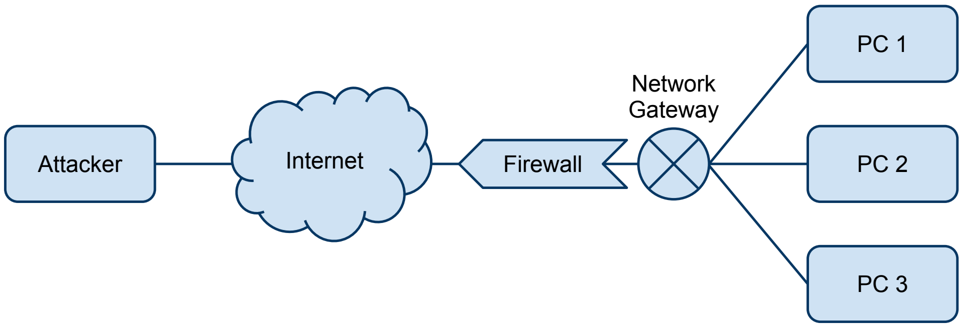

Figure 2.1: The basic architecture the attacker manipulates in order to exfiltrate data

In the event that a network has a DLP or deep packet inspection (DPI) solution in place, as illustrated in figure 2.2, it will have to inspect every piece of outgoing traffic. Most DLP tools are very costly and have dedicated hardware, precisely for this reason. However, this hardware is useless in the face of encrypted traffic. When a channel or piece of data is encrypted, it has a high entropy and will appear to contain random binary information. The attacker can decrypt this data after transmission and it will have passed through the detection suite without setting off an alarm.

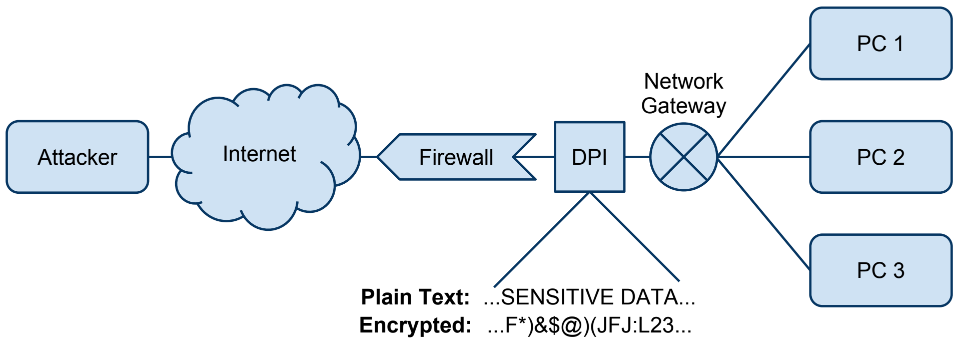
Attacker
Internet
Firewall
DPI
Network Gateway
PC 1
PC 2
PC 3
Plain Text: ...SENSITIVE DATA...
Encrypted: ...F*)&$@)(JFJ:L23...

Figure 2.2: The effects of encryption on an outgoing DPI solution

2.3 Impact of Data Theft

While theft of information from consumers can be costly, the real danger arises from theft of data from corporate and government sources. Theft of information such as trade secrets or marketing strategies from a corporation can cause the corporation brand harm and have a considerable financial impact. Government targets can have much more widespread and severe consequences such as large scale financial loss and significant threats to a nation's safety. An example of this kind of backlash can be observed from the recent controversey surrounding the data theft by Bradly Manning[20].

2.4 Malware Anti-Analysis Techniques

The obstacles that prevent more from being known about these methods from malware are due to the difficulty of malware binary analysis and the industry focus on how breaches occur, rather than how data is exfiltrated back to the attacker. Modern malware has become increasigly sophisticated, particularly in ways that prevent researchers from analyzing and reverse engineering malware samples. Current malware commonly includes code obfuscation, debugger run-time detection, binary packing, and virtual machine detection.

2.4.1 Code Obsfuscation

Obfuscation is a method of complicating the binary executable or providing redundant instructions in code in order to confuse a malware analyzer. Examples of this include creating pointless loops or extensive function nesting which provide no real purpose in the code except to complicate analysis. This method slows down and limits examination of the malware to only the most experienced analysts and provides malware a longer time period to gather data or infiltrate systems before fixers are put in place that would combat the exploit.

2.4.2 Debugger Detection

Most modern malware can easily detect if it is being run inside of a debugger and will often exhibit incorrect behavior in this state in an attempt to slow down the analyzer. Without help of a debugger, malware analysis can be incredibly difficult and time consuming. In order to overcome debugger detection, some modifications must be made to either the debugging environment or to the binary itself, both of which further delay the malware analysis.

2.4.3 Binary Packing

Malware are now often packed(compressed) and when executed, will self modify its code or unpack the full payload. There are often checks in place to ensure the environment is a real and valid target and not a virtual machine or sandbox, further complicating analysis. Therefore, the binary must be put in an environment that will cause it to unpack completely and then allow the malware to be analyzed.

2.4.4 Virtual Machine Detection

In line with binary packing and code obsfucation, most modern malware goes to great lengths to ensure it is not running in a virtual machine or similar sandbox environment. Tests performed by the malware may be as simple as testing the

MAC address manufacturer code, or as complicated as timing and hardware tests that provide different results depending on whether the malware is running on a real system or not.

2.5 Exfiltration Test Bench Motivation

For the above reasons, it is clear that there are very few viable options for executing malware and capturing traffic without real world security implications and ethical issues. For example, the most feasable way to capture exfiltration traffic is to purposely infect a machine that has access to the internet, and to leave the system unattended, while capturing traffic, for a lengthy period of time. This experiment alone is difficult to execute correctly as there can be no prediction of when exfiltration will occur, or if the machine is waiting for instructions from a command and control channel before continuing. The sheer volume of the traffic that must be inspected for the data exfiltration samples will also prove to be a difficulty. On top of these difficulties, there are also obvious ethical and legal implications to purposly infecting machines and letting them proliferate, possibly infecting more machines. Currently, the most feasible option is to simulate this traffic in a controlled setting, allowing for rapid prototyping of detection algorithms.

Chapter 3

TAXONOMY OF EXFILTRATION METHODS

In order to better understand how to defend against data exfiltration, a taxonomy of common methods has been assembled and examined. The majority of these have legitimate uses and are hard to detect among normal network traffic. There are other methods that are an anomalous use of a protocol that may be harder to detect, but provide a small throughput rate. It is important to note that data can be exfiltrated in a nearly infinite number of ways and that the methods listed below are simply the most feasible and most likely to be used. Each method has been given a qualitative rating regarding how much transfer bandwidth is feasible using that method and how unlikely it is that the channel will be discovered by both detection tools and an observing administrator. Ratings can range from 1 having the least amount of available bandwidth or a high likelyhood to be detected to 5 having the most amount of bandwidth or a very low likelyhood of being discovered.

3.1 Bandwidth

For the purposes of this research, bandwidth is regarded as the transfer speed available so that the file (in this case, the data to be exfiltrated) transfers completely and correctly with a relatively low degree of errors occuring. Consider, for example, using Domain Name System (DNS)[17] requests as a way to exfiltrate data[1]. In this scenario, an infected machine will make a large volume of DNS requests to a given subdomain. For example, the client will request a DNS lookup for data-to-be-exfiltrated.example.com. In this case, the host example.com will be queried for

Table 3.1: Taxonomy of exfiltration methods rated in terms of bandwidth and covertness

Name	Protocol	Encryption	Bandwidth	Covertness
File Transfer	FTP	Yes	5	3
Secure Copy	RCP/SSH	Always	5	2
HTTP POST	HTTP	No	5	4
SSL HTTP	HTTPS	Always	5	5
Email	SMTP	No	5	4
SSH Tunnel	SSH	Always	5	2
Instant Message	IRC/XMPP	Yes	3	4
Echo Request	ICMP	No	2	4
DNS Tunnel	DNS	No	1	2
Update Comm.	HTTP/SOAP	No	5	3
P2P	TCP	No	5	2
Custom IP	IP	No	3	4
Custom TCP	TCP	No	5	3
Custom UDP	UDP	No	4	2

the resolution of the address, to which it will reply that it does not exist. However, at this point, the ”datatobeexfiltrated” string has been sent to the receiving server. While this method is arguably hard to detect, the amount of bandwidth that is available to it is relatively small due to the nature of DNS requests and the length of the subdomain that can be requested.

3.2 Covertness

Covertness can be described as the likeliness of the method being discovered. There are some methods that will be generate a considerably larger footprint on a network in regards to how out of place that type of traffic is on a network. Consider an average computer user who does nothing more on a network than basic browsing and email. Now consider an attacker has infiltrated the machine and has gathered data to be exfiltrated using File Transfer Protocol (FTP)[25] or Secure Shell (SSH)[29]r. It is unlikely that these protocols would be used by the average user and therefore it is likely that these methods would be detected easily by any behavioral detection applictions.

3.3 Method Descriptions

It is important to note that the taxonomy table demonstrates encryption on a channel based level. Encryption built into the channel will hide most details of the transfer including the file contents themselves. However, encryption can be included in all methods by simply encrypting or compressing the file before transfer. While this is not as protected as encrypting the entire channel, for the purposes of exfiltration, it is enough to prevent inspection algorithms from examining the contents of files. Compression will also be a much wanted feature of exfiltration in that it shortens the transfer time and makes the transfer less likely to be detected. The majority of the methods described below can usually be implemented easily using an existing library. However, including an external library as a payload to malware can be cumbersome by greatly increasing the executable size. For these reasons, it can be assumed that an attacker will code all methods using standard libraries that are installed by default on the infected machine. Sockets are standard on most targeted machines and will likely be the tool of choice by attackers.

3.3.1 Sample Method

3.3.1.1 Description

In order to accurately describe methods used to exfiltrate data, certain terms and scenarios must be defined. In all cases of exfiltration, there is an ultimate transfer of information from a *target* machine to an attacker's machine. This act of transfer outgoing from the target machine back into a machine in control of the attacker is known as *exfiltration*. An *attacker* is the individual who is receiving the exfiltrated data in hopes of using it for financial or political gain. The attacker can exfiltrate the data *automatically*, through the use of *malware*, or *manually*, by some remote code execution exploit. It is important to recognize that the automatic methods must be incredibly robust as almost all of these methods can fail depending on the network environment. For this reason, it is likely that if the attacker is exfiltrating data automatically, he or she will use methods most likely not to fail and may even have back up methods ready in case of failure states. An attacker exfiltrating data manually has much more flexibility as reconnaissance can be done and the method least likely to be detected can be used. The attacker will exfiltrate data by means of a *channel*, or the concept of an abstract connection. For both manual and automatic exfiltration, the method chosen for exfiltration is likely to be well known and well used channel that has legitimate uses.

3.3.1.2 To Exfiltrate

In order to exfiltrate data, the attacker's code must be executed on the target machine. This can take the form of a program executed automatically by malware or manually by an attacker, or on a channel set up by an attacker through the use of a remote shell. There are certain mechanisms (e.g. common dynamic link libraries) built-in to most consumer systems that allow attackers to have a toolset that is likely already installed. For the more sophisticated methods, some external libraries may be loaded or they may be built manually. An attacker will connect to a server that is using the same exfiltration method, transfer the data, and remove evidence of the transaction.

To receive this data, the attacker must have agreed upon some method that is used by both the target client and the server. The server will listen using the designated protocol and save the data as it is received. The attacker may choose to take the server offline after the data has been transmitted as a means to minimize detection. Since the server is usually under the control of the attacker, he or she has complete control to utilize whatever libraries that are needed, allowing the server to be as complex as needed.

3.3.1.3 To Detect Exfiltration

For most methods, there are telling signs of their use on the network. Often, these methods can be detected without the use of deep inspection of a packet's contents. The challenge that exists is not the detection of the method, but rather the detection of the method with malicious intent. This research attempts to highlight some of the details surrounding data exfiltration and allow the inspection of these methods to show certain characteristics or patterns that can be used to detect or intercept suspicious traffic. Having illustrated the template for an exfiltration method, we will now outline the methods found in this taxonomy.

3.3.2 File Transfer

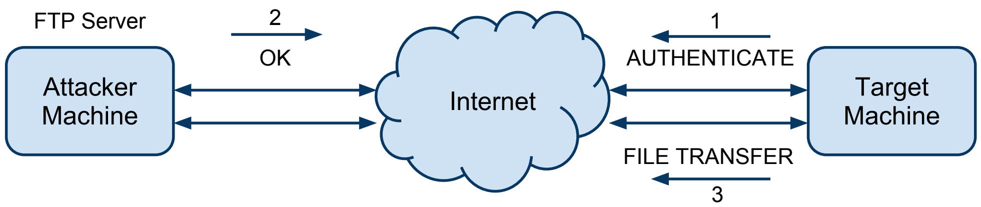

Figure 3.1: Exfiltration using File Transfer Protocol (FTP)

3.3.2.1 Description - File Transfer

File transfer will use the FTP protocol[25] (which relies on Transmission Control Protocol (TCP)[15] for both a transfer channel and an out-of-band command channel) in order to transmit the sensitive data to a server, which will be running an FTP server. The FTP protocol involves an authentication step, followed by the transfer of the file, followed by a closing of the connection. This method uses an out-of-band channel for communications regarding the file name and file size. FTP also has the ability to decide which side of the connection will act as the client or connection initiator. This provides a slight advantage over other methods in that it allows an attacker to choose which way the connection will start, allowing some flexibility to bypass certain target-side firewall restrictions.

3.3.2.2 To Exfiltrate - File Transfer

For an infected machine to exfiltrate using this method, the data typically is encapsulated in some kind of binary format(possibly compressed or encrypted) and the client must attempt to authenticate with the server. The server's address is either hard-coded or dynamically determined[13]. The authentication details such as the user name and password must also be static or dynamically determined. The alternative is that anonymous login can be used, however this may prove to be a security issue as it will allow anyone to access the attacker's server. The FTP protocol can be implemented by hand in C using sockets or using an existing library such as CURL[2]. Associated data will be sent over a TCP channel. Standard file transfer using FTP allows the port of the receiving server to be specified by the server. Therefore, it is perfectly valid for FTP to transfer the file using non-standard ports for both malicious and non-malicious purposes.

To receive this data, the attacker must install and configure a standard FTP server service with corresponding usernames and passwords that will either be hard-coded into the client side or dynamically determined. Additional processing can be done on the server side to organize the data or to uniquely determine which host sent the data. The attacker may choose to only have the server accepting connections during certain times that exfiltration is expected. This may reduce the chance of detection if the server is no longer available after detection.

3.3.2.3 To Detect Exfiltration - File Transfer

Examining packet headers for FTP traffic to servers that have not been approved during previous use (e.g. whitelisting) may be the safest course of action.

The user can be prompted for FTP connections to previously unheard of servers, much like SSH does if the user does not have the SSL key. This method, however, will provide an obstacle to the user if he or she legitimately uses FTP. Alternative methods include packet examination for restricted REGular EXpressions (REGEX) patterns or alerting the use of any transfer of encrypted data over unencrypted FTP channel. A white list approach can be taken on only servers that use TLS[8] encryption, therefore reducing the amount of servers that the user must approve.

3.3.3 Secure Copy

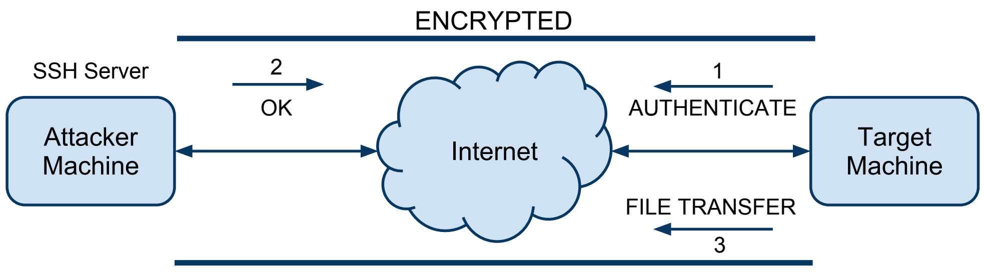

Figure 3.2: Exfiltration using Secure Copy

3.3.3.1 Description - Secure Copy

The secure copy (SCP) method uses RCP (remote copy) in combination with SSH in order to provide a copy method that is encrypted and authenticated. RCP uses a TCP channel provided by an SSH tunnel and provides a series of protocol messages that allow the transfer of files and some file attributes. SSH is a protocol that provides encryption of the entire channel, hiding the internal protocol messages that RCP may produce, further limiting the areas that detection algorithms may inspect.

3.3.3.2 To Exfiltrate - Secure Copy

The client must initiate an SSH session to a remote server using hard-coded or dynamically determined username and password combination. Once authenticated with the server, transfer the file of encapsulated sensitive information using the RCP protocol. The connection is closed once the transfer is complete. This can be implemented using the openSSL library[3] as well as using the sockets interface.

A server is needed that has installed and properly configured an SSH service with corresponding username and password combinations. The attacker may choose to use a non-standard port to avoid suspicion. For a situation where an attacker is manually exfiltrating data, the attacker may also choose to ignore requests from all other machines besides the machine exfiltrating.

3.3.3.3 To Detect Exfiltration - Secure Copy

Detecting exfiltration using this method is particularly difficult due to the fact that the entire transfer channel is encrypted. The most feasible method to detect this type of exfiltration is to examine packet headers and packets for SSH authentication attempts to non-trusted servers through the use of a whitelist. Alternatively, the SSH protocol could be limited by a network adminstrator to only work to trusted servers. For average computer users, the SSH protocol will most likely not be used legitimately and this type of traffic will be observed easily.

3.3.4 HTTP POST

3.3.4.1 Description - HTTP POST

HTTP POST is a method in which an HTML object contains a form and a POST action. This method utilizes the Hypertext Transport Protocol (HTTP)[11] to exfiltrate data. The POST action will send the specified text in the web form to the server specified by HTML. This method is unencrypted and originally was not meant for very large amounts of data. Later, there were modifications made to this

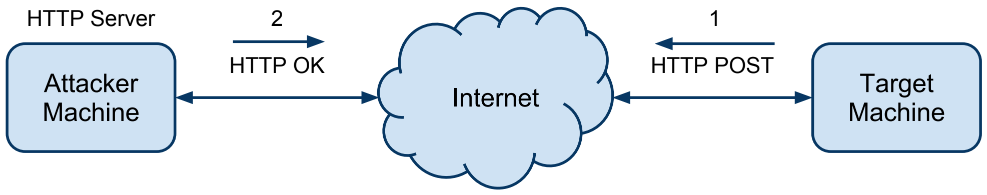

Figure 3.3: Exfiltration using an HTTP POST action

method to accommodate file uploads to web servers[19]. Data can now be encoded using a new encoding specifically for this type of transfer (multipart/form-data). The server will write the received data according to the corresponding ACTION URL.

3.3.4.2 To Exfiltrate - HTTP POST

The packets necessary to submit a POST can be crafted easily using TCP sockets. Multiple POST requests may be needed due to the inefficiencies of transferring large amounts of data in a single POST. The POST requests may also be split up in order to behave more like with existing POST requests. Note that the majority of legitimate web applications that accept file uploads have a limit, usually on an order of megabytes. This must be considered by the attacker and may also be used by the defender as a means to detect exfiltration.

A properly configured HTTP server can handle the expected POST request, or a service can emulate the correct response to the POST using simple sockets. The latter method will be able to better hide its identity by only responding to POST requests and ignoring GET requests. Since there is no authentication built-in to this method, some modifications may be necessary in order to uniquely identify exfiltrated data to a host machine.

3.3.4.3 To Detect Exfiltration - HTTP POST

Exfiltration of data will presumably use a POST submission without requesting data from a site initially. For example, the typical POST usage involves the client requesting an HTML object from a server, followed by the client submitting a POST request. Exfiltration will most likely skip the first step, to minimize unnecessary traffic and may be a flag that can be detected as suspicious traffic. POST requests to servers which have not been seen before can also be marked suspicious. If the server is not using SSL[27], the contents of the request can also be examined to see if any restricted keywords or regular expressions were present.

3.3.5 SSL HTTP

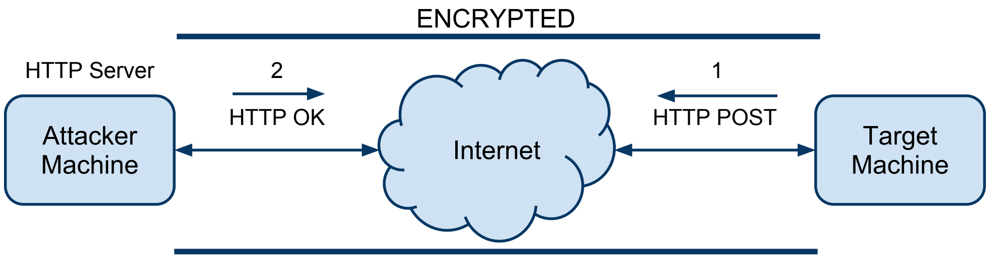

Figure 3.4: Exfiltration using an HTTP POST action in an SSL encrypted channel

3.3.5.1 Description - SSL HTTP

SSL HTTP is simply using the HTTP protocol in an encrypted channel that is facilitated by SSL[27]. A public/private key pair is used to safely exchange a symmetric key which is used for encryption of both sides of the communication. The most feasible method of exfiltration from within this channel is to use HTTP POST. As such, much of the implementation details are similar to that of HTTP POST but resulting in a much more secure connection that will be significantly harder to detect.

3.3.5.2 To Exfiltrate - SSL HTTP

This type of exfiltration will be nearly identical to an HTTP POST submission, where a file upload request is sent to a determined server with the proper attributes. An SSL handshake must be initiated first, which can be accomplished with standard encryption algorithms and standard sockets. The data to be exfiltrated is assumed to be encapsulated in some form and most likely compressed.

Similar to what is needed to accept HTTP POST requests, all that is needed to receive this type of exfiltration is a properly installed and configured HTTP server or service emulation. Additional configuration will be needed, specifically in terms of a valid certificate. There are various options at this point, such as legitimately buying a certificate that can be validated by a third party, providing more legitimacy to the connection or to use a self-signed certificate that will not cost money but may appear suspicious.

3.3.5.3 To Detect Exfiltration - SSL HTTP

This type of exfiltration maybe the most difficult to detect for a multitude of reasons. This type of traffic will likely be one of the most common types of traffic on all consumer machines allowing exfiltration to hide among legitimate traffic. Clients may also connect to a multitude of different HTTPS servers making the use of a white list system cumbersome. Due to the inherent encryption of this traffic, it cannot be inspected. For these reasons, the only feasible method of detection is by using some kind of behavioral system that will detect anomalies that stray from normal traffic patterns[9].

3.3.6 Email

3.3.6.1 Description - Email

The use of Email to transmit files will use Simple Mail Transfer Protocol (SMTP)[24]. This method will attach a file of sensitive data and attempt to transmit

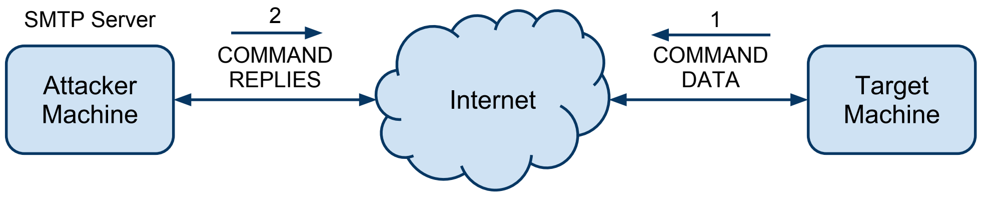

Figure 3.5: Exfiltration using an SMTP server

it via Email to the server. Two methods of exfiltration exist for this protocol: using the user's existing mail server and Email address and connecting to an arbitary mail server to transmit data. Additional methods are possible, but significantly more difficult to implement. An example would be a program that logs into a webmail service and exfiltrates data accordingly. This method is much more likely to be used by an attacker manually exfiltrating data.

3.3.6.2 To Exfiltrate - Email

One method the client can use is to use the user's current mail server and email address, however the exfiltration malware must know the user's credentials and the user must have an existing email address, which is unreliable. The alternative method is to use an existing mail server to transmit the data. A socket connection can be implemented to utilize a mail server that forwards the data as an attachment.

A properly installed and configured mail server must exist to relay the exfiltrated data. A receiving email address may also needed to receive the data. In the case of the attacker connecting directly to an existing mail server, a receiving email address may not be needed as the data can be parsed directly from the mail server.

3.3.6.3 To Detect Exfiltration - Email

Due to the unreliable nature of using the first exfiltration method listed above, the malware will most likely use an arbitrary mail server. Since the typical user will only regularly connect to a small number of mail servers, detecting the connection to a previously unheard of mail server will be trivial. Additional detection methods may attempt to inspect data for restricted keywords or regular expressions.

3.3.7 SSH Tunnel

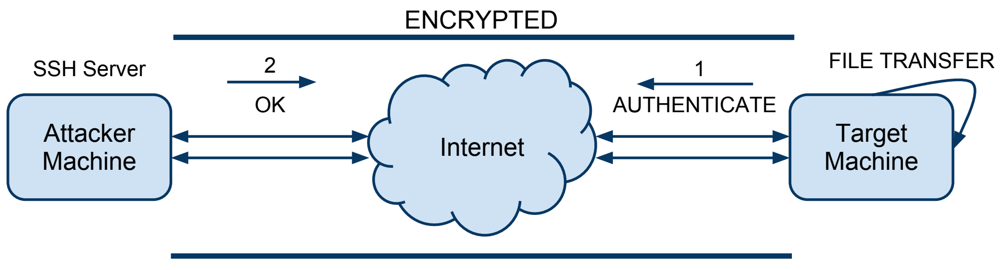

Figure 3.6: Exfiltration using an SSH tunnel set up as a local proxy

3.3.7.1 Description - SSH Tunnel

An SSH tunnel will work by encrypting a channel using an SSH session. Traffic will be tunneled to the SSH host through the use of a local proxy. The data channel will be encrypted and closed when finished. This allows nearly any underlying protocol to be used through an encrypted channel which makes it very appealing to an attacker.

3.3.7.2 To Exfiltrate - SSH Tunnel

The first step to creating an SSH tunnel is to initiate an SSH session with a remote host. An existing library can be used (openSSL) or the secure handshake can be coded using sockets. A proxy will then be initiated over SSH and traffic will be sent to that local proxy on the specified port.

Like secure copy, the only requirements for accepting this type of data would be a properly configured SSH service. Some configuration must be agreed upon beforehand, such as the user name and password to be used.

3.3.7.3 To Detect Exfiltration - SSH Tunnel

There will be two key events that will flag suspicious activity. The first event will the connection to a remote SSH server, which the typical user will only do rarely. A white list can be used in this case to notify the user. The other event will be the use of a local proxy. Any packets sent to 127.0.0.1 will hint at the existence of a local proxy. While it may seem advantageous to use the flexible nature of SSH tunnels, they may be too likely to be detected for an attacker to use.

3.3.8 Instant Message

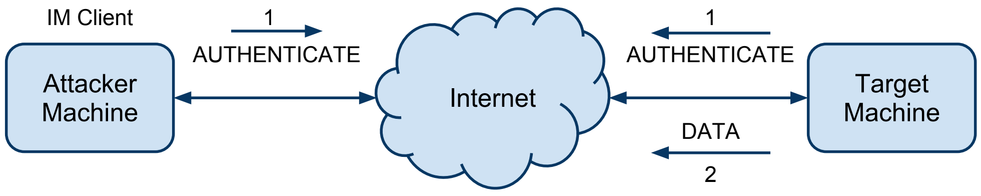

Figure 3.7: Exfiltration using Instant Messaging clients

3.3.8.1 Description - Instant Message

Instant messages can be used to transmit small amounts of data using a number of different overlay protocols. The most common protocols that exist currently are Internet Relay Chat (IRC)[21] and Extensible Messaging and Presence Protocol (XMPP)[22]. Both require authenticating with a corresponding server. Two clients can authenticate with the same server and messages can be passed between the two clients. IRC has been one of the most popular ways for a botnet to communicate with a command and control channel[28] due to the third-party nature of the server and the ease of communication.

3.3.8.2 To Exfiltrate - Instant Message

The attacker must authenticate with a determined server using either the XMPP or IRC protocol. Messages will be passed according to the corresponding protocol. The data must be split up to accommodate message length limits and may have to be encoded if messages may not contain binary data.

To accept this data, a server will be similar to an implemented client. The channel that is established here is similar to two individuals meeting at a specific location and communicating accordingly.

3.3.8.3 To Detect Exfiltration - Instant Message

A white list will help significantly to detect this type of exfiltration as the amount of XMPP or IRC servers the average user connects to would be relatively small. A large volume of messages being transferred may also be a sign of exfiltration as a legitimate user likely will send messages at a much smaller frequency than a machine exfiltrating data.

3.3.9 Echo Request

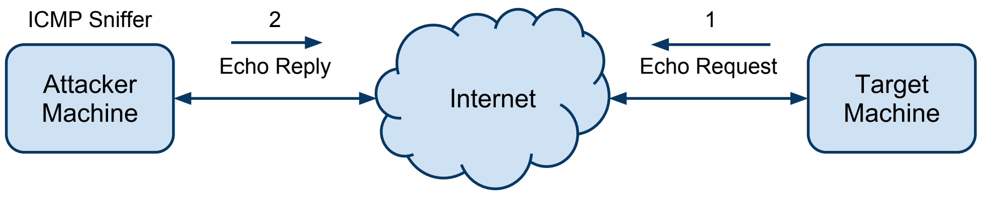

Figure 3.8: Exfiltration using ICMP Echo Request packets

3.3.9.1 Description - Echo Request

An Internet Control Message Protocol (ICMP)[7] echo request packet is often sent by a user to determine if a remote host is available. The host receiving the echo request packet will respond with an echo response with the packet payload being identical to the packet payload received. If the client does not wish to receive a reply, the exfiltration could be further obscured by crafting the echo request packet to have an incorrect source address. One of the overlooked details about this transaction is that the contents of the ping packet are rarely inspected as they are largely irrelevant to determining if a host is available or not. For this reason, the packet payload can contain data to be exfiltrated and the receiving host can simply record the payloads of all ICMP echo request packets it receives.

3.3.9.2 To Exfiltrate - Echo Request

Raw sockets can be used to implement an ICMP echo request fairly easily. There is some flexibility in terms of setting the payload size. Standard echo request packets are 56 bytes, however they can be larger. Making the packet size larger will likely make this method easier to detect. In this case, the attacker must balance the packet size against transfer bandwidth so as to not get detected. The data to be exfiltrated must be encapsulated and split up accordingly. The attacker may also choose to implement some feature to ensure a reliable channel, as these packets are prone to arriving out of order, or not at all.

The receiving system will accept echo request packets, record their payloads and piece together the data accordingly. The server may choose to respond with echo reply packets, in an attempt to look more authentic, however it is not necessary and may do nothing more than produce more noise on the network.

3.3.9.3 To Detect Exfiltration - Echo Request

The main downside to using ICMP echo request packets to exfiltrate data is that there will be a large amount of requests on a network where typical usage will be very little, if any. Detection algorithms may also inspect ICMP packets for keywords or regular expressions. Packets with high entropy[10] may signify the existence of exfiltrated data as the payload should not contain highly varying data.

3.3.10 DNS Tunnel

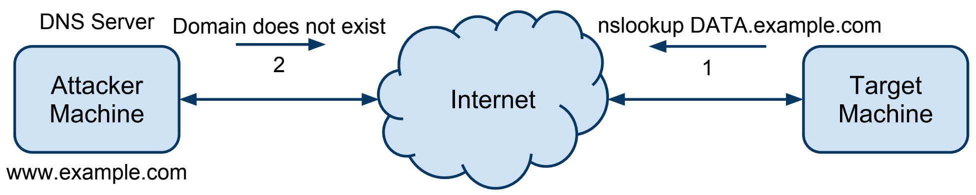

Figure 3.9: Exfiltration using DNS lookup requests

3.3.10.1 Description - DNS Tunnel

DNS requests work by sending a hostname string to a DNS server. The server will either respond with the corresponding IP address or that the requested domain name was not found. In the case of subdomains of hosts, the DNS server or client may request from the known Top Level Domain (TLD) the identify of the subdomain. This aspect of DNS makes the transfer of data via DNS possible. As described earlier, the infected machine will request a subdomain of the attacker's TLD. The TLD will respond that the subdomain does not exist and will record the string as data that has been exfiltrated.

3.3.10.2 To Exfiltrate - DNS Tunnel

An attacker must simply send UDP DNS requests for subdomains to the determined domain with data in the subdomain portion of the domain. The data must be divided into small enough chunks so that they can be transmitted in this manner. The data must also be encoded to support requirements that disallow binary data in subdomain strings.

A specified domain must be owned by the attacker that has DNS request handling capabilies is needed to receive this type of exfiltration. The server must record all DNS requests and piece the data back together accordingly with decoding if it was used.

3.3.10.3 To Detect Exfiltration - DNS Tunnel

The most telling sign of exfiltration in this manner will be repeated and complex DNS requests to the same domain. The examination of the number of DNS requests in any period of time will likely uncover this method. Due to how noisy this method is and how little bandwidth is available to it, this method is not likely to be used for large scale data exfiltration.

3.3.11 Update Communications

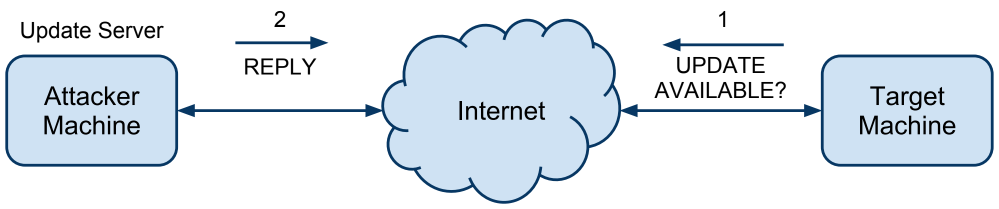

Figure 3.10: Exfiltration using update communications

3.3.11.1 Description - Update Communications

There are a large amount of client applications that periodically update automatically be contacting a determined server. Some applications or operating systems may even download updates automatically. Exfiltration in this manner will most likely utilize the method of contacting a home server to ask for updates. In order to determine if an update is needed, some data must be initially sent to the server. An attacker could emulate this method to appear as an application update.

3.3.11.2 To Exfiltrate - Update Communications

An existing protocol that asks for updates must be examined (e.g. Windows Update) in order to properly implement the client. An update request will be crafted to mimic this type of action while containing exfiltration data rather than an actual request. The destination will be a target owned by the attacker.

On the receiving side, a custom service must be designed and implemented to record the update requests and parse the data. Protocol messages that ensure proper and valid delivery may be used, but will increase the chances of the method being detected. Some update services must transmit a complex state of the system or application and in some cases can cause the amount of outgoing traffic to be very large.

3.3.11.3 To Detect Exfiltration - Update Communications

The number of applications that ask for updates on the client can be aggregated and their known servers can be added to a white list. Even if the traffic looks identical to application update traffic, if it is contacting an alternative server, an alarm can be set. Additional servers must be approved by the user or administrator.

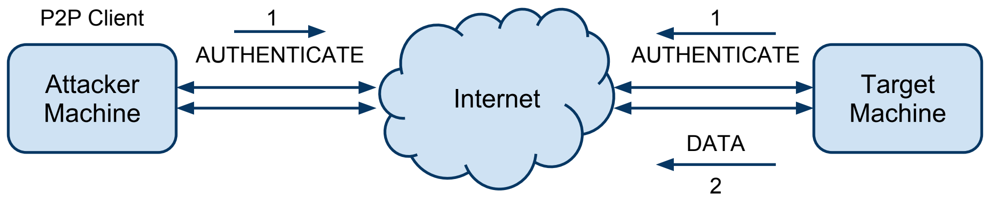

Figure 3.11: Exfiltration using a third-party P2P server to connect clients

3.3.12 P2P

3.3.12.1 Description - P2P

Peer-2-Peer (P2P) traffic can be described as using a centrally located server for each client to communicate with each other. Once the clients have found each they, they will initiate a direct connection and communicate accordingly. If a user actively uses P2P applications, specifically legitimate ones such as those used for software distribution, there will be significantly more places for exfiltration traffic to hide.

3.3.12.2 To Exfiltrate - P2P

A client will contact a third party server as a means of meeting the receiving client. This third party server can be an IRC server or writable web server, for example. Once the clients have communicated, they can open a communication channel and transfer data.

In this method, the server will be nearly identical to the client. The server will likely have open ports so that the sending side can initiate connections, bypassing firewalls in the process.

3.3.12.3 To Detect Exfiltration - P2P

If there is no current P2P traffic existing on the client, than the detection of this type of traffic will be trivial. However, among other P2P traffic, picking out suspicious activity will be extremely difficult. The traffic must be examined closely in an attempt to discover connections that were not initiated by the user.

3.3.13 Custom IP

3.3.13.1 Description - Custom IP

The Internet Protocol (IP)[14] is the basis of most transfers between two machines across the internet and can be used without a transport protocol. Some protocol messages must be implemented in order to support accurate and reliable transfer. This is the one of the most minimal methods and therefore extensive care must be taken to ensure proper delivery.

3.3.13.2 To Exfiltrate - Custom IP

Simple sockets without specifying a transport layer protocol can be used to implement the client side of this method. Additional data must be included to ensure proper delivery. This type of exfiltration will presumably be one of the most difficult versions to implement and may not provide as much covertness as expected. It may stick out due to it being an unheard of protocol. On the other hand, dissection of protocols must be implemented into a detection algorithm and therefore being initially unable to dissect this protocol may provide an advantage.

To receive data, a service must be designed and created that matches the client exfiltration protocol. Special care must be taken to implement a proper reliable transfer. Additional issues may arise due to improper routing or routers that will not forward packets using an unheard of transport layer protocol.

3.3.13.3 To Detect Exfiltration - Custom IP

The IP header of each packet can be examined for the corresponding transport layer flag. Unknown flags or flags that do not match the packet content can be marked as suspicious. There should be little or no practical uses of IP to transfer data without using an existing transport layer protocol.

3.3.14 Custom TCP

3.3.14.1 Description - Custom TCP

The Transmission Control Protocol (TCP)[15] can be used without needing any overlaying or existing protocol. TCP can be used to simply transmit messages to and from a server as needed, inventing protocol messages as needed. This is essentially what FTP uses to transfer files, but without using a command channel. It would be more efficient to know the file size and other attributes of the file before receiving, however in reality, it is not needed. A server could listen for a client to connect on a determined port and assume the file is complete when the client closes the connection.

3.3.14.2 To Exfiltrate - Custom TCP

Simple sockets can be used to transmit the exfiltrated data. Acknowledgement and initialization messages may be used in order to ensure data is sent correctly and fully. The file may be split up depending on how much data needs to be exfiltrated in order to facilitate some kind of protection in case the transfer fails. This may be one of the most covert methods due to the sheer volume of legitimate network traffic that utilizes TCP.

A specific service will be designed and created to match the transfer sequence of the client. A previously agreed upon set of protocol messages can be used to ensure proper exfiltration. For multiple infected machines, the server must find some way to identify different hosts. Data that is sent manually by an attacker may signal the server to shutdown after receiving the data, as to limit investigative features.

3.3.14.3 To Detect Exfiltration - Custom TCP

Detecting this type of exfiltration will be difficult due to the majority of other TCP traffic that exists on a network. Signs to look for include the use of non-standard ports or ports to which the packets do not match the corresponding well known service. A white list can be used in combination with an examination of the packet headers and data to ensure that data sent to a specific port matches the correct application protocol. A combination of strategies must be used to properly detect suspicious activity of this nature.

3.3.15 Custom UDP

3.3.15.1 Description - Custom UDP

Like TCP, the User Datagram Protocol (UDP)[23] can be used to send sensitive data without utilizing any existing application layer protocols. Unlike TCP, UDP does not have any features to ensure a reliable channel. The client-server pair must account for obstacles such as duplicate packets, missing packets, and out of order packets.

3.3.15.2 To Exfiltrate - Custom UDP

Simple sockets can be used to send UDP packets. Care must be taken in order to add proper order and reliable transport to ensure data is received correctly and completely. Without some internal error recovery, this method may only be viable for small amounts of data.

A custom UDP service must be designed and created to properly handle the client side transfer. Custom protocol messages may be implemented to ensure proper delivery. It is strongly recommended that a UDP server find some way to validate that the file has transferred correctly due to the lossy nature of UDP.

3.3.15.3 To Detect Exfiltration

Unlike TCP, there are far fewer services that use the UDP protocol and therefore the examination of all UDP traffic may be feasible. This examination in combination with well known port numbers and a white list will be effective in detecting UDP data exfiltration.

Chapter 4

DESIGN AND IMPLEMENTATION

The exfiltration test bench (*extb*) suite was designed as a flexible framework to initiate many different types of exfiltration. It allows a user to have the identical program and corresponding plug-in on both a client and a server machine. The server (or receiver) machine listens using the specified method and port (if needed). The client then initiates a connection to the designated server and transmits the specified file accordingly. The server receives the file, saves the data to disk, and closes immediately after. The program was designed to be lightweight, robust, and easy to use. Python[4] was used as the programming language for this framework due to its rapid prototyping abilities and large library base which allow plug-ins to be programmed quickly and efficiently, requiring little code. A block diagram illustrating the design of the program can be found in figure 4.1.

4.1 Initial Design

The original design involved an auxiliary command channel which would confirm the method, file size, and some kind of verification that the file transferred completely and correctly. Later in the design process, this command channel was removed in order emulate an authentic environment. In a real world scenario, a malware author would have to find some alternative way to specify the size of the file and other attributes necessary for correct and complete file transfer. This is particularly important for nonstandard protocols or protocols that were not intentionally meant for file transfer. As an example, consider a protocol like FTP which

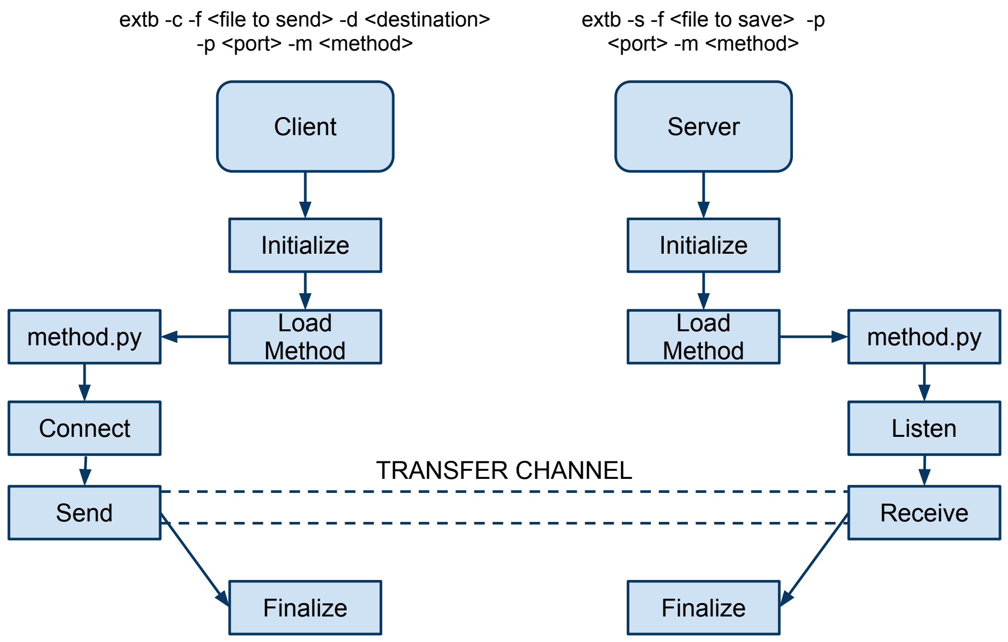

Figure 4.1: Block diagram of the extb test bench utility

was designed for file transfer. In this case, the file size is transmitted in the command channel. A separate TCP channel is opened up for the actual transfer of the file and the sending side closes the connection to signify the end of the transfer. This process allows the receiving system to verify with some degree of accuracy that the file transferred completely and correctly. On the other hand, consider a covert channel such as using ICMP packets that look like an echo request, but actually carry exfiltrated traffic. Due to the small packet size, many of these packets must be sent for a moderately sized file to be transmitted. With no reliable data channel features (such as those implemented with TCP) there will be a high probability of errors occurring during transfer. The attacker must consider these attributes when designing malware and may have to balance the covertness of the channel versus the reliability and bandwidth of the channel.

4.2 Program Architecture

As seen in figure 4.1, an identical program having both sending and receiving functions with an identical method library is loaded on two seperate machines. The server will be started first in order to eliminate race conditions. This is due to the fact that the client will transmit data without waiting for acknowledgement and in some cases will still send packets even if the destination is not listening. After program initialization, the server will load the external library specified on the command line. This library will handle all receiving aspects of the method. After receiving and recording the data sent, the server will return control to the main program which will take care of any needed finalization. The client works in an almost identical manner except that the client section of the library code is called rather than the server section. Some configuration options are passed over the command line such as the port number and destination options. In future work, these options may be expanded to allow configuration of per method attributes such as packet size or acknowledgement options.

4.3 Plug-in Design

In order to implement a plug-in for this framework, the method must be programmed using the Python language. The advantage to using this language is that there are a very large number of external libraries that have implemented many major protocols. As opposed to a language like C, Python will allow the user to ignore some of the specific details of programming a method. For example, consider the HTTP POST method. In C, this would likely have to be programmed using sockets and obstacles such as socket creation, transmission buffering and framing would have to be overcome before the programmer could even begin the actual exfiltration. In Python, libraries for these functions already exist and provide the user the flexibility to avoid these extraneous details. Additionally, if low-level control is needed, Python allows C code to be embedded inside its environment, allowing fine grain control over the exfiltration method.

In order to properly design a plug-in, two functions need to be defined: server and client. These functions will handle all aspects of their corresponding job. For example, the client will handle reading the file to be sent, breaking the file into appropriately sized blocks, loading any external libraries needed, opening the connection, transmitting the file, and closing or terminating the connection. The framework has been designed so that the plug-in has a large amount of flexibility and is therefore not hindered by the framework itself.

Chapter 5

RESULTS AND ANALYSIS

The test bench described in Chapter 4 was used as a means to investigate the methodology that malware authors would use when choosing how to exfiltrated sensitive data. The test suite proved useful in demonstrating various characteristics that were common to all methods that may be used to either develop further exfiltration methods as well as improving current detection algorithms. In order to demonstrate the usefulness and capabilities of different techniques, each exfiltration method was used to exfiltrate a file of constant size (500MB) on a loopback interface, in order to minimize errors due to packet loss. Each method was measured using the time command in order to show how long the program took to finish, how much computation was needed at the user level, and how much computation was used as the kernel level. The results of this experiment can be found in table 5.1. It is important to note that due to the file size used and the reliability issues associated with using the ping method to transmit a file, a reading for ping could not accurately be measured.

5.1 Time Measurements

The time that each method took to complete can be thought of as the available native throughput of the method. There are some characteristics of the method that will reduce this throughput rate of each method, typically due to the use of methods that were not designed to transfer such large amounts of data. Since UDP has no built-in reliability features, the attacker must choose whether to transfer

Table 5.1: Measurement of method time, user time, and system time

Method	Real Time (s)	User Time (s)	System Time (s)
http	45.404	3.316	8.025
https	96.060	69.552	6.944
ftp	26.385	1.852	3.396
ftps	35.772	31.750	2.404
tcp	17.479	7.288	4.428
udp	58.227	33.482	21.633
ping	-	-	-

data with additional acknowledgement and retransmission features added or to simply hope that the file transfers correctly and to retransmit the entire file if an error occurs. Other methods that use encryption on the actual channel must also allow time for each system to encrypt and decrypt messages before transmitting them over the channel, slowing down the throughput of that file transfer.

5.2 Reliability

In the course of testing each of these methods, it also became apparent that some of these methods were only reliable to a certain degree. UDP, for example failed to transmit the 500MB file correctly in some cases. This can presumably be attributed to the fact that packets arrived out of order or too fast for the server program to properly accept them. The ICMP method also proved to have difficulty sending files as small as 1MB. However, it is important to keep in mind that not all data exfiltrated is identical. There are situations where reliability may not be completely needed and the loss of some data during transfer can be acceptable. For example, the ICMP method would be ideal for sniffing keystrokes. The malware or attacker would log all key strokes and periodically exfiltrate the data. A large amount of useful data can be extracted from a key logger such as passwords, identifying information, and encryption keys. This will also prove useful to the attacker if he or she is searching for a specific piece of information that is not accessible on the machine. As another scenario, suppose the user does not save passwords or credit card information on the computer. The only way to capture this information would be as the user types it. Working in tandem with the ICMP method, after sniffing the necessary information, the password can be sent as a normal-sized ping packet with a very small likelihood of being detected. If by chance the necessary packet failed to transfer correctly, the attacker could simply wait until the next instance of the user typing in the desired piece of information.

5.3 Current State of Detection Algorithms

By testing the methods implemented on existing detection algorithms, it is likely that the majority of these exfiltration attempts would succeed without being detected. Most detection suites work using the same set of rules[26] described below.

5.3.1 Sensitive File Flagging

These detection suites will allow the administrator to select certain files that are sensitive and may not exit the network unless given the proper permissions. This technique is largely aimed at the accidental exfiltration of certain files or at the most primitive and unsophisticated attacker. It is trivial to bypass this check by simply changing the file name, a small amount of the file contents (to alter hashing sums), or by simply compressing or encrypting the data. Since the attacker is likely to use compression just to minimize the amount of data transferred, it is likely the attacker will defeat this method without even trying.

5.3.2 Keyword Matching

The administrator may also be given the option to specify keywords that may not exist in data exiting the network. Often these keywords will exist in sensitive documents and may prevent an attacker from exfiltrating data even if the file contents or file names are modified slightly, however, it is evident that this method is also aimed at a user accidentally exfiltrating data. Often, attackers will scrape files for the sensitive information and compile it into a separate file, eliminating the keywords that would exit the system. This method also can do nothing against attackers that compress or encrypt the file before exfiltration.

5.3.3 Regular Expression Matching

Like keyword matching, the administrator can set certain regular expressions that would prevent the exfiltration of data that matches the expression. A common example of this would be an expression such as [0-9]3\-[0-9]2\-[0-9]4 which would match social security numbers. This expression roughly translates into any string of numbers in the form of 000-00-0000 where 0 can be any single digit number. This is slightly better than keyword matching as the actual sensitive data can be matched rather than keywords that can be safely deleted from the file. This method will only work, however, if the file can be inspected which excludes cases where the file is compressed or encrypted.

5.3.4 Blacklists/Whitelists

Some detection applications also allow the administrator to white list or black list certain destinations or IP addresses. This method is only effective if the malicious IP address is known, which would not happen until after exfiltration has occurred. Even for most botnets, if the source IP address is determined and distributed, it will not prevent exfiltration through other members of the botnet.

5.4 Current Detection Algorithm Results

Using information from feature lists and data sheets of well known detection suites, it is clear that even some of the most primitive methods would succeed without detection. All methods that transmit over an encrypted channel would not be able to be inspected by these detection suites. Some of the detection suites cannot understand every protocol and would not even attempt to inspect the contents of ICMP packets. It is clear that these detection suites were originally designed to prevent accidental data exfiltration but provide little to no protection against the moderately equipped attacker or malware.

5.5 Common Characteristics

By examining various methods of exfiltration, there were certain characteristics that were evident in each technique. It is clear that there are certain states that the exfiltration process must go through before the transfer can be considered complete and successful. These characterics are descried below.

5.5.1 Destination Address Determination

The malware or attacker must determine which destination IP address to send exfiltrated data to. For malware, this address cannot be hard-coded as it would be trivial for systems to prevent the exfiltration by simply finding the destination and blacklisting it. Most modern malware uses what is known as fast-flux[13] to determine the destination. The malware may use a form of fast-flux where the malware determines a set of possible DNS names based on an internal algorithm and an external expression, such as the current date. The domain names are determined by the attacker and purchased at the appropriate time. The malware may also use a single domain lookup which the attacker changes rapidly to different IP addresses. This allows the malware to maintain the same DNS lookup that will result in IP addresses that vary over time. One of these techniques must be used and may allow a detection algorithm to detect suspicious activity. For example, rapid DNS lookups to containing seemingly random domain names may be a sign of fast-flux. Manual DNS cache flushing followed by repeated lookup of the same domain name will also signify suspicious activity.

5.5.2 Compression and Encryption

An attacker is likely to compress sensitive data collected from a machine for various reasons. In order to minimize the amount of data being exfiltrated and in turn the likeliness of being detected, only the most important of data is encapsulated. Excessive data is discarded and only key information is grouped. Compression is then used to minimize the file size even more. Compression in itself prevents data from being inspected without first being extracted. Encryption is a natural extension of compression which can prevent the data from being decompressed without the proper key. This prevents most features of detection systems simply by eliminating the ability to inspect packets. However, the tradeoff of compression and encryption is that they are processor intensive procedures. A detection algorithm can examine computations performed that may be indicative of compression and encryption and this can be used as a signal of possible malicious activity.

5.5.3 Symmetry

An attacker will go to great lengths to minimize the amount of traffic that is transmitted to and from the destination. For this reason, exfiltration will often not be acknowledged and there will be little or no response from the destination after exfiltration has completed. This is vastly different from the majority of traffic when examined on a session level. For an average user, most traffic is initiated by the user and the majority of data exchanged is downloaded from a destination to the host machine. Exfiltration would have a much different ratio of outgoing to ingoing traffic as the majority of the session traffic would be outgoing. While there is limited use in detecting active data exfiltration, the concept of session symmetry can be used and sessions with certain symmetry ratios can signify malicious activity.

5.6 Analysis

It is clear that the a study of the attack surface of exfiltration can uncover other details that may be common to exfiltration techniques. These techniques can be compiled and used with a behavioral approach in order to accurately detect and halt data exfiltration. It is important to note that the most sophisticated attacker will likely examine current traffic and choose a technique that hides well with the traffic that is native to that machine. For example, if the user commonly uses a P2P client, the attacker can detect this traffic and exfiltrated in a way that simulates that traffic. This gives the attacker the knowledge of existing traffic in order to better hide behind legitimate traffic.

Chapter 6

CONCLUSION

The impetus for this research is the lack of valid traffic samples of active exfiltration. Modern malware authors make it extremely difficult to examine malware samples in a research environment. Malware prevents inspection by detecting the existence of debugger and virtualization tools. For this reason, the most feasible way to demonstrate exfiltration traffic is to emulate it using information from existing research and mimicking the likely goals of the attacker. After implementing many methods that are likely to be used by an attacker, there are several chracteristics that are important to the attacker and certain tradeoffs that must be made. For example, the attacker must choose whether to compress and encrypt traffic to hide it from inspection from a detection algorithm or to send traffic as clear text and not alarm the user to any uninitiated processing by the machine. There are also certain features such as the types of traffic that legitimately exist on the system and the type of data being exfiltrated. UDP or ICMP may be ideal for sending data from a keylogger, but would prove unreliable for larger transfers where loss is unacceptable. P2P traffic would likely stand out among an average user's traffic who does not use any P2P clients. The timing of the exfiltration is also important for various reasons. Exfiltration occurring during hours that the user does not typically use the computer would be obvious due to the lack of background noise and the low probability that a user would be executing that action at that time of day. Additionally, it may be important to spread the transfer out over certain periods of time, so as to not saturate the network in one burst of data. The former would be more covert, but in the event that it is detected, the attacker may no longer be able to exfiltrated data. In the latter case, the user may be detected, but only after the data has been sent in full. These characteristics need to be further examined in conjunction with the development of detection algorithms. This joint investigation may allow researchers to prevent further exfiltration and gain some ground in the ever-present fight against cyber criminals.

6.1 Future Work

There is a significant amount of work that can be done to extend and make use of the findings in this thesis. The main purpose of this research was to examine the attack surface of advisaries using exfiltration as well as to provide the ability to generate samples of data that would represent exfiltration traffic. These samples can be used to test existing detection algorithms and can be used to improve these algorithms.

6.1.1 Implement Additional Techniques

The most logical extension of this work would be to implement additional methods in an attempt to examine and detail nearly every feasible exfiltration technique. This would greatly enhance the usefulness of this research and provide even more sample methods that can improve detection research. In addition to implementing more methods, there needs to be a more refined level of control for all methods included. For example, attributes such as packet size, packet timing, maximum throughput, and other features can be implemented so they can be configured at runtime.

6.1.2 Behavioral Inspection and Characterization

Since the method of exfiltration that is least likely to be detected is the method that hides well among legitimate traffic of the same type or protocol, having an application that inspects current traffic and determines the most covert technique will prove useful. This program would sniff and examine all outgoing traffic, primarily for protocol information and use this information to compute a list and frequencies of traffic types that legitimately exist on the system. The malware or attacker can then choose from this list the method that will be most ideal in terms of reliability and throughput.

6.1.3 Commercial Suite Testing

It would be useful to test the exfiltration test suite against existing commercial DLP suites. These tools often come in the form of a dedicated system which inspects all traffic exiting the private network. They inspect every packet for keywords, regular expressions, and other signatures of sensitive data leaving the network. Running the test bench program through this inspection system would allow the detection system to test its effectiveness as well as provide insight into the types of traffic that are not being properly examined.

BIBLIOGRAPHY

[1] free DNS tunneling service. `http://www.dnstunnel.de/`.

[2] libcurl - the multiprotocol file transfer library. `http://curl.haxx.se/libcurl/`.

[3] OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org/`.

[4] The Python Programming Language. `www.python.org`.

[5] The Spamhaus Project. `http://www.spamhaus.org/`.

[6] Jaafar Almasizadeh and Mohammad Abdollahi Azgomi. A New Method for Modeling and Evaluation of the Probability of Attacker Success. In *2008 International Conference on Security Technology*, page 49, 2008.

[7] S. Deering. ICMP Router Discovery Messages, June 1991. `http://www.ietf.org/rfc/rfc1256.txt`.

[8] T. Dierks and C. Allen. RFC 2246: The TLS Protocol, January 1999. `http://www.ietf.org/rfc/rfc2246.txt`.

[9] Adam Ely. HTTPS Is Evil. *Darkreading*, March 2011. `http://www.darkreading.com/authentication/167901072/security/privacy/229301300/tech-insight-https-is-evil.html`.

[10] Tyrell William Fawcett. ExFILD: a tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic. Master's thesis, University of Delaware, 2010.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, June 1999. `http://www.ietf.org/rfc/rfc2616.txt`.

[12] Kelly Jackson Higgins. Twitter Offers Users A Default SSL Setting. *Darkreading*, March 2011. `http://www.darkreading.com/insider-threat/167801100/security/application-security/229301047/twitter-offers-users-a-default-ssl-setting.html`.

[13] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and Detecting Fast-Flux Service Networks. 2008.

[14] Information Sciences Institute. RFC 791: Internet Protocol, September 1981. http://www.ietf.org/rfc/rfc791.txt.

[15] Information Sciences Institute. RFC 793: Transmission Control Protocol, September 1981. http://www.ietf.org/rfc/rfc793.txt.

[16] Simon Liu and Rick Kuhn. Data Loss Prevention. *IEEE IT Pro*, March/April, 2010.

[17] P. Mockapetris. RFC 1035: Domain Names - Implementation and Specification, November 1987. http://www.ietf.org/rfc/rfc1035.txt.

[18] Daniel Nasaw. Hackers breach defences of joint strike fighter jet programme. *The Guardian*, April 2009. http://www.guardian.co.uk/world/2009/apr/21/hackers-us-fighter-jet-strike.

[19] E. Nebel and L. Masinter. RFC 1867: Form-based File Upload in HTML, November 1995. http://www.ietf.org/rfc/rfc1867.txt.

[20] Denver Nicks. Private Manning and the Making of Wikileaks. *This Land Press*, September 2010. http://thislandpress.com/09/23/2010/private-manning-and-the-making-of-wikileaks-2/.

[21] J. Oikarinen and D. Reed. RFC 1459: Internet Relay Chat Protocol, May 1993. http://www.ietf.org/rfc/rfc1459.txt.

[22] Ed. P. Saint-Andrew. RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, October 2004. http://www.ietf.org/rfc/rfc3921.txt.

[23] J. Postel. RFC 768: User Datagram Protocol, August 1980. http://www.ietf.org/rfc/rfc768.txt.

[24] J. Postel. RFC 821: Simple Mail Transfer Protocol, August 1982. http://www.ietf.org/rfc/rfc0821.txt.

[25] J. Postel and J. Reynolds. RFC 959: File Transfer Protocol (FTP), October 1985. http://www.ietf.org/rfc/rfc959.txt.

[26] Paul E. Proctor and Eric Quellet. Magic Quadrant for Content-Aware Data Loss Prevention. *Gartner RAS Core Research*, June 2010.

[27] E. Rescorla. HTTP Over TLS, May 2000. `http://www.ietf.org/rfc/rfc2818.txt`.

[28] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on computer and communications security*, pages 635–647. ACM, November 2009.

[29] T. Ylonen and Ed. C. Lonvick. RFC 4251: The Secure Shell (SSH) Protocol Architecture, January 2006. `http://www.ietf.org/rfc/rfc4251.txt`.