9-10-2010

# Hijacking User Uploads to Online Persistent Data Repositories for Covert Data Exfiltration

Curtis P. Barnard


# HIJACKING USER UPLOADS TO ONLINE PERSISTENT DATA REPOSITORIES FOR COVERT DATA EXFILTRATION

THESIS

Curtis P. Barnard

AFIT/GCO/ENG/10-16

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

## AIR FORCE INSTITUTE OF TECHNOLOGY

**Wright-Patterson Air Force Base, Ohio**

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCO/ENG/10-16

# HIJACKING USER UPLOADS TO ONLINE PERSISTENT DATA REPOSITORIES FOR COVERT DATA EXFILTRATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Curtis P. Barnard, BS

September 2010

AFIT/GCO/ENG/10-16

**HIJACKING USER UPLOADS TO ONLINE PERSISTENT DATA REPOSITORIES FOR COVERT DATA EXFILTRATION**

Curtis P. Barnard, BS

Approved:

//signed//

_____ _____

Dr. Barry E. Mullins (Chairman) Date

//signed//

_____ _____

Lt Col Jeffrey W. Humphries, Ph.D. (Member) Date

//signed//

_____ _____

Dr. Gilbert L. Peterson (Member) Date

## Abstract

As malware has evolved over the years, it has gone from harmless programs that copy themselves into other executables to modern day botnets that perform bank fraud and identity theft. Modern malware often has a need to communicate back to the author, or other machines that are also infected. Several techniques for transmitting this data covertly have been developed over the years which vary significantly in their level of sophistication.

This research creates a new covert channel technique for stealing information from a network by piggybacking on user-generated network traffic. Specifically, steganography drop boxes and passive covert channels are merged to create a novel covert data exfiltration technique.

This technique revolves around altering user supplied data being uploaded to online repositories such as image hosting websites. It specifically targets devices that are often used to generate and upload content to the Internet, such as smartphones. The reliability of this technique is tested by creating a simulated version of Flickr as well as simulating how smartphone users interact with the service. Two different algorithms for recovering the exfiltrated data are compared. The results show a clear improvement for algorithms that are user-aware. The results continue on to compare performance for varying rates of infection of mobile devices and show that performance is proportional to the infection rate.

**Acknowledgments**

I want to thank my advisor, Dr. Mullins, for his patience and support during this process. I appreciate that you had faith in my ideas even when I didn't always have faith in them myself. Also, a big thanks to Dr. Peterson and Lt Col Humphries for challenging me to be a better student. I would also like to thank my fellow classmates who gladly listened to me ramble on about my research, and always being willing to discuss my ideas. Finally I would like to thank my family and friends that supported me during my time at AFIT. I truly could not have made it without them.

Curtis P. Barnard, AFIT

## Table of Contents

Page

## List of Figures

Figure Page

## List of Tables

Table Page

# HIJACKING USER UPLOADS TO ONLINE PERSISTENT DATA REPOSITORIES FOR COVERT DATA EXFILTRATION

## I. Introduction

As malware has become increasingly advanced, it has generated a need to communicate with the malware creator as well as other machines infected with the same code. Security researchers are responsible for understanding how these programs communicate in order to defend against them and to develop new offensive techniques for use in cyber warfare. To this end, a new technique for exfiltrating data from a target network is developed and examined to quantify its performance.

### 1.1 Research Motivation

In order to maintain active communication channels for malware, they must be covert. If the communication channel can be disrupted, then machines infected with malicious code will not be particularly useful. Many techniques have been developed to allow Command and Control (C2) traffic to bypass network filtering devices, but many of these techniques still leave some signature that can be traced. For the purpose of exfiltrating data, the ideal communication channel would generate no additional network traffic, be completely undetectable at the network level, and hide both the source and destination of data transfer.

### 1.2 Hijacking User Uploads for Covert Communication

Steganography drop boxes are not a new concept. In fact, the technique made headlines recently when it was discovered a Russian spy was using this very method for communicating sensitive information from the U.S. back to Russian agents [Sha10]. This technique traditionally involves coordinating the location of the drop box before communication takes place. Passive covert channels, on the other hand, modify data that has been generated by a user. Often this means altering unused or optional header fields in a TCP packet. This requires the recipient of this secret information to observe the traffic in transit, which is often impossible. This new technique involves combining these two concepts by modifying the data being sent to online social media websites and creating a drop box passively. This gives the benefits of allowing the data to persist in an online repository, not requiring any traffic to be generated specifically for covert communication, and being nearly undetectable by observing network traffic alone.

### 1.3 Research Goals

This thesis presents the proposed covert channel in detail, and analyzes its performance in an environment that closely resembles a real world scenario. Its reliability and practicality are examined with specific interest placed on applications on smartphones. An online data repository is created and simulated users' interaction with it to emulate a user uploading files.

To this end, the goals of this research are:

- Describe the details of this new covert channel
- Assess this technique's feasibility in conjunction with smart phones and Flickr

- Examine the reliability of different algorithms searching for exfiltrated data

### 1.4 Assumptions and Limitations

This technique relies on a series of conditions to be satisfied in order to be possible. First and foremost, anyone implementing this method of communication must already have access to execute code on a victim's device without their knowledge. This means it must be assumed an attacker has already exploited the device and has the ability to execute arbitrary code, as this technique does not address device exploitation. There are three other important limitations of this technique that will be discussed individually.

#### 1.4.1. Privacy

The first major obstacle challenging this technique is the issue of privacy. Many social media sites levy strict access controls on the content users may contribute. Since it is not always possible to know the privacy settings of the destination data repository, it is possible that some data hidden inside the media will be completely unrecoverable. In order to find this data, the searching agent must have access to view the uploaded media. This does not necessarily mean that the destination repository must be public. If the malware created to leverage this technique spreads through the social media site, it is possible that someone who was granted access to private media for a particular user was also infected, allowing it to be recovered by using their session to search for non-public images.

### 1.4.2. Data Integrity

Many social media websites alter the media themselves before making them accessible. For example, many sites may add a watermark to an image or resize it to particular dimensions. The steganography algorithm used to hide data must be able to function despite these modifications. While algorithms can be developed to be resilient against these kinds of changes, it is possible that it would also be much easier to detect that data has been hidden inside the media. The developer of the algorithm must balance the algorithm's resiliency with the risk of detection.

### 1.4.3. Searchability

The final main obstacle is the ability to search for the media once it has been uploaded. The repositories must be searched at a rather quick pace to keep up with the rate of uploaded media. It is possible that the host of the social media service could detect this searching activity and realize it is being conducted by a computer program rather than a human. Their reaction could include anything from blocking the offending IP addresses to contacting the authorities to investigate which would cause decreased performance as best, and potential legal issues at worst. Some of these social media service providers, however, encourage third party developers to create applications which search through the media as part of their normal function. If this is the case, this complication can be avoided by searching at a rate that would be common with one of these applications.

### 1.5 Thesis Overview

This chapter introduced a new covert channel for exfiltrating data from a compromised device, specifically smartphones, through embedding that data in user uploads to social media sites. Chapter 2 discusses the background to understand the technique itself, why it is important, and how it differs from current techniques for covert communication. Chapter 3 discusses the methodology used to test this technique's performance. It also covers the experiments that are generated to determine the feasibility of using this technique on smartphones through Flickr services. Chapter 4 covers the results of those experiments and what those results mean to someone implementing this technique. Chaper 5 lists the conclusions of this research and makes suggestions as to how this research could be continued in the future.

# II. Background

This chapter provides context for the covert channel developed in this research. Section 2.1 describes how malware has evolved over the year and explains why covert communication is important to malware authors. Section 2.2 discusses steganography, how it has developed over several hundred years, and how it is used as a modern way to hide the existence of a communication, an important aspect of our covert channel. Section 2.3 describes some modern covert channels and explains concepts used in the development of this new method of data exfiltration. Finally, Section 2.4 concludes this chapter.

## 2.1. Evolution of Malware

Since the explosion of personal computing, clever programmers have been finding ways of writing code that causes these machines to behave in unexpected, and often undesirable ways. Malware is one byproduct of this creative, yet destructive, practice. Modern malware is highly sophisticated, but that was not always the case. In order to better understand how modern day malicious code functions, it is important to understand how malware evolved over time.

### 2.1.1. Virus

The term "computer virus" is one of the earliest categories of malware, coined as early as 1983 by Fred Cohen [Spa94]. His early definition simply described viruses to be any program capable of self replication. This liberal interpretation could technically be applied to normal programs such as compilers and text editors [Spa94]. Other researchers

have applied a more conservative definition to the term, stating that viruses are programs which replicate by attaching copies of themselves to other executable binaries [Kur89][Nac97]. While viruses may include code designed to damage the host machine, known as payload [Nac97] or logic bomb [Kur89], a virus does not necessarily have to contain a payload to be accurately deserving of the classification.

Viruses must have some way of embedding themselves into existing executable code. There are several techniques for accomplishing this, and while not an exhaustive list, there are three common subclasses of viruses, grouped by their replication method.

Shell viruses essentially wrap themselves around the original executable. The existing code is moved to another location, and called later as if it were a function of the virus [Spa94].

Add-on viruses replicate by adding their own code to either the beginning or the end of existing executables. In the case of being added to the beginning, the virus code executes first, and then hands control back to the original unedited host code [Spa94][Nac97].

Intrusive viruses actually overwrite code in the host program and replace it with their own. It is quite possible that this technique could render the host executable unusable. Because of the propensity to cause damage to the host file, this virus type is often considered the least common [Spa94].

Advances in anti-virus technology forced further virus evolution [Nac97]. Since viruses had historically created exact replicas of themselves on disk, anti-virus programs simply had to search for the virus' binary string on disk to locate it. In an attempt to avoid this very simple detection technique, viruses began using encryption to alter their

signature, or mutate, upon every infection [Nac97][Spa94]. To accomplish this, a small code segment known as a decryption engine had to be present in an unencrypted form. This small algorithm would then decrypt the original virus, and the flow of execution would be handed off to the virus which would execute normally [Nac97].

While pure viruses are not the norm today, there are a few early examples of very simple viruses. The following sections describe two examples.

#### 2.1.1.1. BRAIN

The BRAIN virus is a simple and relatively benign virus that would spread through the use of floppy disks. When a floppy disk was inserted into the drive of the computer, it would change the volume label assigned to the disk to "BRAIN". If the floppy was used as a boot disk, it would destroy some data to insert code into the boot sector which would infect the machine causing that computer to infect any additional floppies inserted into it later with the BRAIN virus [Kur89].

#### 2.1.1.2. Israeli Virus

The Israeli Virus was discovered with a logic bomb that destroys files a user tries to execute, but was set to trigger on Friday, May 13, 1988. It would also trigger every Friday the 13th thereafter. It would add a copy of its code to any COM or EXE file. Due to a bug, it would reinfect the same files multiple times, leading to large file sizes. This bug eventually led to the virus' discovery.

### 2.1.2. Worm

As time went on and computers became increasingly interconnected through networks, more sophisticated malware entered the scene. The term "worm" is used to

describe a program that automatically propagates over the network by way of security vulnerabilities in popular services [WPSC03]. The distinction between a virus and a worm is subtle and often times blurry. Viruses propagate by using files that users execute on their systems, while worms propagate by leveraging the network itself. Additionally, user interaction is often not central to the spread of a worm, while it is almost always necessary in the case of a virus [WPSC03]. One of the first worms to get public visibility was known as the Morris worm, and it struck in early November 1988 [ER89] [EGH+89], five years after the first computer virus.

Since these worms must find and infect hosts automatically over a network, a number of new problems must be resolved by worm developers. One big hurdle to overcome is the fact that worms must discover and select targets that are vulnerable to infection [WPSC03]. There are many techniques used by worms for discovering new targets, but special attention is paid to two methods in particular: scanning discovery and passive discovery [WPSC03].

In the case of scanning discovery, an infected machine will search through a set of Internet addresses until it finds a vulnerable host [WPSC03]. Two common and simple scanning algorithms are random and sequential scanning [WPSC03]. Sequential scanning will probe different addresses incrementally until every address in the worm's search space has been exhausted. Random scanning, on the other hand, selects addresses to scan at random. While scanning may be a crude form of target acquisition, it has proven effective in the case of the "Code Red" worm, in which over a quarter of a million computers may have been compromised with random scanning [MSC02]. At least one

study suggests that the worm may have compromised nearly every single vulnerable machine which was publicly available [WPSC03].

Another interesting technique for target discovery is passive discovery. Worms using this method never generate traffic for finding vulnerable hosts. They rely on the user or some program to interact with other machines, and by fingerprinting this interaction, they then determine if that host is a suitable target [WPSC03]. One example of a worm which leveraged this technique is CRClean [WPSC03]. This worm would passively listen for a probe from the Code Red worm, and upon its receipt, would use the same exploit Code Red used to infect the machine which issued the probe. After infection, it would delete Code Red and install itself. It was in essence an anti-virus worm which spread without any active scanning [WPSC03].

Worms tend to be relatively high profile due to the number of users they affect. The following sections describe some examples of popular worms that were released upon the public.

#### 2.1.4.1. Morris

The Morris worm was one of the first known pieces of code that conforms to what is formally considered to be a worm. It was released on the Internet by Robert Morris Jr. in November of 1988 and infected roughly 6,000 computers. While the program was not designed to destroy files or data, its 99 lines of code did cause the infected machines to slow to a halt [EGH+89] [MSC02].

#### 2.1.4.2. Code Red

Over 359,000 computers were compromised by the Code Red worm after it was released on July 19, 2001 [MSC02]. The code used a random scanning technique for discovering new targets to infect, but suffered from using a static seed for the random number generator [MSC02]. On a specific day, all machines infected by Code Red would participate in a Denial of Service attack against www1.whitehouse.gov [MSC02]. Several variants of this worm were released before the security vulnerability was brought under control, including a variant with a repaired random number generator. The fast probing speed of this worm along with its wide spread infection overloaded some telecommunications infrastructure, and some estimates calculate total financial losses to be as much as $2.6 billion [MSC02].

#### 2.1.4.3. Slammer

The Slammer worm was released on January 23, 2003 and was notable for its alarmingly fast propagation speed. Some research suggests that as many as 90 percent of machines vulnerable to infection were compromised after a mere 10 minutes [MPS+03]. It is suggested that its growth was slowed by bandwidth limitations. While Slammer infected far fewer machines then Code Red, it spread at roughly two orders of magnitude faster [MPS+03]. The main reason for this difference is the fact that Slammer scanned with UDP, and Code Red scanned with TCP. Since Code Red used TCP, it had to wait for a SYN/ACK packet to determine if a host was vulnerable, while Slammer could send out as many UDP packets as its network's bandwidth allowed [MPS+03].

### 2.1.3. Rootkit

One of the main reasons for an attacker performing an intrusion is for intelligence gathering [HB05]. Towards this end, malware writers came up with techniques for hiding their activities on compromised machines. One way this was achieved was through the use of rootkits. One author defines rootkits as "a set of programs and code that allows permanent or consistent, undetectable presence on a computer" [HB05].

The two main concepts in this definition of a rootkit are those of permanent and undetectable presence. While these goals are often in direct conflict with one another, they are both highly important for the purpose of intelligence gathering. In general, rootkits satisfy these two goals by modifying a computer's decision making process. A rootkit author will cause the computer to react to certain data, forcing a poor decision [HB05].

One method used for hiding the presence of code on a computer is through a technique known as "hooking" [HB05]. In order to force the computer to make a poor decision, you must either affect the execution flow of the code that generates the decision, or you must influence the data the computer is examining [HB05]. Hooking works by altering the flow of execution. Certain events that occur on a computer can cause system level functions to execute. For example, bringing up the task manager on a computer running Windows XP causes taskmgr.exe to request the system to execute a function called ZwQueryProcessInformation() as shown in the first step of Figure 1. That function queries a list of kernel objects representing every executable running on the system, and returns that list to the requesting program, in this case taskmgr.exe as shown in step 2 of Figure 1. After that, the task manager displays that information back to the

screen. Figure 2 demonstrates a popular technique for preventing task manager from displaying a malicious program in the task list by hooking the ZwQueryProcessInformation() function. A malicious user hooking this function can trick taskmgr.exe into calling a different specially-crafted function which in turn calls the real ZwQueryProcessInformation() as seen in step 1 of Figure 2, but filters its output to remove any references to his malicious executable as shown in steps 2 and 3 [HB05]. By modifying the execution path of the call to ZwQueryProcessInformation(), a rootkit author can be sure that any program requesting a list of running processes will not be made aware of programs he wishes to hide.

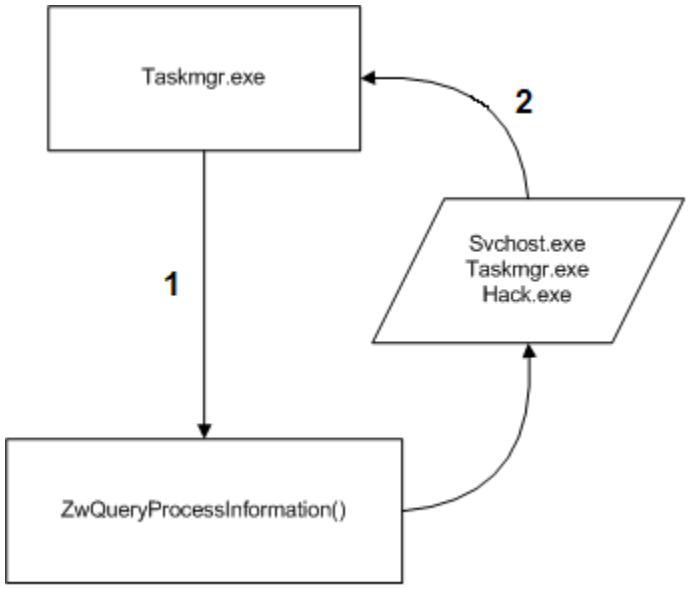


**Figure 1:** Standard ZwQueryProcessInformation() call

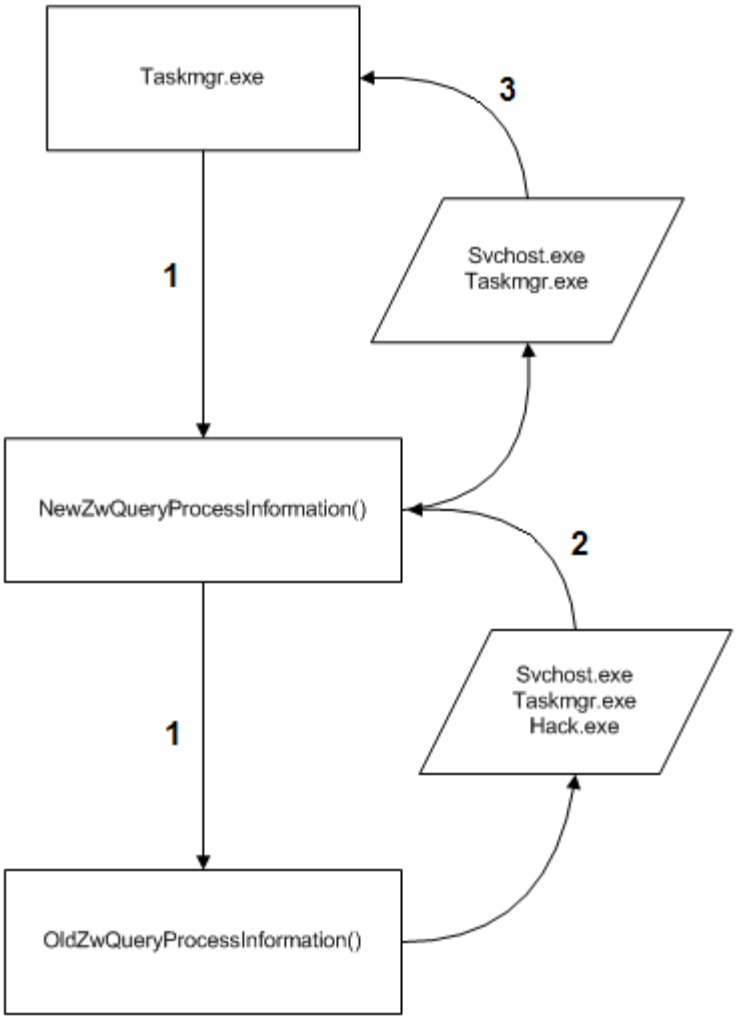


**Figure 2:** Hooked call to ZwQueryProcessInformation()

The other activity rootkits perform is remote access [HB05]. An attacker would like the ability to maintain control over a computer and allow himself the opportunity to run commands and access data on the machine. Specifically, the process of downloading data from a target system is of particular interest, known as "exfiltration" [HB05]. Any time an attacker intends to exfiltrate data from a machine, it requires the computer to access a network [HB05]. In modern computing, this means communicating through a TCP or UDP socket, which is a process that triggers a number of observable events, both on the machine itself, as well as on the network. Many firewalls and intrusion detection systems

monitor and log newly established TCP connections, which makes the process of covert exfiltration difficult [HB05]. One strategy that is used to conceal network traffic dedicated to data exfiltration is to mask it to look like traffic that is common on that network [HB05]. This makes it difficult for anyone monitoring the network traffic to distinguish it from normal traffic. For example, it is possible to pass data through a network with a technique known as DNS tunneling. DNS traffic is common on almost every network, and is nearly universally white listed to pass through firewalls. An attacker can write code that will wrap data inside a packet that looks like a DNS request, bypassing most monitoring methods.

Rootkits exhibit a very high level of technical sophistication, and often depend highly on the operating system of the target. One rootkit will not necessarily work on all Windows machines the way a higher-level virus might. Despite the narrow window of usefulness for some rootkits, the techniques they employ can be used to develop alternate covert channels.

### 2.1.4. Botnet

As malware became increasingly sophisticated and attackers grew more and more concerned about maintaining access to previously compromised machines, a new type of malware appeared: the botnet. Botnets are groups of compromised computers that are all remotely controlled under a common Command and Control (C2) protocol and are considered to be one of the most serious threats to Internet security to date [LTG09]. Due to the ability to remotely control thousands, if not millions, of infected hosts at once, botnets have certain capabilities that an attacker could not otherwise hope to harness.

Most notably, botnets can carry out a Distributed Denial of Service (DDoS) attack, which is capable of bringing even some of the highest load tolerant web services offline [LTG09]. Additionally, botnets are responsible for a large portion of the spam that traverses the Internet [LTG09]. These kinds of activities are possible due to the large number of compromised machines, and they often rely on certain probabilities (which are frequently low) that a particular security vulnerability exists. For example, many botnets spread through well known security flaws that are patched on most systems. These flaws are not patched on all systems however, and even if only one percent of computers are vulnerable to attack, the botnet can grow to a substantial size by scanning a larger number of possible hosts.

Another popular use of a botnet is to gather huge amounts of data from a large number of users [LTG09]. Once a botnet has taken hold in a user's computer, it can capture keystrokes, files, screenshots, chat logs, website credentials, or any other piece of data stored on a computing device. When a large number of machines are infected, the probability that one of them is being used for online banking or contains a social security number greatly increases. It is this statistical phenomenon that makes botnets so unique and effective. Successfully preventing these kinds of attacks requires a defender to protect every machine at all times, while an attacker must simply locate a low probability vulnerability.

Over recent years there have been a number of botnets that have grown to a substantial size, and obtained media attention. Two such examples are described in detail.

#### 2.1.4.1. Conficker

In early 2009, a new botnet known as Conficker began growing to an extraordinary size, inciting Microsoft to offer their fifth ever reward for indentifying a malware's creator [Por09]. Despite using mostly tried and true techniques rather than inventing novel ones, this random scanning botnet was highly successful in infecting a large number of users. It used a method known as Fast Flux for selecting command and control servers, and featured an encrypted Peer-to-Peer (P2P) protocol for updating and distributing new vulnerabilities [Por09]. Unfortunately, the motives of the Conficker developers remains mostly a mystery, although it is currently being used mostly to conduct well understood forms of Internet fraud [Por09].

#### 2.1.4.2. Torpig

Torpig is a common piece of malware whose primary goal is to obtain sensitive information from a user and is commonly distributed through what is known as a drive-by-download attack [SGCC+09]. It is usually downloaded as part of a rootkit known as Mebroot, which infects a computer's Master Boot Record, causing it to execute before the operating system has even loaded [SGCC+09]. One group of researchers had the unique opportunity to assume control of this botnet to observe it's information gathering activities [SGCC+09]. They noticed that Torpig would contact a central C2 server every 20 minutes to upload additional stolen data, and used some very basic protocol obfuscation to prevent easy detection [SGCC+09]. Torpig gathers most of this stolen data through what is known as a man-in-the-browser phishing attack, where a user enters a website into their browser (such as a bank) but the botnet malware actually causes a

different page to appear which closely resembles the one the user intended to visit. The user enters their credentials into the malicious webpage, then that malicious page forwards them along to their normal banking website [SGCC+09]. Over a period of ten days, these researchers observed 180 thousand infections and observed over 70 GB of stolen data collected [SGCC+09].

### 2.2. Steganography

A technique central to this research is steganography. Steganography is the art and science of hiding the existence of information in a message [TZH+08]. Unlike cryptography which aims to protect information from being intercepted by an unintended third-party, steganography aims to conceal that information had ever been transmitted at all. The implementation of this concept can range from simply knowing a hiding place for information to using complicated mathematics to conceal data inside digital media files. In this section, we examine the gamut of steganography starting with early uses and ending with modern techniques still being developed by researchers.

#### 2.2.1. Steganography History

One of the first well known usages of steganography was carried out by Histiaeus in 494 B.C. during a major struggle between Greece and the Persian Empire[LMS10]. Histiaeus shaved the head of one of his slaves, and then tattooed an important message onto the slave's scalp. He then waited for the hair to grow back, covering the message. The slave then traveled to the intended recipient and shaved his head, revealing the message which led to a revolution against the Persian Empire [LMS10]. Another technique that was used in that era involved carving a message into a wooden tablet. The

author would then cover the plank in several layers of wax before transporting it. When the plank was received by the intended recipient, they would heat it, melting the wax away and revealing the message [LMS10]. Something as simple as invisible ink can also be used to send secret messages. During World War I, a German spy performed shows in Britain as a cover for espionage and sent information back by writing messages in invisible ink on sheet music [LMS10]. The level of sophistication increased during World War II, when the Germans would shrink pages of information down to a one millimeter point and disguise it as a period in an otherwise innocuous letter [LMS10].

### 2.2.2. Modern Methods

Modern techniques for employing steganography often require the use of a computer, and generally revolve around editing audio, video, or image files and data streams in such a way as to embed additional data within them, while keeping the content unchanged from the perspective of the user observing it [FGD01]. We will examine several modern techniques for hiding data in digital images which will assist us in determining what methods would be most effective for data exfiltration.

#### 2.2.2.1. Digital Images

Digital images have been one of the more popular data types for embedding steganography, and several techniques have been developed for using them as cover data. Probably the simplest algorithm for hiding data in images is known as Least Significant Bit (LSB) manipulation. This technique requires an individual to alter each pixel in an image just slightly, being undetectable to the human eye [FGD01]. The bitmap image format stores every pixel value in a three item set of 8-bit unsigned integers. The first

integer is the amount of red for the pixel, the next is the amount of green, and finally blue. These integers can range in value from 0 to 255. While a human can easily observe the difference between a pixel that contains a red value of 0 and one that contains a red value of 255, the difference becomes much less obvious at closer values. In fact, when the values differ by only one or two, the difference is almost undetectable to the human eye. It is this property of images that LSB manipulation relies on. When a user creates a piece of data they wish to hide in an image, they add it to the cover image one bit at a time. The first pixel is examined in the images, and the lowest bit of one of the 8-bit color values is replaced by the first bit of the data to be encoded. The process is repeated for every pixel and every bit of data until all the bits have been embedded, then the remainder of the image remains unaltered. Some variants of this technique have been used where multiple or all of the color values in a pixel are altered, or where data is placed in the 2 least significant bits instead of just the one. In fact, anyone could modify this technique in a variety of ways, and it would still be a successful method for hiding data in images. While this technique is effective, it is also relatively easy to detect. Most data is encrypted before being embedded within an image, causing the data to be random. Frequency analysis of the least significant bits of the pixels in an image can reveal if there appears to be encrypted data hiding there.

### 2.3. Covert Channels

Covert channels, while similar in many respects, differ from steganography in one important way. Steganography hides information in a message, while covert channels transmit messages in such a way that an observer cannot detect the transmission. With the

massive amount of bandwidth and traffic that already exists on the Internet, it becomes an ideal carrier for secret communications [ZAB07]. Covert channels can be broken down into two main classifications - storage channels, and timing channels [ZAB07]. Storage channels require the sender to write data to an object and have the receiver recover that object later. Timing channels involve modulating the use of a resource in a way that is observable to the receiver. Techniques that fall into each of these categories will be discussed later in this section.

### 2.3.1. The Prisoner Problem

The prisoner problem was described by Gustavus Simmons as a model for developing covert channels. In this problem, two prisoners, Alice and Bob, are planning an escape, but must communicate with one another to coordinate it. They are kept in separate cells, and can only pass messages to one another after being scrutinized by the warden. If the warden determines the messages are anything but innocuous, he will discard it [ZAB07]. In this model, wardens come in three varieties. Passive wardens are only able to observe the messages passing between Alice and Bob. Active wardens are able to modify the content of a message as long as it does not alter the semantic context. Finally, passive wardens are able to alter messages as they see fit [ZAB07]. When extending this model to the Internet, Alice and Bob are represented by end users who want to communicate covertly, while the warden is represented by the network administrators monitoring and securing the network.

### 2.3.2. Timing Channels

There are several methods a covert channel can be created based on timing modulation. Many techniques exploit tolerance levels inherent in the TCP, UDP, and IP protocols. Removing the thresholds that make these techniques possible would likely require a complete overhaul of the protocol [ZAB07] so it is unlikely that these techniques will be thwarted in the near future. Several of these methods are outlined below.

#### 2.3.2.1. Packet and Message Sequence Timing

One method for establishing covert communication using a timing channel is by varying the rate that hosts send packets [ZAB07]. The sender of covert information can select different rates for sending packets to the receiver, and the receiver can interpret those rates to infer the binary data the sender was intending to transmit. One specific implementation is to specify a time interval and transmit packets in a timeslot when you intend to send a covert "zero", and stop transmitting packets when you intend to send a covert "one" [ZAB07]. Another method is to modulate the time between sending consecutive packets in a transmission. A short interarrival time between two packets may represent a covert "one", and a larger interarrival time may represent a "zero" [ZAB07]. Both of these strategies for sending covert messages must account for inherent latency in the network, which may seriously limit the rate of covert data transmission.

#### 2.3.2.2. Collision, Loss, and Retransmission

Covert communication can be generated through exploiting Medium Access Control (MAC) protocols. Over the Ethernet protocol, the covert sender may intentionally cause a

collision with another host. He then waits either the minimum or the maximum wait time during the backoff period as a method for encoding his data. The covert receiver then examines the order of the retransmissions between the sender and the host that was jammed and can infer the bit that the sender encoded [ZAB07]. This is possible because even though the Ethernet protocol specifies how collisions should be handled, a malicious network driver can violate that procedure without it being obvious that the protocol is being ignored.

Any protocol using sequence numbers can also be exploited through forcing errors. A covert sender can force artificial packet loss by modifying sequence numbers. Periodically, a sequence number may be skipped causing a sequence error. The packet is correctly retransmitted later, but through either inferring data from the sequence number that was skipped, or by modulating the frequency of sequence errors, the receiver can extract the data hidden by the sender [ZAB07].

Finally, malicious hosts can also use forced packet retransmissions in a wireless environment by injecting frames with intentionally corrupt checksums, or in a wired environment by retransmitting frames. Similarly to the method for hiding data in sequence errors, modulating the rate of these errors is a method that can be used for hiding data [ZAB07]. The users of these covert channels and those attempting to disrupt or observe them must, however, account for naturally occurring errors, and not assume that all checksum errors are attempts at covertly exfiltrating data.

### 2.3.3. Storage Channels

Storage channels require modifying data objects in ways that do not disrupt normal network communication. Any data section where the value is irrelevant or modifying values that are intended to be random in non-random ways are both methods for covert communication through a storage channel. Many techniques have already been developed and are described in the following sections.

#### 2.3.3.1. Unused Header Bits

Several header fields in packets of many protocol types can be set to arbitrary values under certain conditions which pave the way for this covert storage channel. One example is the Don't Fragment (DF) bit in an IP header. If the sender is aware of the maximum transfer unit for an IP connection, the DF bit can have an arbitrary value. An attacker can set this bit to represent the data he is trying to covertly communicate [ZAB07]. Padding bits are another example of space that can be assigned arbitrary values. Ethernet frames must be padded to 60 bytes and not all protocols specify what that padding must be [ZAB07]. An attacker might choose to embed his data into this padding as a method for covert communication [ZAB07].

#### 2.3.3.2. Optional Header Fields

Similar to the previous method of putting data in unused headers, someone constructing a covert channel can use optional headers for hiding data. Many protocols have allowances for headers which they may have not yet developed, giving attackers an opportunity to create their own, and place their covert data there [ZAB07]. Additionally, some standard headers which typically provide additional optional information can carry

information by modulating if you choose to use them in a particular packet or not [ZAB07].

#### 2.3.3.3. Semantic Overloading of Header Fields

Data can also be transferred covertly by modifying some fields in a packet in a way that encodes data but is syntactically identical to the unmodified version of that packet. For example, initial sequence numbers (ISNs) are selected in a TCP connection to be unique from other recent TCP connections [ZAB07]. Usually, the ISN is set through a pseudo-random number generator, but covert data can be sent by replacing the most significant byte of the ISN to the value of the first byte of data you want to transmit covertly, and replace the lower three bytes with zeros [ZAB07]. The resulting packet is semantically identical to the original unaltered version. Another example is to alternate the case of letters in an HTTP request. When an HTTP request is received, case is ignored, so altering the case of those characters has no impact on the query itself, but the modulation can be detected and interpreted as covert data [ZAB07].

### 2.3.4. Indirect Channels

An indirect channel is where the user sending a covert message uses an innocent third party as either a relay or a host for covert information through the use of indirect storage and indirect timing channels. One example of an indirect storage channel is when a malicious user sends a packet to an innocent third party embedding data in the ISN, and spoofing the source IP address to that of the intended recipient. The third party responds with an ACK to the intended recipient with ISN+1 as its sequence number. The receiver then knows that ISN-1 is a block of covert data.

## 2.4. Conclusions

This chapter outlined why malware communicates, and why it is important for malware authors to be able to communicate covertly over the network. It also examined two critical concepts behind the novel concept of user upload hijacking for data exfiltration – steganography and covert channels. It showed that steganography is a distinct discipline from classic covert channels, and laid the foundation for understanding how the two concepts are combined in the technique created through this research.

# III. Approach

Developing and understanding covert channels is an important exercise for a number of reasons. It is critical to understand how a covert channel might be developed in order to better defend against them. Additionally, in order to maintain information superiority in the cyber domain, the ability to develop covert communication channels would certainly be to one's advantage.

This chapter outlines a new technique for covertly exfiltrating data through the Internet from an infected smartphone. It continues on to describe a method for evaluating the performance and reliability of this new technique.

## 3.1. Problem Definition

After an attacker compromises a computing device and can run arbitrary code, he often wants to conceal the intrusion. In the case of more sophisticated malware such as botnets, attackers also want to conceal any network traffic it might generate for communication purposes. For this reason, many covert channels have been developed which can avoid detection from security devices with network-level introspection. Unfortunately, most if not all of these techniques leave artifacts that are detectable at the network level.

Active covert channels, such as HTTP wrapping, generate packets dedicated to covert communication but disguise them as another type of traffic. While from a network monitoring perspective, the traffic itself may seem innocuous, the existence of information transfer between two hosts can still be inferred through traffic analysis and monitoring source and destination addresses. Passive channels, on the other hand,

sacrifice reliability for additional stealth. These channels, such as TCP header modification, ride on top of traffic that is being generated by the computer naturally. Due to this design characteristic, no new packets are generated in order for communication to occur. One complication, however, is that the receiver of this information must be able to observe the traffic that is naturally being generated by the sender. Often this limitation is bypassed by having the sending host initiate communication with the receiver, or positioning a sensor node in a position where it can monitor outbound communication. Additionally, passive channels usually generate anomalies that are detectable with network monitoring equipment. Non-random sequence numbering and frequent packet retransmission are just two examples of observable artifacts generated by some of the more popular covert channel techniques. This research aims to generate a novel one-way covert channel which causes no observable disturbance to regular network traffic and examine its performance.

### 3.2. The Technique Explained

The technique developed through this research is designed with a few specific features in mind. Specifically, it is designed as to not generate any network packets and to only modify existing packets in a way that could be attributed a legitimate user. This idea involves employing a storage channel using user-supplied data as the storage vehicle as opposed to TCP or IP headers as used in other techniques. This new covert channel is called "Hooking User Uploads to Public Online Persistent Data Repositories".

With the explosion of the Internet, social networking, and social media, users are encouraged to record and share much of what they experience with online communities.

Many of these communities encourage their users to share this information publicly. This trend of publicly sharing media online offers a unique opportunity for those developing covert channels. User uploads can be used as a container for exfiltrating data from a device through the use of steganography. Specifically, smartphones are targeted since these devices are used to create and upload data to content hosting websites as well as used to make purchases, such as applications, meaning the device will have access to personal information such as credit card numbers. After this data is uploaded, it is retrieved using a botnet to search the data repository for media containing hidden data. A simple example of what this topology might look like is shown in Figure 3, with smartphones and mobile devices uploading data to an online media repository, and a network of machines (presumably a botnet) searching that repository for hidden data.

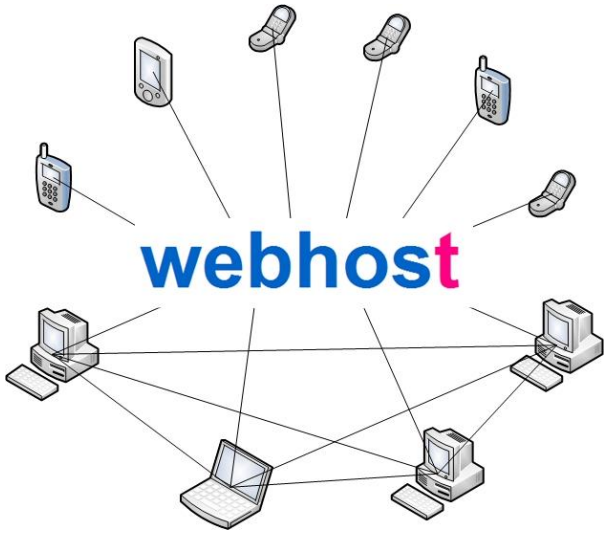


**Figure 3:** Sample topology for user upload hijacking

The heart of the covert channel lies in altering the data that users upload to social media sites from their phones. This process is illustrated in Figures 4, 5, and 6. First, it is presumed that some malware has infected a mobile device (1), and a data collection

mechanism has been put in place, such as a keylogger, that adds data to steal to a buffer stored on the phone (2).

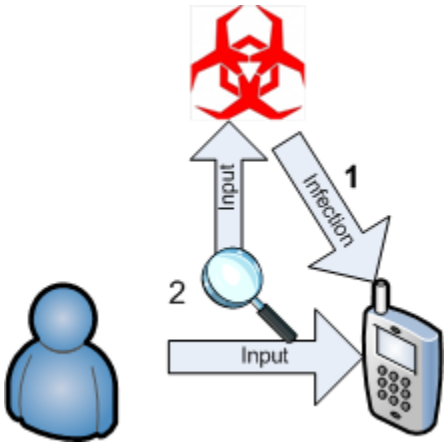


**Figure 4:** Infection and data collection phase

During the course of normal use, the owner of the phone will likely take a photo, video, or audio clip and upload it to some website (3). This action occurs when a user selects a piece of media and triggers an upload function on their device. This upload function is either located in a phone application or as part of the phone's operating system. As shown in Figure 5, this function can be hooked in a similar fashion that rootkit developers hook kernel functions. Once execution flow is diverted to the malicious function, it is used to pull data from the buffer of information to steal, and hide it inside the user's image, video, or audio file (4). The flow of execution is then returned back to its normal path, which will send the altered image to the site selected by the user making this modification completely indistinguishable from a regular user action at the network level (5).

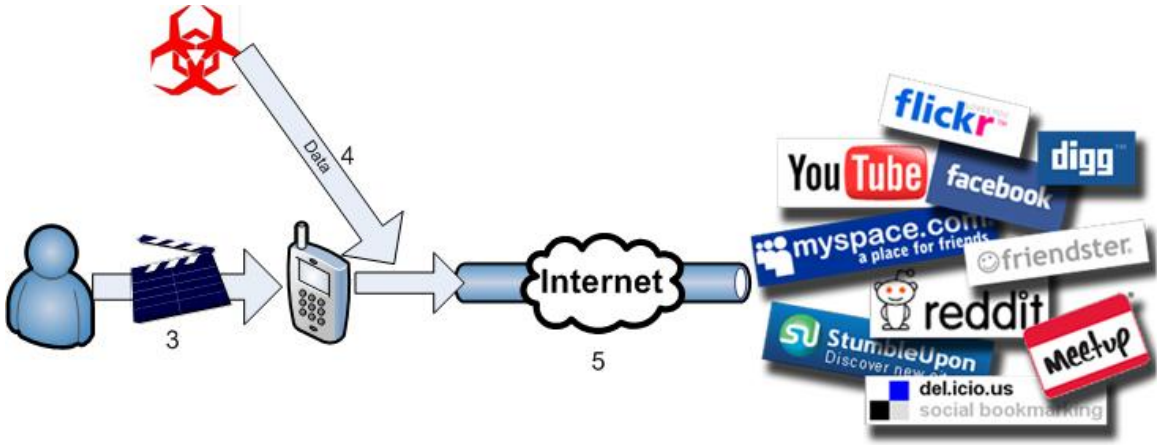


**Figure 5:** Media interception and data embedding phase

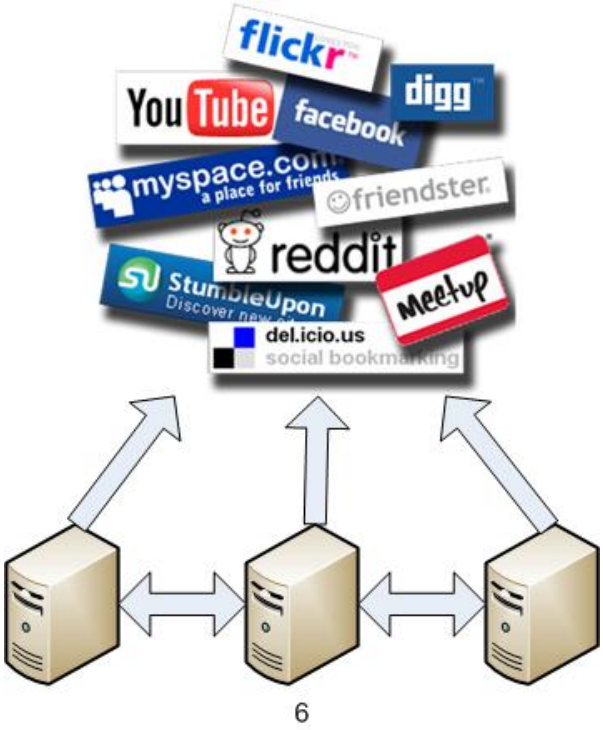


**Figure 6:** Data searching phase

The resulting uploaded media appears unaltered, but contains the information which was collected and exfiltrated. If the user account associated with the infected user is known, locating their uploaded media is a trivial operation. If the username is unknown, locating the file becomes significantly more difficult, possibly requiring a brute force

search effort. Because of this, distributing this searching process over multiple machines, or possibly an entire botnet, is proposed (6).

This technique was designed with a botnet application mind. Sifting through every piece of media uploaded to a popular media hosting service is very computationally expensive. When the task is distributed over a large number of hosts, the probability of successfully recovering data should grow significantly. For this reason, a botnet would particularly well suited entity for performing the search. This research, however, focuses on utilizing only one searching agent and using the rate of search to infer performance over a larger number of hosts.

### 3.3. Assumptions and Limitations

This technique relies on a series of conditions to be satisfied in order to be possible. First and foremost, anyone implementing this method of communication must already have access to execute code on a victim's device without their knowledge. This means it must be assumed an attacker has already exploited the device and has the ability to execute arbitrary code, as this technique does not address device exploitation. There are three other important limitations of this technique that will be discussed individually.

#### 3.3.1. Privacy

The first major obstacle challenging this technique is the issue of privacy. Many social media sites levy strict access controls on the content users may contribute. Since it is not always possible to know the privacy settings of the destination data repository, it is possible that some data hidden inside the media will be completely unrecoverable. In order to find this data, the searching agent must have access to view the uploaded media.

This does not necessarily mean that the destination repository must be public. If the malware created to leverage this technique spreads through the social media site, it is possible that someone who was granted access to private media for a particular user was also infected, allowing it to be recovered by using their session to search for non-public images.

### 3.3.2. Data Integrity

Many social media websites alter the media themselves before making them accessible. For example, many sites may add a watermark to an image or resize it to particular dimensions. The steganography algorithm used to hide data must be able to function despite these modifications. While algorithms can be developed to be resilient against these kinds of changes, it is possible that it would also be much easier to detect that data has been hidden inside the media. The developer of the algorithm must balance the algorithm's resiliency with the risk of detection.

### 3.3.3. Searchability

The final main obstacle is the ability to search for the media once it has been uploaded. The repositories must be searched at a rather quick pace to keep up with the rate of uploaded media. It is possible that the host of the social media service could detect this searching activity and realize it is being conducted by a computer program rather than a human. Their reaction could include anything from blocking the offending IP addresses to contacting the authorities to investigate which would cause decreased performance at best, and potential legal issues at worst. Some of these social media service providers, however, encourage third party developers to create applications which

search through the media as part of their normal function. If this is the case, this complication can be avoided by searching at a rate that would be common with one of these applications.

### 3.4. Experiment Design

In order to examine the performance of this technique, an experiment that mimics the process of uploading photos from a device such as a smart phone to a social media website is developed. For the purposes of these experiment, the experience a user would have uploading images to Flickr from an Android powered smart phone is simulated. Using a custom web application, a small number of features that are provided by Flickr are emulated. Specifically, image uploading, image browsing, and searching for images uploaded by a particular user are the capabilities that were reproduced. A network traffic generation tool is then leveraged to simulate users uploading images to the emulated Flickr site, a certain number of uploads contain steganography, simulating image uploads from regular users as well as users that have been infected with malware that functions as described. Additionally, agents downloading and scanning images from the site according to varying algorithms are also simulated, recording when they locate an image with data stored in it. Metrics for evaluating the reliability and response time of this technique are also generated. Since the effectiveness of user upload hijacking is being tested, the system under test (SUT) is a simulated Flickr site as well as the associated user actions required to execute the technique. A block diagram for this SUT is shown in Figure 7.

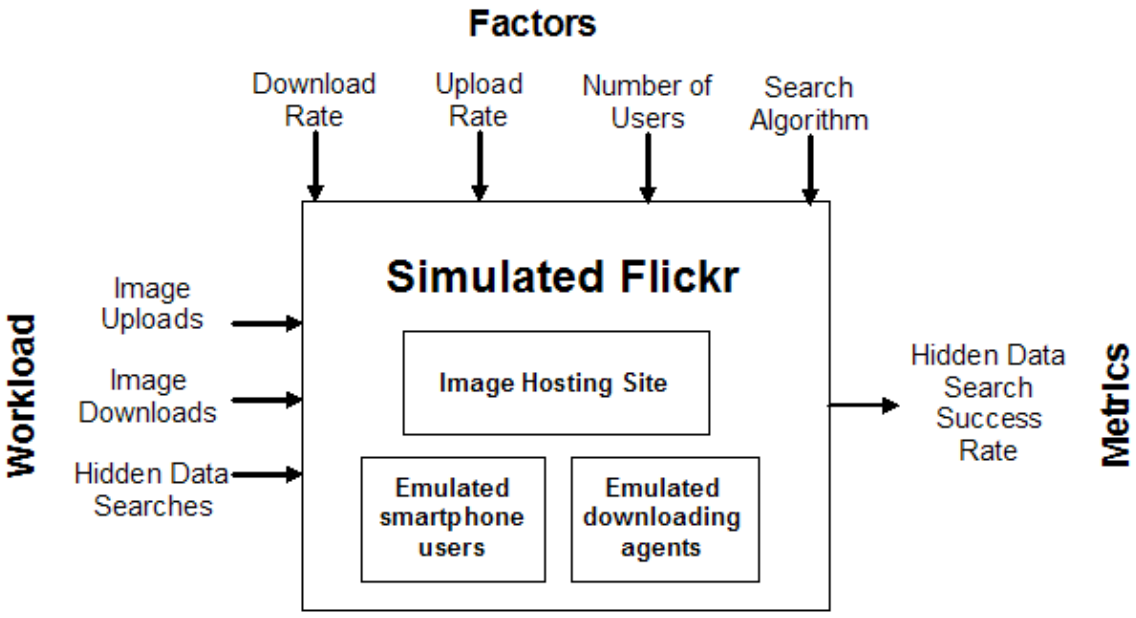


**Figure 7:** The system under test

In an effort to simulate the kinds of network activity needed to carry out this technique, a few discrete types of actions users perform when interacting with social media sites through their mobile devices were relied upon. Specifically, users that upload media, and the searching hosts acting as downloading agents are of particular interest. To satisfy the conditions required, programs are written to simulate these actions, and scheduled using a network traffic generator developed by Pacific Northwest National Laboratory (PNNL) called the Security Assessment Simulation Toolkit (SAST).

The searching algorithm is both a factor as well as the component under test (CUT). Section 3.4.1 describes the hosting site simulating Flickr while Section 3.4.2 goes into detail about the simulated smartphone users, and Section 3.4.3 discusses the downloading agents.

### 3.4.1. Image Hosting Site

Our technique hinges upon the existence of a public repository for hosting uploaded media, therefore one was created to be monitored in a controlled environment. Leveraging Ruby on Rails, an open-source web application framework, an image hosting web application was created with a few key features. Most importantly, users are able to upload image files through a simple web interface. They are also able to specify a username to associate the image with. User account authentication with passwords and sessions were not enforced since all users are simulated entities which can simply input their username into the proper field. In a normal situation, the agents downloading and searching the repository would have to keep a record internally of images they have checked, but an added feature for being able to mark images as "hidden" gives similar functionality with a more simplistic implementation. Hidden images are not listed on the main search page, ensuring the downloading agents do not check the same images more than once. All of the services provided by the emulated webhost are illustrated in Figure 8. The flow of information (specifically the transfer of images) is represented in the figure by the direction of the arrows.

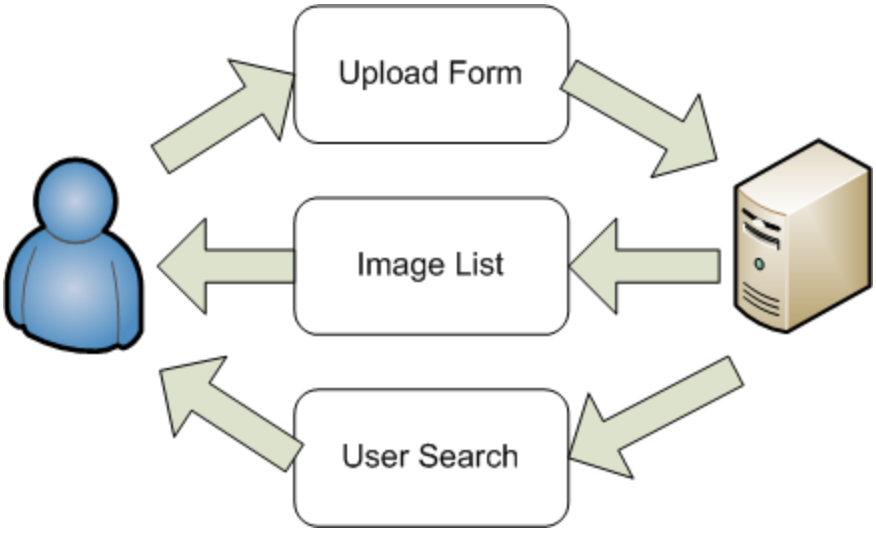


**Figure 8:** Webhost services

### 3.4.2. Emulated Smartphone Users

Thousands, if not millions, of users upload media to social media sites every day, so this activity has to be simulated for the experiments since it directly affects the performance of the covert channel. Scripts that simulate the simple act of uploading an image to the emulated image hosting website were created to simulate this activity. It simply navigates to the website and fills out the image upload form in the same way a human would, selecting a username randomly from a configuration file which is generated at runtime as described in Section 3.6. Additionally, the script records that the image was uploaded in a log file for generating statistics at the conclusion of the experiment.

A second script performs an identical operation to simulate an infected device uploading an image with data embedded in it, but it has one important difference. Before uploading the image, this script alters the first several pixels of the image, simulating a steganography algorithm. This algorithm changes their value to represent a text string

containing the username, plus a timestamp, followed by a random number to prevent a possible duplicate signature for uploads that occur at the same time.

The algorithm used to hide the data within the uploaded images is extremely simple and easily detectable, even from visual inspection. Full pixel manipulation is used to embed an unencrypted ASCII encoded string within the image. This means that the full red, green, and blue values of every pixel are altered in sequence until the full string is stored within the picture. This creates a colored band in the top left corner of the image. An example of such an artifact is shown in Figure 9. The image was magnified in order to make the band more visible. Since this research does not deal with detectability of steganography algorithms, the use of this algorithm will not affect the outcome of the experiments. If this technique should be used in the wild, however, the one implementing it should use a stronger algorithm to ensure the messages remain undetected after transmission.

**Figure 9:** Steganography artifact

### 3.4.3. Emulated Downloaders

Since the effectiveness of retrieving data uploaded by infected devices is being tested, the retrieval of that data needs to be simulated. Once again, the script navigates to the site to download and inspect images. After the images are downloaded, the same steganographic algorithm is applied in reverse, and result is compared to the pattern generated by the simulated malicious uploaders. Two distinct types of downloading agents are created, one that is aware of user accounts, and one that is not.

The agent that is unaware of user accounts searches through the images completely at random. This is to represent sites where a user account might not be associated with a particular piece of media. If no usernames are available, or if there is no way to search for media associated with a particular user, and completely random search may be the only method of media discovery.

The agent who is aware of user accounts, on the other hand, keeps a record of all the usernames associated with images it already located which contained hidden data. This algorithm begins in a completely random search phase identical to the method described earlier. The difference occurs when an item containing hidden data is identified. This algorithm saves the username associated with the image into a file. When the next search begins, all the usernames listed in that file are queried to see if they have uploaded any new images since the last search. If they have, those images are examined first. If none of them have uploaded additional media, the algorithm returns to a random search phase where it will attempt to locate additional users with infected devices. This priority system should lead to an increased discovery rate for embedded data.

### 3.5. Factors

The parameters for the system under test include the download rate, the upload rate, the number of users, and the searching algorithm used. The upload rate and number of users are actually two-part factors. Both have a clean user and malicious user component, meaning part of the total upload rate is dedicated to images with no hidden data, while another part is dedicated to images with hidden data. Similarly, the total number of users is divided between users with infected devices and users with uninfected devices. For the initial experiments comparing the two different algorithms, each value is set individually. Table 1 shows the list of initial experiments using a random search algorithm and the associated factors. Table 2 shows another set of initial experiments which used the user-aware algorithm. The data collected from these runs can be seen in Appendix A.

**Table 1:** Initial experiments with random search

| | Clean Users | Malicious Users | Clean Upload Rate | Malicious Upload Rate | Download Rate |
|---|---|---|---|---|---|
| Experiment 1 | 50 | 5 | 19 | 1 | 3 |
| Experiment 2 | 1000 | 1 | 19 | 1 | 3 |
| Experiment 3 | 1 | 1 | 10 | 10 | 1 |
| Experiment 4 | 1 | 1 | 10 | 10 | 5 |
| Experiment 5 | 50 | 5 | 10 | 10 | 3 |
| Experiment 6 | 1000 | 1 | 10 | 10 | 3 |
| Experiment 7 | 1 | 1 | 10 | 10 | 15 |
| Experiment 8 | 1 | 1 | 10 | 10 | 5 |

**Table 2:** Initial experiments with user-aware search

| | Clean Users | Malicious Users | Clean Upload Rate | Malicious Upload Rate | Download Rate |
|---|---|---|---|---|---|
| Experiment 9 | 50 | 5 | 19 | 1 | 3 |
| Experiment 10 | 1000 | 1 | 19 | 1 | 3 |
| Experiment 11 | 500 | 50 | 19 | 1 | 3 |
| Experiment 12 | 190 | 10 | 19 | 1 | 3 |
| Experiment 13 | 500 | 50 | 17 | 3 | 3 |
| Experiment 14 | 190 | 10 | 17 | 3 | 3 |
| Experiment 15 | 500 | 50 | 17 | 5 | 3 |
| Experiment 16 | 190 | 10 | 17 | 5 | 3 |
| Experiment 17 | 500 | 50 | 17 | 3 | 10 |

A fractional design was used to reduce the number of factors and levels for the primary experiments. Flickr keeps statistics for the cameras that uploaded photographs uploaded to their site including the total number of photos for a particular camera, and the number of users that utilized the service over a 25 hour period. This statistic allowed the total number of users to be reduced to a single level, however the ratio of infected and clean users still needs to be determined. The data collected to determine the number of users running the Android operating system and uploading images to Flickr is shown in Appendix B. Furthermore, the upload rate is limited by the traffic generation system discussed in Section 3.6. After some trial and error, it was determined that a maximum 20 uploads could be launched before SAST becomes unstable. Again, this limits the total upload rate to one level, and only leaves the ratio of malicious to clean uploads to be determined. Finally, it was decided that the random search algorithm would only be examined in the pilot study due to the length of time it takes to complete each run of an experiment. This effectively reduces the number of factors down to two, the download rate and the rate of infection for the malware on the simulated smartphone users. The

levels chosen for the parameters are shown in Table 3. The list of experiments resulting from the selected levels is shown in Table 4 including the derived values for upload rate and number of users.

**Table 3:** Parameters and associated levels

| Infection Rate | Download Rate |
|---|---|
| 5%<br>10%<br>25% | 1<br>2 |

**Table 4:** List of primary experiments

| | Experiment 1 | Experiment 2 | Experiment 3 | Experiment 4 | Experiment 5 | Experiment 6 |
|---|---|---|---|---|---|---|
| **Clean Users** | 1159.5 | 1159.5 | 1391.4 | 1391.4 | 1468.7 | 1468.7 |
| **Malicious Users** | 386.5 | 386.5 | 154.6 | 154.6 | 77.3 | 77.3 |
| **Clean Upload rate** | 15 | 15 | 18 | 18 | 19 | 19 |
| **Malicious Upload rate** | 5 | 5 | 2 | 2 | 1 | 1 |
| **Download rate** | 1 | 2 | 1 | 2 | 1 | 2 |
| **Time (m)** | 554 | 554 | 554 | 554 | 554 | 554 |
| **Infection Rate** | 0.25 | 0.25 | 0.1 | 0.1 | 0.05 | 0.05 |

### 3.6. Performance Metrics

One primary success metric is chosen for determining the performance of this technique. Quantifying the level of covertness of the technique is not considered for this research; rather the reliability of data recovery is examined. Instead, the outcome of every individual search is examined to see if it results in finding a piece of exfiltrated data or not. In the pilot studies, the ratio of successful searches to unsuccessful searches is compared between both the random and the user-aware algorithm. For the main experiments, only a cumulative count of successful searches is used to determine performance.

### 3.7. SAST

After creating scripts that would simulate the discrete actions of users interacting with the social media website, a program is needed to schedule when those actions will occur. Capabilities that exist within SAST are leveraged to satisfy this requirement.

SAST is most commonly used as a network traffic generator for computer network-based exercises and competitions; however its functionality is generic enough to be used to send out arbitrary network traffic, or even simply to launch command line applications at a desired frequency. Scenarios are generated with SAST, and data collected during the execution of those scenarios are used as the primary method for evaluating this technique.

The process of creating a scenario requires several steps. First, "tasks" are defined. SAST comes with a number of pre-defined tasks, such as simulating browsing a particular webpage or sending e-mails, but the parameters for these tasks can be specified by the scenario creator. Once all the tasks are created, a timeline is generated.

A timeline is composed of pairs of "tasks" and "curves", and it represents a collection of functions that perform a set of actions to mimic a particular behavior. For example, a scenario to simulate the behavior of a user's network activity during a work day may be required. To accomplish this, an e-mailing task, a web browsing task, and an FTP connection task could be created which are activities which are activities an employee would likely perform during a work day.

Each timeline is assigned to a curve, which will determine the frequency of these events over time. When selecting or creating a curve, an average number of events per minute can be specified as well as a standard deviation for that average. For example, all event frequencies can be dropped to a low level around noon to simulate a user leaving for his lunch break, or a high standard deviation can be selected for the frequency of sending and downloading e-mails. One option is to set the standard deviation at zero, causing the events to fire deterministically at regular intervals throughout the timeline.

Once a timeline is created, it is assigned to a group of actors. Actor groups exist to create a "personality type" of simulated users, such as system administrators versus accountants. Every actor created in that group will use that group's timeline once the simulation begins, but every individual actor can have special parameters specified, such as first and last name or e-mail address.

The final piece to a scenario is setting up hosts that will contain the actors. Hosts are physical computers running the Host Service Application (HSA), which is part of the SAST software. Any computer running an HSA can be used to generate traffic for part of the scenario. Different HSA's are usually deployed to represent some larger

organizational structure within a network, such as a different department or branch. Once an HSA is configured, actors can be added and the scenario can begin.

A very simple scenario is used to create the experiments. Four tasks are created, malicious uploader, clean uploader, downloader, and a reset task. The reset task simply initializes the log files, erases any remnants from previous experiments, and creates a file containing the user names to be used in the next experiment. The tasks are set to fire every 12 seconds, and the number of times each script would run when the event triggered is specified in the task itself. This allows the number of times an event would fire per minute to be controlled by altering the task rather than changing the curve. Since the scripts handle selecting which username to use, only one Actor Group with only one actor must be created for each of the two HSAs being used. While upload tasks were executed on both HSAs, the downloader task was only executed from one HSA due to its design. A diagram of all the required settings to create a complete scenario is illustrated in Figure 10.

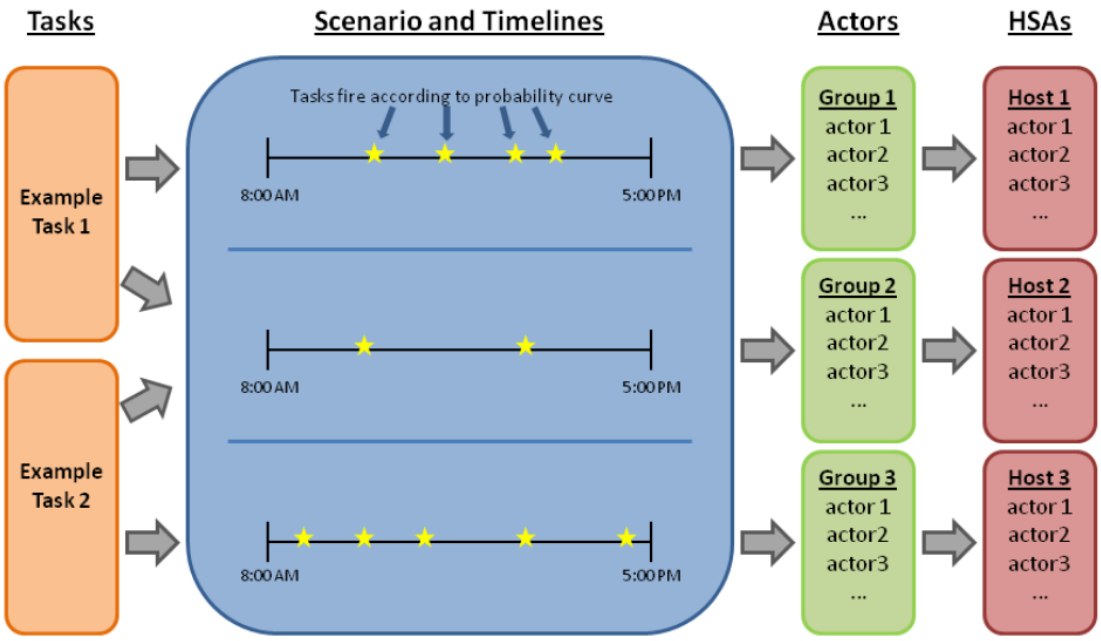


**Figure 10:** SAST scenario diagram [Mye10]

### 3.8. Log Data

For the purposes of collecting statistics from the experiments, several log files are generated during their execution. Every time a clean image is uploaded, the username and time of upload is collected. When an image with data embedded in it is uploaded, the username, time of upload, and random number used to uniquely identify the occurrence is logged. Whenever an image was downloaded, the script would record if it was a clean image or an image with data hidden in it as well as the time that it was downloaded. If the image contained hidden data, that would also be recorded along with the username associated with the upload to verify the integrity of the data. Output from the initialization script that generates the usernames is also collected for verification purposes.

### 3.9. Pilot Studies

Two simple experiments were run in order to address some of the limitations of this technique relative to Flickr's hosting services. First, a simple test is performed to ensure the integrity of the images once they are uploaded. An MD5sum of an image is generated, then that image is uploaded to Flickr. After uploading it, it is then downloaded from Flickr and an MD5sum of that image is created and compared to the MD5sum of the original image. The second experiment is to test if many images uploaded by smartphones would be publicly available, and also to make sure a downloading script would not get flagged as a bot and blocked by Flickr. A program is created using Flickr's public API that constantly searches through the most recently uploaded public photos, examines the EXIF data of those photos, and determine if they were uploaded by a smartphone. After consulting with a developer who frequently uses the Flickr API, the query rate was restricted to 30 queries per second [Nj10]. This was fastest rate at which Flickr could be safely searched without causing the script's API key to be blocked.

### 3.9. Approach Summary

In evaluating the performance of this new covert channel, all of the actions involved in the process must be simulated. The SUT includes a simulated version of Flickr, simulated smartphone users, and simulated downloading agents. Several of our factors were reduced or eliminated through a combination of Flickr provided statistics and pilot studies. The pilot studies compare two different searching algorithms, while the primary experiments evaluate more closely the conditions that would be experienced by using

Flickr. These experiments should provide a useful picture as to how this technique should perform in the wild on Flickr or a similar service.

# IV. Experimental Results

This section deals with the results of the experiments described in Chapter 3, including the pilot studies. Each section deals with a particular category of experiment, and several forms of analysis to determine the performance of this technique under a particular set of conditions. Section 4.1 discusses the results of the pilot studies, and what they suggest about the feasibility of using this technique in the wild. Section 4.2 examines the differences between the random searching algorithm and the user-aware algorithm. Section 4.3 covers the impact of the simulation environment had on the experiments is examined, and how it may have affected the experimental results. Finally, Section 4.4 provides a summary of the results.

## 4.1. Pilot Studies

In an effort to prove the practicality of this technique on an existing platform, two pilot studies are conducted. Specifically, these studies deal with the integrity of images once uploaded, and the ability for a personal computer to search for smartphone images using Flickr's API.

### 4.1.1. Data Integrity

The check to see if the data in an image remained unaltered is straightforward. After comparing the MD5sum of an original image to the MD5sum of the same image after being uploaded to Flickr resulted in a positive match if the "actual size" version of the image is selected. This means that Flickr leaves an unaltered version of uploaded images

on their site, which allows for the use of any form of image steganography. Therefore, it is unnecessary to develop a new algorithm to persist through a compression algorithm.

### 4.1.2. Searchability

The second pilot study tests the ability to search Flickr without being identified and blocked, and to test the ability to efficiently locate images uploaded by smartphones. Using the Flickr API, both of these questions could be quickly answered. After running the test script for 24 hours at an attempted rate of 30 queries per second, 24,745 images were found that were uploaded by iPhones and 1,141,806 total photos were examined. This puts the actual rate of search closer to 15.8 images per second rather than the attempted 30. This could be attributed to a higher rate of total requests to Flickr during the day, causing them to service requests slower at different times throughout the day. When compared to Flickr's self reported number of uploads for that day by iPhone users [Fli10], it was discovered that an estimated 72,395 images were uploaded via iPhone. This means one single machine was able to locate 34.18% of all images uploaded by iPhone users in a 24 hour period – a much greater number than anticipated. It is unknown if Flickr includes private images that are unsearchable in their figure for the number of images uploaded that day. This study also suggested that roughly 2% of all public images uploaded to Flickr are uploaded by an iPhone. This algorithm could easily be improved to be distributed among several searching agents, as Flickr includes an API call for retrieving public images based on a user provided time window. From the rate of image discovery listed above, three machines searching at this same rate around the clock with a

perfectly distributed algorithm could potentially analyze every image uploaded by iPhone users.

### 4.2. Algorithm Performance

The following sections deal specifically with the performance of the technique when two different searching algorithms are applied. The first is a random search, followed by a user-aware algorithm which remembers users who previously uploaded an image with hidden data. How different factors impact the accuracy of the search and how the algorithms compare against each other are both examined in detail.

#### 4.2.1. Random Algorithm

The first algorithm used simply searches through all images at random one by one, hoping to find an image that contains steganography. Several factors are altered, including the number of users, the rate of clean image uploads, and the rate of "dirty" image uploads.

While 5 experiments are run with the exact same settings, the data suggests that the simulation software was not able to perfectly recreate the scenario every time, leading to a high amount of variance in the actual values and rates chosen for the selected factors. For example, the attempted upload rate is set at 10 clean images per minute, the actual rate may be closer to 8. The reasons for this discrepancy are discussed in Section 4.3.3.1.

Despite this variation, confidence intervals can still be established for a mean rate of searching success for every individual run thanks to a large sampling size. The results of "similar" but not "identical" runs of the same experiment can then be compared to search for trends in the data.

When examining these results, only one factor causes the rate of data discovery to change – the ratio of clean image uploads to dirty image uploads. The data suggests that the ratio of searches that resulted in data being discovered is equal to the proportion of dirty uploads to total uploads with 95% confidence. The raw data associated with these experiments can be examined in Appendix A.

### 4.2.2. User-aware Algorithm

The user-aware algorithm remembers users that have uploaded dirty images in the past and prioritizes it's searching efforts on images uploaded by those users. This is expected to increase the success rate of the search. To show this, a hypothesis test is performed. Since a random search has been shown to result in successful searches equal to the ratio of malicious uploads to clean uploads over time, it is hypothesized that the mean for the success rate of the user-aware algorithm will be above the rate expected from a random search. To accomplish this, the mean of multiple runs is calculated while varying several factors between runs. The impact of varying these factors is examined, and the results are compared to the random search algorithm to evaluate if the user-aware algorithm performs better on average with statistical confidence.

According to the data in Appendix A, the random searching algorithm performs like a binomial distribution. The expected mean rate of success can be calculated simply by comparing the rate of malicious image uploads to the total number of uploads. When the same metric is used to predict the mean of the user-aware algorithm, the observed mean is always greater than the expected mean with 95% confidence. This reinforces a concept that should seem intuitively obvious - when users who have been known to be infected

with your malware are prioritized in the search, the rate of success for recovering exfiltrated data increases.

Simply stating that the algorithm performs better than a random guess is not very informative, however. Another set of experiments with new parameters that would most closely resemble conditions on Flickr will be run. Mobile device infection rate and the image searching rate are varied, and the results are examined to observe how these factors impact overall performance. For this set of experiments the parameters were selected based on Android phone usage rather than iPhone usage since the Android user base is smaller. With a smaller number of users, the experiments would have to run for less time to simulate a full day of activity. Table 3 shows the factors selected for each experiment. After examining usage statistics on Flickr's website, the total number of users uploading images from Android powered phones was determined. The percentage of infected devices and the download rate are varied between runs, and the remaining parameters are calculated based on two selected factors. The number of clean and malicious users is determined by multiplying the infection rate by the total number of users uploading from Android devices. While that number changes across experiments it is really a function of the infection rate. The upload rates were calculated in an identical manner. During initial testing, it was determined that more than 20 queries at a time would cause the image server to hang and cause the experiments to terminate early, so the total rate of simultaneous uploads never exceeds 20.

**Table 5.** List of primary experiments

| | Experiment 1 | Experiment 2 | Experiment 3 | Experiment 4 | Experiment 5 | Experiment 6 |
|---|---|---|---|---|---|---|
| **Clean Users** | 1159.5 | 1159.5 | 1391.4 | 1391.4 | 1468.7 | 1468.7 |
| **Malicious Users** | 386.5 | 386.5 | 154.6 | 154.6 | 77.3 | 77.3 |
| **Clean Upload rate** | 15 | 15 | 18 | 18 | 19 | 19 |
| **Malicious Upload rate** | 5 | 5 | 2 | 2 | 1 | 1 |
| **Download rate** | 1 | 2 | 1 | 2 | 1 | 2 |
| **Time (m)** | 554 | 554 | 554 | 554 | 554 | 554 |
| **Infection Rate** | 0.25 | 0.25 | 0.1 | 0.1 | 0.05 | 0.05 |

Figure 11 shows a cumulative count of images discovered with hidden data within them as a function of the number of searches performed for Experiment 1. The shape of the curve for individual runs appears to contain “humps” in certain places. The Figure 11 shows the data from an individual run on Experiment 1.

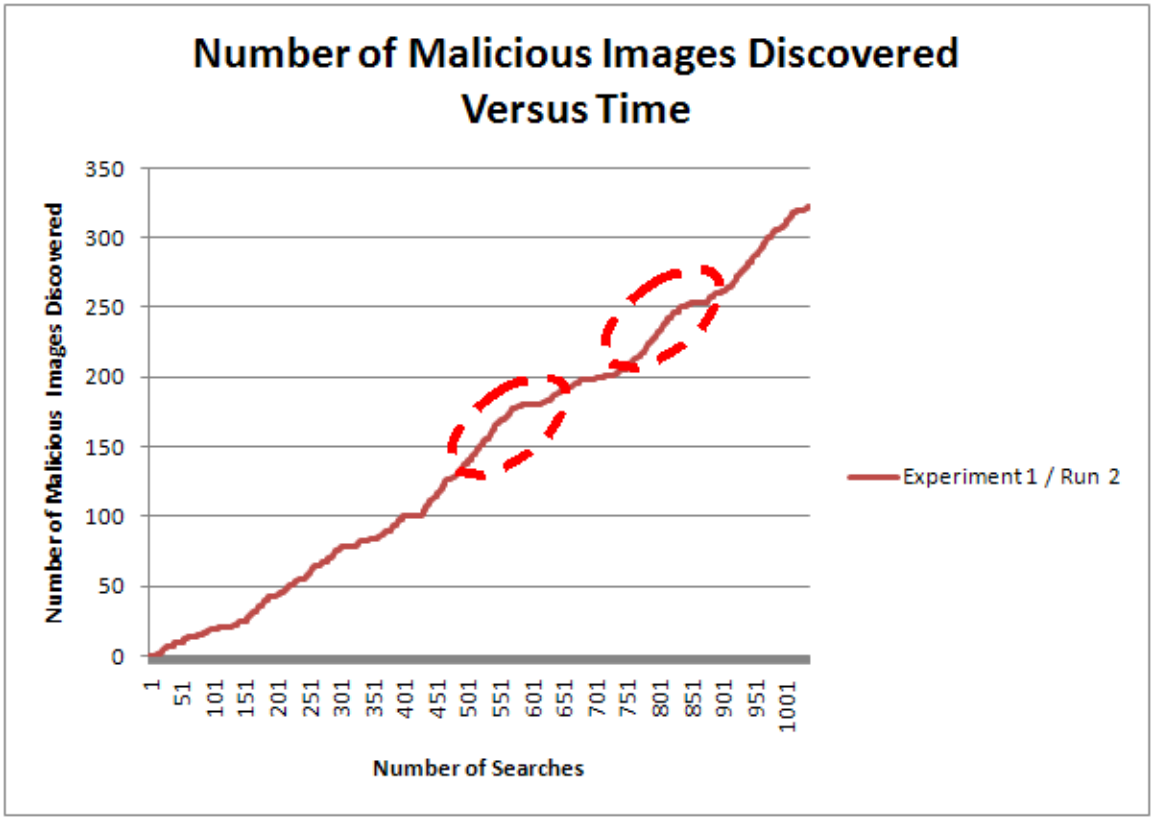


**Figure 11:** Hump pattern exhibited in individual runs

When the algorithm identifies an image with data inside it, it adds the user who uploaded that image to a list of users known to have uploaded images with embedded data. It then searches images uploaded by those users before reverting back to the random searching algorithm. If a user has uploaded several images before the searching agent discovers one of them, there is a large list of images that will quickly be located on its next searches. It is hypothesized that each hump represents the discovery of a new user that has uploaded malicious images. The hump pattern indicates the searching agent locating other images previously uploaded by the newly discovered user, and then the rate drops again after all of those images have been read and analyzed.

For the rest of the analysis, the average number of items discovered across all runs is used to average out the humps identified earlier. It also gives a more accurate description of the way this algorithm should perform over time.

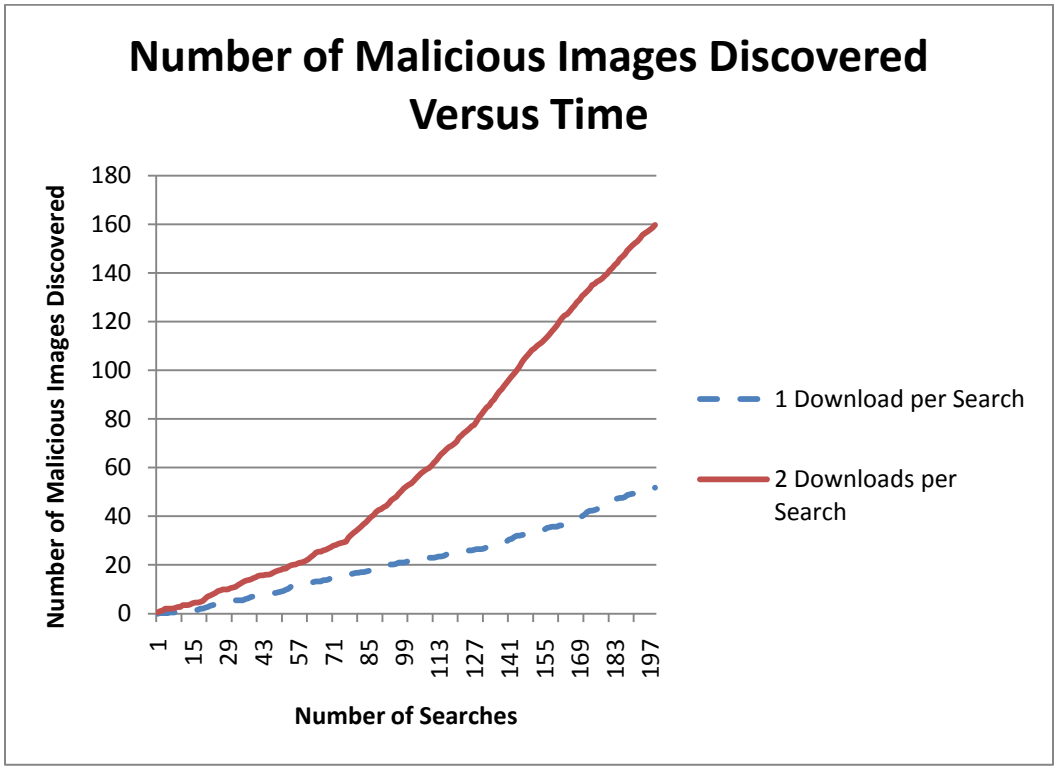


**Figure 12:** Infection rate of 25%

Figure 12 compares the rate of finding images with hidden data with an infection rate of 25%. Experiment 2 has the downloading agents searching twice as fast for the images, but it recovers data more than twice as fast as in Experiment 1. This trend is repeated when experiments 3 and 4 are compared as shown in Figure 13, as well as experiments 5 and 6 as shown in Figure 14. This implies that the relationship between number of searches and number of images discovered is a non-linear function. The shape of the graph above seems to more closely resemble a quadratic, which is also exhibited in the other experiments.

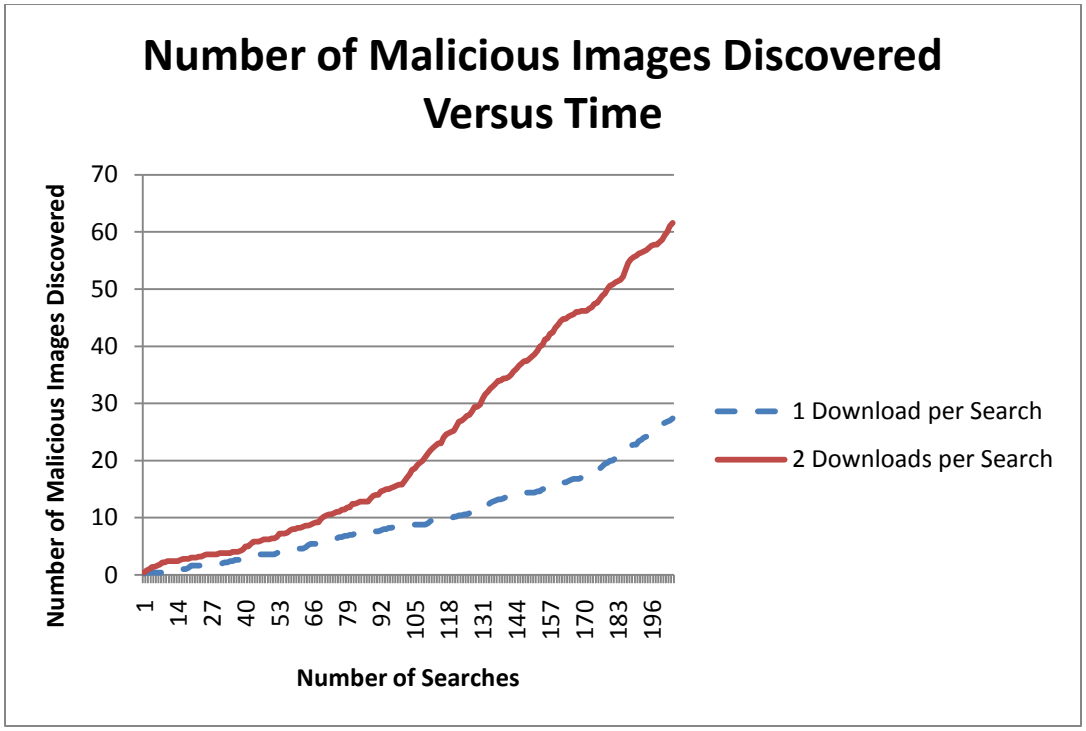


**Figure 13:** Infection rate of 10%

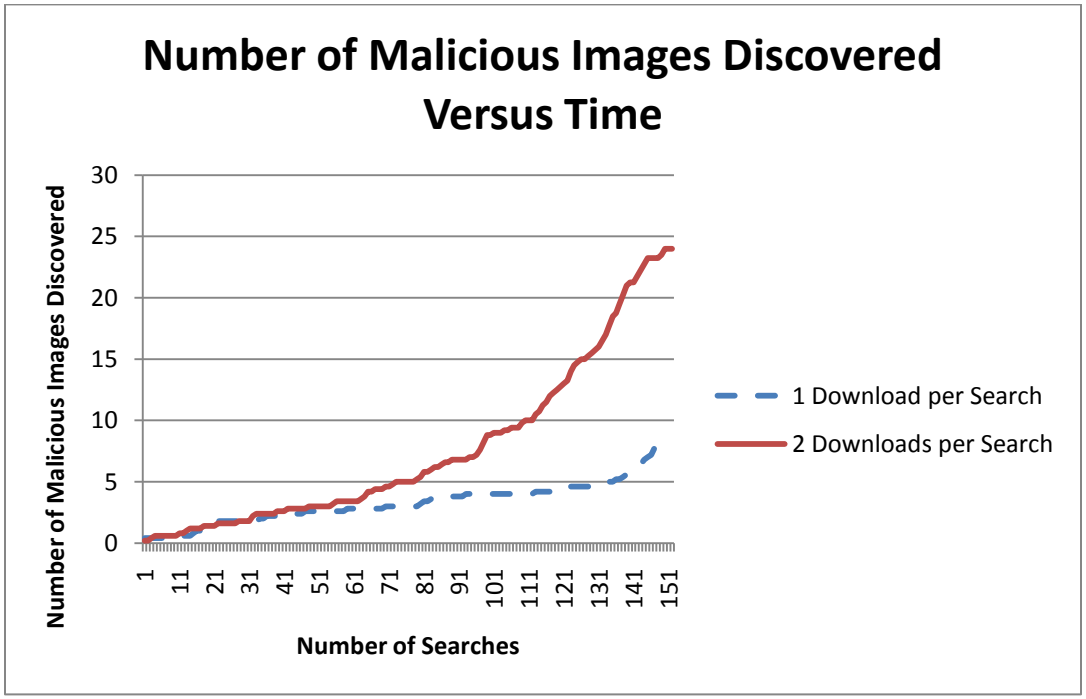


**Figure 14:** Infection rate of 5%

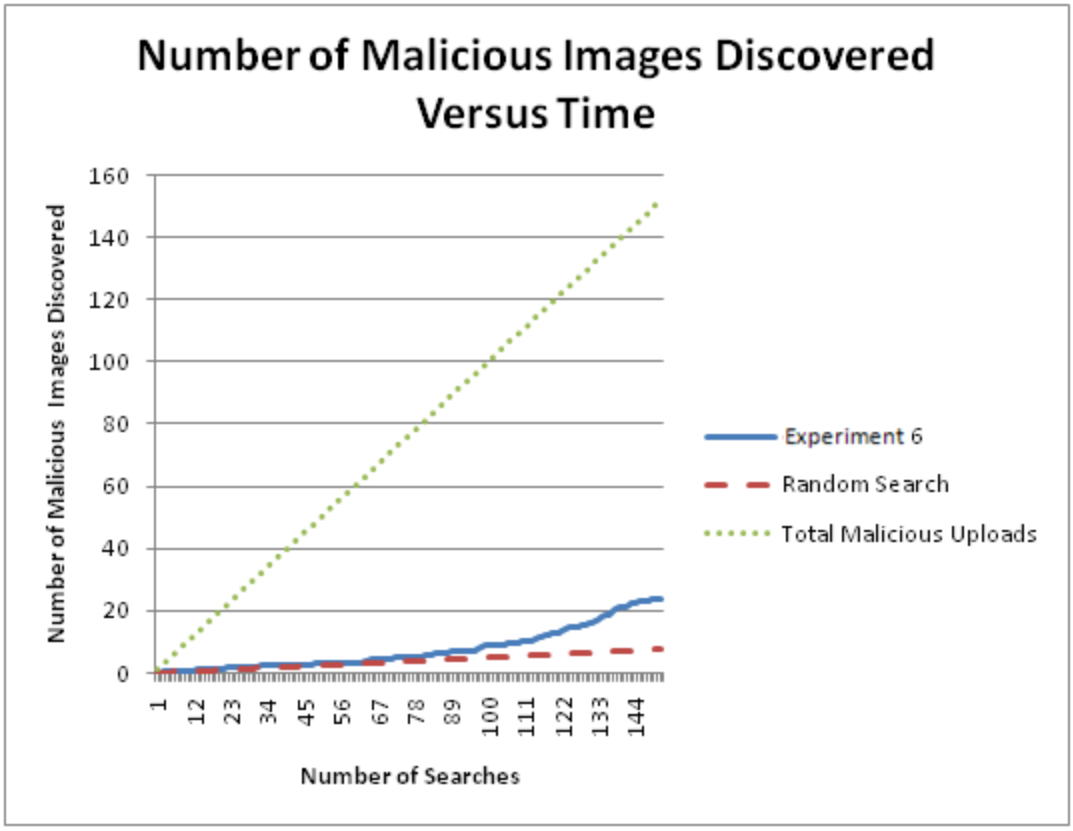


**Figure 15:** Comparing theoretical upper and lower bounds.

Figure 14 illustrates how the searching agent performs against other characteristics of the experiment. It compares the rate of image discovery to the expected rate of discovery for a random search algorithm, as well as the maximum possible upload rate. The algorithm appears to perform very similarly to a random search early in the run. This is the phase where it has not identified any users who have been uploading images with data hidden inside. As the run progresses, the algorithm performs progressively better than a random search. As time progresses, the number of discovered images for the user-aware algorithm should asymptotically approach the number of uploaded items. It is important to note that the rate of malicious image discovery can temporarily exceed the rate of

malicious image uploading, assuming the rate of search is greater than the rate of upload as is the case in Experiment 6 shown in Figure 14. If the download rate is less than the rate of upload for malicious images, the rate of discovery of new images should asymptotically approach the rate of download instead of the rate of upload, which would still imply perfect performance just throttled by the capacity to process images. While the algorithm may perform "perfectly", meaning it discovers images with hidden data 100% of the time it searches, it does not imply that every image is being located. Unless the upload rate is known (which would likely be impossible with malware loose in the wild) it cannot be known for sure that all of the images uploaded with embedded data have been located unless every single image being uploaded can be examined. This means that there is an extremely high degree of uncertainty associated with this technique.

Finally, the rate of infection is isolated as the only variable, and the download rate remains constant. Figure 16 shows this relationship for an infection rate of 25%, 10%, and 5% with a constant download rate of one download per search.

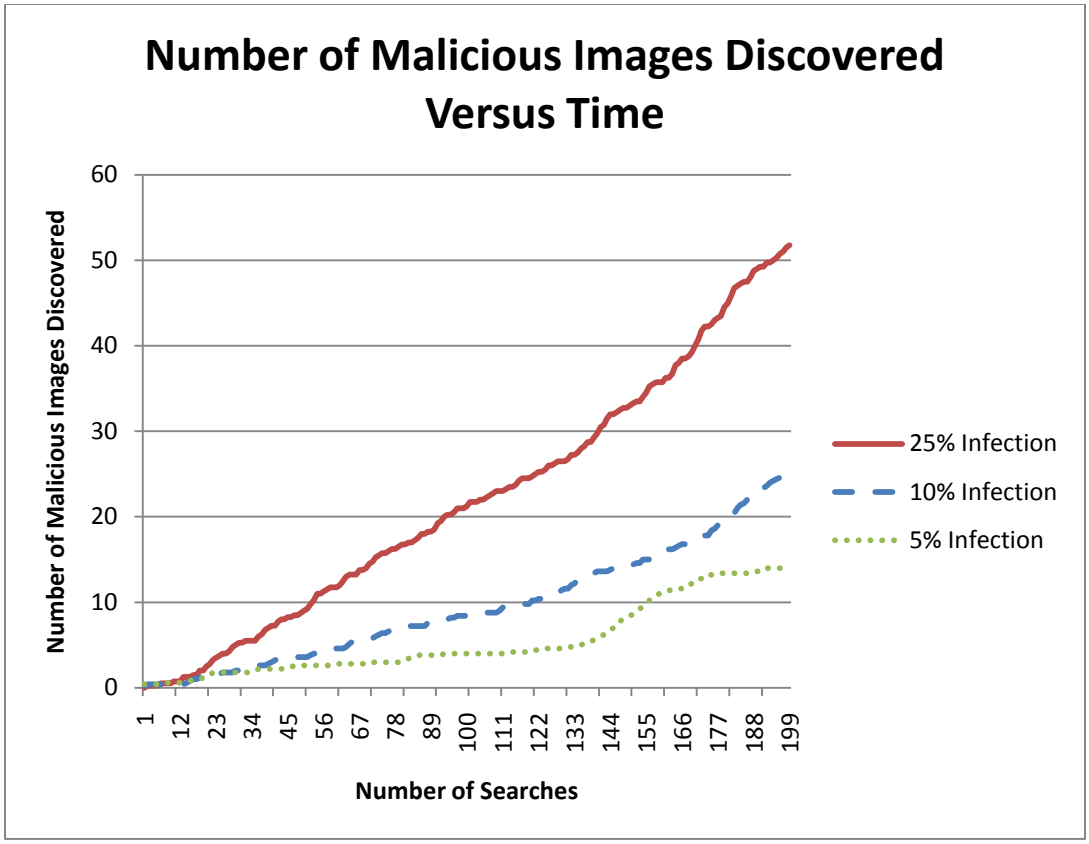


**Figure 16:** Multiple infection rates with download rate of 1

The ratio of successes between experiments is roughly proportional to the rates of infection. This is also the case when the download rate is doubled as shown in Figure 17.

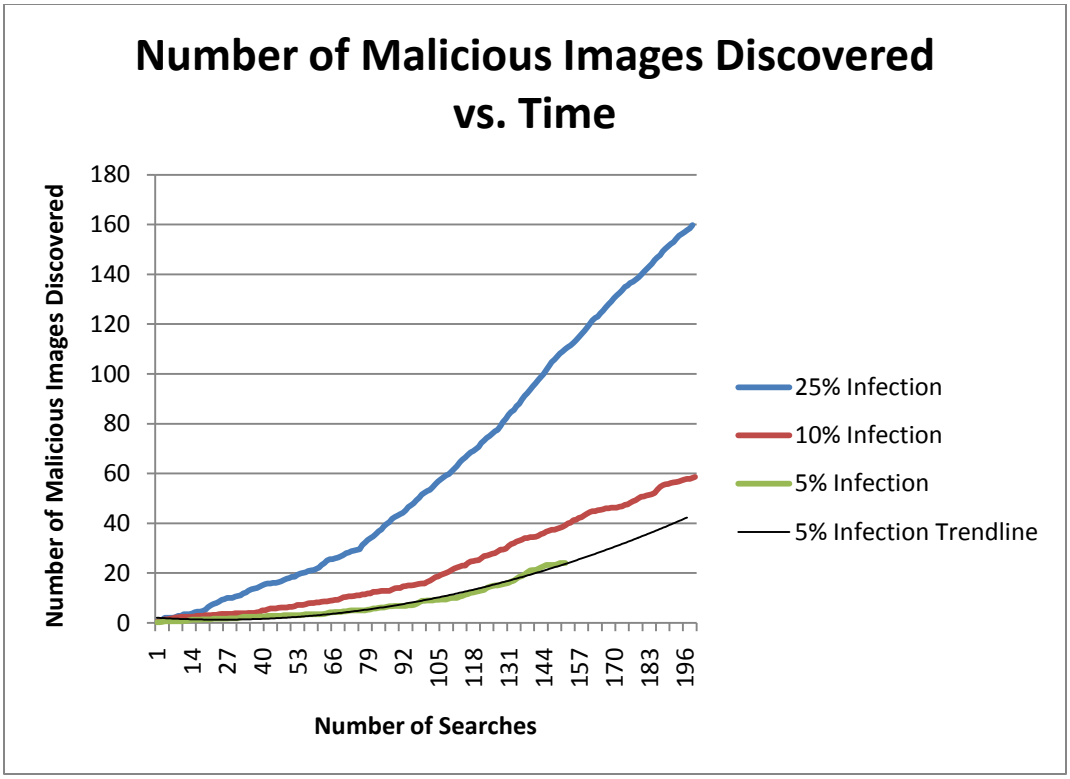


**Figure 17:** Multiple infection rates with download rate of 2

When the download rate is doubled, a similar ratio of success can be observed based on the infection rates. All of the trials for the 5% infection rate experiments terminated early. The suspected reasons for this are discussed later in this chapter. To compensate for the lack of data, a polynomial trend line of order two was added, as it seemed to best represent how the data behaves when other factors are selected. Despite being the closest approximation, it seems like it may be a fairly optimistic forecast of the performance of the algorithm under those particular factors due to the steepness of the curve relative to the other experiments which have been shown to perform better.

### 4.3. System Performance

Finally, the simulation package and web hosting framework are examined to assess their impact on the experiments. One difficulty experienced throughout all the experiments was early termination. After running for a period of time, all of the SAST tasks would succeed at a much lower rate than attempted. It is believed that individual shortcomings in both the simulation package and the web hosting framework may have compounded to amplify the problem.

#### 4.3.1. SAST

SAST was designed with network traffic generation as its primary objective. Most of the tasks it uses are Python modules that run rather efficiently. Switching to command line tasks causes a significant reduction in performance. After observing system resource usage while experiments were running, it is possible that there is a memory leak associated with command line tasks. This causes the program to halt or occasionally crash when it runs for an extended amount of time. This may have contributed to some of the events firing at irregular intervals, or terminating altogether.

#### 4.3.2. Web Host Framework

The web service representing Flickr was reproduced by creating a web application with Ruby on Rails. In the initial implementation, it was not designed to scale as well as would be necessary during the experiments, causing a large amount of data to be returned for general queries that were executed frequently. While several optimizations were made before finally running the experiment, one optimization in particular was not made that would have been very useful in reducing overhead. When a general query is made to get

a list of all images that have been uploaded, the result is not paged. All results are returned in one reply. Towards the end of the experiment, that query could have several thousands of results. As time went on, the downloading agents would make requests for that page, and it would take several seconds to generate a reply, and it might not return before the next query for the same information was made. Over time, this meant there would occasionally be tens or hundreds of requests for the same information simultaneously. That would overload the web server based on the configuration variable that determines the maximum number of simultaneous requests, causing any successive queries to hang. This would hang not only downloads, but also uploads. This problem is avoided in the Flickr API by limiting the number of items that can be returned by a single query, and it would be wise to apply this philosophy to this simulation environment in any future research. This should not have an adverse effect on the results, however, as the queries which are forced to timeout should occur completely random.

### 4.4. Results Summary

This chapter examined the results of the various experiments and pilot studies conducted to evaluate the performance of the covert channel. The pilot studies showed that many of the expected limitations of the technique are not an issue when using Flickr. In fact, the rate at which Flickr can be searched is much closer to the total upload rate than the rest of our experiments reflect, suggesting performance would be even better in practice. The user-aware algorithm performed measurably better than the random searching algorithm in almost all cases, suggesting that a social media site requiring user authentication would be preferred to using one that does not. The remainder of the

experiments supported the notion that increased search rate and increased infection rate lead to improved performance for the user-aware algorithm.

# V. Conclusions

This section identifies the significance of this research and identifies areas in which it could be extended in the future. Section 5.1 discusses the significance of this research, Section 5.2 suggests ways this research can be extended in the future, and Section 5.3 concludes this thesis with a summary.

## 5.1. Significance of Research

This research has shown the practicality and reliability of a new covert channel under several sets of conditions based around Flickr usage. Some of the most important discoveries were made in the pilot studies. It was shown that many users upload images publically from their smart phones, and searching through them at random is not entirely impractical. In fact, the rate at which images uploaded by smart phone users can be located is reasonably close to the rate at which they are uploaded, and by distributing the problem, it may not be difficult to identify and analyze every image uploaded from a smart phone.

The second significant discovery was the performance difference between recognizing and prioritizing different users. Some sites do not provide a mechanism to associate a username with a piece of uploaded content, while others require it. This may suggest that someone intending to implement this technique using a service other than Flickr may want to choose one where authentication is required. It was also learned that infection rate is proportional to the rate of image discovery, particularly in the case when a user-aware algorithm is being used

Finally, a third discovery was that in the case of the user-aware algorithm, the rate of recovering exfiltrated data approaches the rate of malicious image uploads over time. Theoretically, assuming the rate of users joining the social media site being used is small, the algorithm should perform perfectly after a long ramp up period.

### 5.2. Future Research

The logical next step in this research would be to create a proof of concept program for smart phones. The Android operating system would be ideal due to the open nature of the platform and how easily the device can be unlocked. Once it is possible to use this technique in the wild, additional experiments should be conducted. The Flickr API should be used to search for images actually uploaded by phones running the proof of concept code. Special attention should be paid to the number of users using the Flickr service daily. It should be noted that all of the experiments in this research were conducted with a static number of users. On real sites, users are joining all the time. More accurate experiments could better reflect the changing number of users over time. Additionally, researchers should look for more ways to optimize the searching algorithm without increasing the possibility of being detected. Finally, this research should be coupled with research on the topic of how malware spreads through social networking sites. This technique would be far more effective if targeted specifically against users of the service being leveraged for data exfiltration.

### 5.3. Summary

This research aimed to create a novel covert channel for data exfiltration. It accomplished that goal by combining techniques used in steganography and existing

passive covert channels to accomplish that goal on smartphones and mobile devices. It has been shown that this technique is both possible and practical when used in conjunction with a social media website such as Flickr where the identified limitations are insignificant. Two different searching algorithms were compared, and it was found that search accuracy can be greatly improved by being aware of authenticated users. Additionally, search accuracy can be increased by improving the rate of search and the infection rate of the targeted devices. Considering the results and conclusions of this research, it has been shown that Hijacking User Uploads to Online Persistent Data Repositories is a novel, viable, and practical method for conducting covert data exfiltration.

## Appendix A

Table 6 and Table 7 are lists of experiments and factors for testing if the random searching algorithm performed as expected, as well as how it performed relative to the user-aware algorithm. The results of the experiments follow.

**Table 6:** Initial experiments with random search

| | Clean Users | Malicious Users | Clean Upload Rate | Malicious Upload Rate | Download Rate |
|---|---|---|---|---|---|
| Experiment 1 | 50 | 5 | 19 | 1 | 3 |
| Experiment 2 | 1000 | 1 | 19 | 1 | 3 |
| Experiment 3 | 1 | 1 | 10 | 10 | 1 |
| Experiment 4 | 1 | 1 | 10 | 10 | 5 |
| Experiment 5 | 50 | 5 | 10 | 10 | 3 |
| Experiment 6 | 1000 | 1 | 10 | 10 | 3 |
| Experiment 7 | 1 | 1 | 10 | 10 | 15 |
| Experiment 8 | 1 | 1 | 10 | 10 | 5 |

**Table 7:** Initial experiments with user-aware search

| | Clean Users | Malicious Users | Clean Upload Rate | Malicious Upload Rate | Download Rate |
|---|---|---|---|---|---|
| Experiment 9 | 50 | 5 | 19 | 1 | 3 |
| Experiment 10 | 1000 | 1 | 19 | 1 | 3 |
| Experiment 11 | 500 | 50 | 19 | 1 | 3 |
| Experiment 12 | 190 | 10 | 19 | 1 | 3 |
| Experiment 13 | 500 | 50 | 17 | 3 | 3 |
| Experiment 14 | 190 | 10 | 17 | 3 | 3 |
| Experiment 15 | 500 | 50 | 17 | 5 | 3 |
| Experiment 16 | 190 | 10 | 17 | 5 | 3 |
| Experiment 17 | 500 | 50 | 17 | 3 | 10 |

The following table lists the result of the experiments above. If nothing is listed in a row, is due to a lack of data for the particular trial. The rightmost column suggests that

the observed mean is equal to the expected mean with 95% confidence if an "=" occupies the cell, or that the observed mean was greater than the expected mean with 95% confidence if an ">" occupies the cell, or that the observed mean was less than the expected mean with 95% confidence if an "<" occupies it. The expected mean was calculated by assuming the trials were Binomial, meaning mutually independent and exclusive events, as was the case with the random algorithm.

**Table 8:** Results of initial experiments

| | Positive Hits | Number of Samples | Success Rate | Expected Success Rate | Expected Rate vs. Actual Rate of Success |
|---|---|---|---|---|---|
| **Experiment 1** | 13 | 294 | 0.044 | 0.05 | = |
| | 15 | 297 | 0.051 | 0.05 | = |
| | 14 | 297 | 0.047 | 0.05 | = |
| | 20 | 297 | 0.067 | 0.05 | = |
| | 12 | 294 | 0.041 | 0.05 | = |
| **Experiment 2** | 13 | 294 | 0.044 | 0.05 | = |
| | 17 | 294 | 0.058 | 0.05 | = |
| | 13 | 297 | 0.044 | 0.05 | = |
| | 9 | 294 | 0.031 | 0.05 | = |
| | 13 | 246 | 0.053 | 0.061 | = |
| **Experiment 3** | 49 | 100 | 0.49 | 0.503 | = |
| | 44 | 99 | 0.444 | 0.5 | = |
| | 38 | 98 | 0.388 | 0.503 | < |
| | 35 | 98 | 0.357 | 0.5 | < |
| | 48 | 99 | 0.485 | 0.5 | = |
| **Experiment 4** | 236 | 460 | 0.513 | 0.5 | = |
| | 232 | 728 | 0.319 | 0.341 | = |
| | 255 | 767 | 0.332 | 0.349 | = |
| | 261 | 759 | 0.344 | 0.353 | = |
| | 165 | 355 | 0.465 | 0.513 | = |
| **Experiment 5** | 146 | 296 | 0.493 | 0.497 | = |
| | 141 | 294 | 0.48 | 0.497 | = |
| | 143 | 294 | 0.486 | 0.497 | = |
| | 146 | 294 | 0.497 | 0.503 | = |

| | | | | | |
|---|---|---|---|---|---|
| | 149 | 294 | 0.507 | 0.497 | = |
| **Experiment 6** | 117 | 244 | 0.48 | 0.509 | = |
| | 727 | 1461 | 0.498 | 0.5 | = |
| | 716 | 1461 | 0.49 | 0.489 | = |
| | 93 | 150 | 0.62 | 0.572 | = |
| | 166 | 312 | 0.532 | 0.507 | = |
| **Experiment 7** | 449 | 792 | 0.567 | 0.6 | = |
| | 171 | 364 | 0.47 | 0.502 | = |
| | 208 | 450 | 0.462 | 0.49 | = |
| | 153 | 297 | 0.515 | 0.5 | = |
| | 247 | 492 | 0.502 | 0.501 | = |
| **Experiment 8** | 152 | 336 | 0.452 | 0.501 | > |
| | 101 | 294 | 0.344 | 0.05 | > |
| | 66 | 297 | 0.222 | 0.05 | > |
| | 102 | 294 | 0.347 | 0.05 | > |
| | 85 | 294 | 0.289 | 0.05 | > |
| **Experiment 9** | 101 | 294 | 0.344 | 0.05 | > |
| | 98 | 297 | 0.33 | 0.05 | > |
| | 100 | 297 | 0.337 | 0.05 | > |
| | 100 | 297 | 0.337 | 0.05 | > |
| | 98 | 297 | 0.33 | 0.05 | > |
| **Experiment 10** | 97 | 284 | 0.342 | 0.05 | > |
| | 42 | 297 | 0.141 | 0.051 | > |
| | 39 | 297 | 0.131 | 0.05 | > |
| | 41 | 294 | 0.139 | 0.051 | > |
| | 43 | 294 | 0.146 | 0.05 | > |
| **Experiment 11** | 33 | 297 | 0.111 | 0.05 | > |
| | 62 | 288 | 0.215 | 0.05 | > |
| | 94 | 294 | 0.32 | 0.05 | > |
| | 42 | 297 | 0.141 | 0.05 | > |
| | 82 | 296 | 0.277 | 0.05 | > |
| **Experiment 12** | 34 | 259 | 0.131 | 0.05 | > |
| | 112 | 273 | 0.41 | 0.13 | > |
| | 41 | 294 | 0.139 | 0.051 | > |
| **Experiment 13** | 43 | 294 | 0.146 | 0.05 | > |
| | 33 | 297 | 0.111 | 0.05 | > |
| | 62 | 288 | 0.215 | 0.05 | > |
| | 94 | 294 | 0.32 | 0.05 | > |
| **Experiment 14** | 42 | 297 | 0.141 | 0.05 | > |
| | 34 | 259 | 0.131 | 0.05 | > |

| | | | | | |
|---|---|---|---|---|---|
| | 190 | 486 | 0.391 | 0.15 | > |
| | 225 | 490 | 0.459 | 0.15 | > |
| **Experiment 15** | 221 | 490 | 0.451 | 0.149 | > |
| | 209 | 490 | 0.427 | 0.149 | > |
| | 202 | 490 | 0.412 | 0.15 | > |
| | 284 | 495 | 0.574 | 0.15 | > |
| | 301 | 490 | 0.614 | 0.15 | > |
| **Experiment 16** | 267 | 490 | 0.545 | 0.15 | > |
| | 304 | 490 | 0.62 | 0.15 | > |
| | 293 | 490 | 0.598 | 0.149 | > |
| | 317 | 957 | 0.331 | 0.152 | > |
| | 322 | 923 | 0.349 | 0.152 | > |
| **Experiment 17** | 330 | 910 | 0.363 | 0.149 | > |
| | 303 | 873 | 0.347 | 0.152 | > |
| | 293 | 964 | 0.304 | 0.154 | > |
| | 346 | 979 | 0.353 | 0.151 | > |
| | 388 | 980 | 0.396 | 0.151 | > |

The experiments using the random search algorithm resulted in an observed mean that was within a 95% confidence interval of the binomial mean, implying each trial was mutually exclusive and independent. There are a few runs that were exceptions, but the number of runs that failed the hypothesis test was consistent with what would be expected with a 95% confidence interval – roughly 5%. The experiments using the improved algorithm performed better than a Binomial on every single run with 95% confidence.

## Appendix B

The following table contains statistics collected from Flickr's website on Android camera phone usage [Fli10].

**Table 9:** List of Android Phone usage on Flickr

| Camera | Average Daily Users | Total Number of Uploads | Number of Users Yesterday | Number of Uploads Yesterday |
|---|---|---|---|---|
| Acer beTouch E110 | 0 | 0 | 0 | 0 |
| Acer Liquid - A1 | 0 | 0 | 0 | 0 |
| Dell Mini 3i | 0 | 0 | 0 | 0 |
| Dell Streak | 0 | 0 | 0 | 0 |
| Geeksphone One | 0 | 0 | 0 | 0 |
| HTC Desire/Passion/Bravo | 211 | 104041 | 288 | 1625 |
| HTC Droid Incredible | 138 | 93964 | 242 | 2299 |
| HTC Eris - Desire | 187 | 241957 | 195 | 1494 |
| HTC G1 - Dream | 114 | 439754 | 88 | 511 |
| HTC Legend | 0 | 0 | 0 | 0 |
| HTC Magic MyTouch 3g | 83 | 161703 | 78 | 807 |
| HTC Nexus One | 6 | 3391 | 1 | 17 |
| HTC Sprint Hero | 139 | 220212 | 137 | 998 |
| HTC Sprint Hero 200 | 195 | 296734 | 204 | 1737 |
| HTC Tattoo | 0 | 0 | 0 | 0 |
| Huawei 845 Vodaphone | 0 | 0 | 0 | 0 |
| Huawei Pulse | 0 | 0 | 0 | 0 |
| Huawei Pulse Mini | 0 | 0 | 0 | 0 |
| LG Ally | 0 | 0 | 0 | 0 |
| LG Eve GW620 | 0 | 0 | 0 | 0 |
| LG GT450 Optimus | 0 | 0 | 0 | 0 |
| LG Optimus Q - LU2300 | 0 | 0 | 0 | 0 |
| Motorola Backflip | 0 | 0 | 0 | 0 |
| Motorola Cliq | 40 | 40041 | 44 | 161 |
| Motorola Cliq XT | 0 | 0 | 0 | 0 |
| Motorola Devour | 0 | 0 | 0 | 0 |
| Motorola Droid - Milestone | 236 | 219698 | 269 | 1431 |
| Motorola Motoroi | 0 | 0 | 0 | 0 |
| Samsung Behold II | 0 | 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| Samsung Galaxy A | 0 | 0 | 0 | 0 |
| Samsung Galaxy S | 0 | 0 | 0 | 0 |
| Samsung Moment | 0 | 0 | 0 | 0 |
| Samsung Spica i5700 | 0 | 0 | 0 | 0 |
| Sony Ericson Xperia X10 | 0 | 0 | 0 | 0 |
| Sony Ericson Xperia X10 Mini | 0 | 0 | 0 | 0 |
| Tmobile MyTouch 3g Slide | 0 | 0 | 0 | 0 |
| Total: | 1349 | 1821495 | 1546 | 11080 |

[EGH+89] T. Eisenberg, D. Gries, J. Hartmanis, D. Holcomb, M. S. Lynn, and T. Santoro. The Cornell Commission: on Morris and the worm. *Communications of the ACM*, 32(6):706–709, 1989.

[ER89] M.W. Eichin and J.A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. pages 326 –343, may 1989.

[FGD01] J. Fridrich, M. Goljan, and Rui Du. Detecting LSB Steganography in Color, and Gray-scale Images. *Multimedia, IEEE*, 8(4):22 –28, Oct.-Dec. 2001.

[Fli10] Flickr. Camera finder, August 2010.

[HB05] Greg Hoglund and Jamie Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.

[Kur89] S. Kurzban. Viruses and Worms - What Can You Do? *SIGSAC Review*, 7(1):16–32, 1989.

[LMS10] J. Lubacz, W. Mazurczyk, and K. Szczypiorski. Vice Over IP. *Spectrum, IEEE*, 47(2):42 –47, February 2010.

[LTG09] Wei Lu, Mahbod Tavallaee, and Ali A. Ghorbani. Automatic Discovery of Botnet Communities on Large-scale Communication Networks. In *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 1–10, New York, NY, USA, 2009. ACM.

[MPS+03] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *Security Privacy, IEEE*, 1(4):33 – 39, July-Aug. 2003.

[MSC02] David Moore, Colleen Shannon, and K Claffy. Code-red: A Case Study on the Spread and Victims of an Internet Worm. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 273–284, New York, NY, USA, 2002. ACM.

[Mye10] J. Myers. A Dynamically Configurable Log-based Distributed Security Event Detection Methodology Using Simple Event Correlator. Master's thesis, Air Force Institute of Technology, 2010.

[Nac97] Carey Nachenberg. Computer virus-antivirus coevolution. *Communications of the ACM*, 40(1):46–51, 1997.

[Nj10] Norby and jedrek. flickr, August 2010. IRC conversation.

[Por09] Phillip Porras. Inside Risk: Reflections on Conficker. *Communications of the ACM*, 52(10):23–24, 2009.

[SGCC+09] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your Botnet is my Botnet: Analysis of a Botnet Takeover. In *CCS '09: Proceedings of the 16th ACM conference on Computer and Communications Security*, pages 635–647, New York, NY, USA, 2009. ACM.

[Sha10] Noah Shachtman. FBI: Spies hid secret messages on public websites, June 2010.

[Spa94] Eugene H. Spafford. Computer Viruses as Artificial Life. *Artificial Life*, 1(3):249–265, 1994.

[TZH+08] Hui Tian, Ke Zhou, Yongfeng Huang, Dan Feng, and Jin Liu. A Covert Communication Model Based on Least Significant Bits Steganography in Voice over IP. pages 647 –652, nov. 2008.

[WPSC03] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *WORM '03: Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 11–18, New York, NY, USA, 2003. ACM.

[ZAB07] S. Zander, G. Armitage, and P. Branch. Covert Channels and Countermeasures in Computer Network Protocols[reprinted from IEEE communications surveys and tutorials]. *Communications Magazine, IEEE*, 45(12):136 –142, December 2007.

## REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| **1. REPORT DATE** *(DD-MM-YYYY)* | **2. REPORT TYPE** | **3. DATES COVERED** *(From – To)* |
|---|---|---|
| 16-09-2010 | Master's Thesis | Sept 2008 – Sept 2010 |

**4. TITLE AND SUBTITLE**

HIJACKING USER UPLOADS TO ONLINE PERSISTENT DATA REPOSITORIES FOR COVERT DATA EXFILTRATION

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Barnard, Curtis P.

**5d. PROJECT NUMBER**
N/A

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)**
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-7765

**8. PERFORMING ORGANIZATION REPORT NUMBER**

AFIT/GCO/ENG/10-16

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Robert Kaufman
318 Information Operations Group/DD
688th Information Operations Wing
102 Hall Blvd, Suite 311, San Antonio, TX 78243-7078
(210) 925-4425

**10. SPONSOR/MONITOR'S ACRONYM(S)**
IOG/DD

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**
(210) 925-4425

**12. DISTRIBUTION/AVAILABILITY STATEMENT**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

As malware has evolved over the years, it has gone from harmless programs that copy themselves into other executables to modern day botnets that perform bank fraud and identity theft. Modern malware often has a need to communicate back to the author, or other machines that are also infected. Several techniques for transmitting this data covertly have been developed over the years which vary significantly in their level of sophistication.

This research creates a new covert channel technique for stealing information from a network by piggybacking on user-generated network traffic. Specifically, steganography drop boxes and passive covert channels are merged to create a novel covert data exfiltration technique.

This technique revolves around altering user supplied data being uploaded to online repositories such as image hosting websites. It specifically targets devices that are often used to generate and upload content to the Internet, such as smartphones. The reliability of this technique is tested by creating a simulated version of Flickr as well as simulating how smartphone users interact with the service. Two different algorithms for recovering the exfiltrated data are compared. The results show a clear improvement for algorithms that are user-aware. The results continue on to compare performance for varying rates of infection of mobile devices and show that performance is proportional to the infection rate.

**15. SUBJECT TERMS**
COVERT CHANENLS, STEGANOGRAPHY, BOTNETS

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **REPORT** | **ABSTRACT** | **c. THIS PAGE** | | | Dr. Barry Mullins |
| **U** | **U** | **U** | **UU** | 87 | **19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 7979 (bmullins@afit.edu) |

**Standard Form 298 (Rev: 8-98)**
Prescribed by ANSI Std. Z39-18