Air Force Institute of Technology
AFIT Scholar

Theses and Dissertations

9-15-2011

Twitter Malware Collection System: An Automated URL Extraction and Examination Platform

Benjamin B. Kuhar

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Computer and Systems Architecture Commons, and the Digital Communications and Networking Commons

**TWITTER MALWARE COLLECTION SYSTEM: AN AUTOMATED URL
EXTRACTION AND EXAMINATION PLATFORM**

THESIS

Benjamin B. Kuhar

AFIT/GCO/ENG/11-07

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/11-07

TWITTER MALWARE COLLECTION SYSTEM: AN AUTOMATED URL EXTRACTION AND EXAMINATION PLATFORM

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

Benjamin B. Kuhar, BS

August 2011

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

TWITTER MALWARE COLLECTION SYSTEM: AN AUTOMATED URL EXTRACTION AND EXAMINATION PLATFORM

Benjamin B. Kuhar, BS

Approved:

(signature)	9/7/11
Rusty Baldwin, PhD (Chairman)	Date
(signature)	9/6/2011
William Kimball (Member)	Date
(signature)	7 Sep 2011
Richard Raines, PhD (Member)	Date

AFIT/GCO/ENG/11-07

Abstract

As the world becomes more interconnected through various technological services and methods, the threat of malware is increasingly looming overhead. One avenue in particular that is examined in this research is the social networking service Twitter.

This research develops the Twitter Malware Collection System (TMCS). This system gathers Uniform Resource Locators (URLs) posted on Twitter and scans them to determine if any are hosting malware. This scanning process is performed by a cluster of Virtual Machines (VMs) running a specified software configuration and the execution prevention system known as ESCAPE which detects malicious code. When a URL is detected by a TMCS VM instance to be hosting malware, a dump of the web browser used is created to determine what kind of malicious activity has taken place and also how this activity was allowed.

After collecting over a period of 40 days, and processing a total of 466,237 URLs twice in two different configurations, one consisting of a vulnerable Windows XP SP2 setup and the other consisting of a fully patched and updated Windows Vista setup, a total of 2,989 dumps were created by TMCS based on the results generated by ESCAPE.

Acknowledgments

I would like to thank my good friends and family members both near and far.

Without you, I don't believe this research effort would have been possible.

Table of Contents

List of Figures

List of Tables

**TWITTER MALWARE COLLECTION SYSTEM: AN AUTOMATED URL
EXTRACTION AND EXAMINATION PLATFORM**

I.   Introduction

Imagine for a moment that a famous comedic movie actor has a Twitter account with a massive number of followers, say, in excess of 1,200,000. These followers all receive updates when this actor posts an update to Twitter. With this large of a following, something posted by this actor is "heard" by a large audience. Now, assume that someone has come up with a scheme to steal from people using a method that requires a significant number of trusting people is able to gain control of the actor's Twitter account. This thief, now having the ability to send a message to over 1.2 million users, decides to take advantage of this massive group of admirers by posting a link that is said to contain a screensaver for the actor's upcoming new movie. The link instead contains a piece malware that steals from the victim's banking account. There's no need to imagine, this event has actually taken place. The actor Simon Pegg's Twitter account was hijacked and used to spread a banking Trojan [1]; and this is not the first time such an event has occurred and will likely not be the last time.

1.1 Problem Background

As consumer technology advances and becomes more affordable, more users are able to experience the amazing and revolutionary things made possible that were not previously. The global Internet is an example of such technology. Through the use of the Internet, information spreads to far away destinations at incredible speeds. This world-wide connection hosts both legitimate and illegitimate actions. A website can host

an online store through which tangible products can be purchased and shipped half way across the globe while another website can deceive users into downloading and installing fake anti-virus scanning software. Phone calls can be made using the Voice over Internet Protocol to family members across the ocean or malicious executables can be spread through a link contained within a status posted to the social networking service Twitter. As time marches on, the ability to discern whether or not something stumbled upon while using the Internet as advantageous or malicious becomes increasingly difficult. The reason for this is that the creators of the illegitimate actions are progressively able to take advantage of not only a computing system, but also the user of the system. Determining ways to deal with these sorts of malicious actions proves problematic and requires much effort and thought.

1.2 Goals

The primary goal of this research is to create a system to scan hyperlinks posted on the social networking service Twitter. The system determines whether or not malicious activity is detected at these links.

An additional goal of this research is to create an archive of the offenders that are deemed malicious to study the means and methods by which the malicious activity takes place.

1.3 Document Outline

Chapter 2 of this document provides a literature review relevant to this research effort. Chapter 3 provides the experimental methodology for the system. Chapter 4

provides analysis on the results generated by the system. Chapter 5 provides a concluding summary of the findings of the system as well as future research endeavors.

II. Literature Review

2.1 Malware Overview

This section of chapter two provides a short history of malware as well providing some pertinent definitions.

2.1.1 Defining Malware

Malware, short for malicious software, is software whose purpose is to exfiltrate data or cause damage to one or more computer systems without the system owner's explicit permission [2]. The existence of malware is common knowledge as the media frequently describes wide-spread attacks [3].

Malicious software itself, though, is not new. It has become more prevalent due to the wide spread connectivity information via personal computers and the Internet [3]. One of the earliest notable pieces of malicious software is the fork bomb. A fork bomb is a program or shell script that rapidly creates new processes via the fork() system call. The goal of a fork bomb is to consume entries in the process table and thereby cause a denial of service which will bring the affected computer to a halt [4].

Malware is increasingly becoming more sophisticated, stealthy, and even weapon-like as was recently seen with the so-called "cyber weapon" Stuxnet [5]. Malware authors take advantage of things such as delays between patch creation and patch installation, user susceptibility to social engineering, and the likeliness of users to pursue "attractive" material. These authors realize that systems, as well as their users, are vulnerable.

2.1.2 Trojan Horses

A Trojan Horse is a piece of software that appears legitimate to the user, but contains unknown functionality which can be leveraged in order to gain a level of control on the victim's computer [6]. This type of malware, along with others that provide unauthorized access to a user's computer, have the potential for dire consequences.

2.1.3 Rogueware

Within the past three years, fake anti-virus products, also known as rogueware, have skyrocketed in popularity among malware distributors. These pieces of rogueware trick the affected user into paying money for a license to remove what the programs identify as "infected" files. As a result, the false licenses that are purchased add up to a significant amount of capital for the malware distributors. The ability to play on the fears of people rather than the vulnerabilities contained within a user's system itself allows the creators of these rogueware items to scam bystanders for their money [7].

2.2 *Malware and Exploit Collection Systems*

There are many different variations of malware and exploit collectors, or crawlers, in existence. Of particular interest with regards to this research effort are the ones that use crawlers to identify Uniform Resource Locators (URLs) with malicious content. Some of these crawlers are explored in the following sections to determine how they are useful in different applications and how similar methods may be applied to this research.

2.2.1 Strider HoneyMonkey

The Strider HoneyMonkey project is a Microsoft sponsored system. This system visits various websites with the intent of finding zero-day exploits as well as established exploits that can compromise an unpatched system [8].

The Strider system checks for the illegitimate and unsanctioned creation of files and system configuration changes and is combined with a HoneyMonkey, which is essentially a proactive honeypot, to determine when an exploit has successfully executed. With this system, multiple variations of the HoneyMonkey execute on different Virtual Machines to test different levels of patches and the levels of "aggressiveness" from various websites [8].

The HoneyMonkey system detects exploits through a three step process. The first step, known as scalable mode, visits a configured number of URLs at the same time from a single virtual machine. If an exploit is detected, the system will check one URL per virtual machine and re-test each of those URLs to determine which specific URL contains the exploit. In the second step, the HoneyMonkeys determines what pages are malicious through recursive redirection analysis by examining the URLs contained within the initial page. Then, in step three, HoneyMonkeys continuously scan the results from step two within fully updated virtual machines to determine if any exploits are leveraging zero-days [8].

Since signature-based detection tends to be a cat and mouse process, Strider HoneyMonkey uses a black-box non-signature based approach. A HoneyMonkey is run that launches a new instance of a browser to defeat any code containing a timer that delays execution of said code. Since user interaction is not incorporated into the

HoneyMonkey, any modifications made to the system outside of the browser's authorized area of operation indicate that an exploit has successfully run. This approach detects exploits that leverage both known and unknown vulnerabilities. After the HoneyMonkey has visited a requested URL, the virtual machine is examined to detect if there are any noticeable executables created, if any files have been modified outside the permitted folders, if any new processes have been created, if any windows registry configuration changes have occurred including both the addition and modification of keys, if any known vulnerabilities have been leveraged, and finally, if any redirect-URLs have been visited [8].

Within redirection analysis, a large number of URLs deemed malicious in step one were content providers serving up attention-getting items to lure in potential victims. If successful, traffic is redirected to the actual exploit providers which infect or compromise the victims' machines [8].

In generating URLs to crawl, the Strider HoneyMonkey team used URLs from a search for sites that were known to host malicious content, a search for hosts files, and a further crawling of the exploit containing URLs discovered from these two groups [8]. Of the 16,190 URLs generated from URLs suspected to host malicious content in step one, 207 of these URLs, which equates to approximately 1%, were identified as containing exploits [8]. For the top 1,000,000 websites that were examined, based on their popularity rankings, 710 of these URLs were found to contain exploits. In step two, once recursive redirection analysis had taken place, the list of malicious URLs from the suspicious list had increased by 263% to 752 URLs [8]. For the popular site list, the number of exploit URLs increased to 1,036, or 46% [8]. In stage three, one of the virtual

machines successfully captured a zero-day exploit, the JView [8] profiler in javaprxy.dll, when instantiated within Internet Explorer contained a remote code execution vulnerability totally compromised the victim's system [9].

2.2.2 *SpyProxy*

SpyProxy is an extended web proxy system that protects users from malicious URLs. It consists of a system containing virtual machines to process requested URLs on-the-fly in a similar manner to previously discussed Strider HoneyMonkey. It is unique in that it functions within a proxy server and serves as a defense platform rather than just a measurement tool. SpyProxy ideally should keep clients safe from malicious content and it also should not reduce the usability of a browser, for example, by generating large delays between requests. SpyProxy downloads content on the requesting client's behalf and evaluates it to verify whether the URL is malicious or benign. During experimental trial runs of SpyProxy, the average delay from when a client requests a URL to when the client's browser begins to render was a surprising 600ms [10].

SpyProxy first performs a static analysis of the requested URL. If it is unable to identify or process an object, which would be the case for any non-HTML content types, it forwards the object to a virtual machine which visits the URL and checks for any system state changes such as newly created processes, modifications to the file system, registry modifications, or Operating System crashes [10].

Various optimizations have increased the performance of the SpyProxy system. One of the first is the caching of the post-security checking of the requested page which produces a hit rate that is competitive with a typical web cache. These hits are only

generated if all of the requested content is served from the proxy cache which eliminates the possibility of an unexpected outcome from as dynamically generated content [10].

Another optimization to the system is pre-fetching requested content for the client delaying execution until the SpyProxy system allows the page to be rendered. A further optimization technique is the periodic release of content to the client from the proxy by processing part of a page and then immediately sending it to the client if it is non-malicious thereby making the process more streamlined [10].

2.2.3 HoneyIM

Instant Messaging (IM) malware can spread very quickly making it a significant security risk for users. A variant of the Kelvir worm caused Reuters to disable its Instant Messaging service back in 2005. Two main methods of malware spreading through IM clients are URLs linking to malicious websites and file transferring. Once a machine has been compromised, the malware spreads by sending similar messages with malicious URLs or through file transfer to the users on the infected client's buddy list which spreads the malware at an exponential rate [11].

There are some protection schemes that enhance IM security such as using CAPTCHA to counter the spread of IM malware. The burden of such security can dissuade a user from using the service [11].

The HoneyIM system detects the spread of malware through "dummy buddies" on a users buddy list. This essentially eliminates false positives since fake buddies should never receive messages from a legitimate user. HoneyIM is based on the open-source IM client, Pidgin, and the client honeypot, Capture. In simulated executions of

HoneyIM with only 5% of total users comprised as fakes on the Instant Message network, HoneyIM was able to detect the spreading malware after only 0.4% of the users, on average, had been infected [11].

The HoneyIM system has four components: the communication module, the detection module, the suppression module, and the notification module. The communication module parses IM traffic sent to decoy buddies on the network. It relays messages to the detection module which determines whether a URL was included in a message or if a file transfer request was made. This module then notifies the suppression module which examines network traffic and filters out messages with malicious intent. The notification module alerts network administrators when spreading malware has been identified [11].

The communication module supports all of the functions a normal IM client supports and also supports all of the various IM protocols available. The detection module identifies clients as compromised if a URL or file transfer request is received. Encryption will not circumvent the detection module as the final message received within an IM conversation must be in plain-text. If a URL is received by the detection module, it can use HoneyMonkey to detect system anomalies post visit. Taint analysis could also determine if an executable can compromise the system and generate a signature of the file. The suppression module acts as a network filter by denying traffic generated from identified offensive clients. The notification module informs network administrators when malware has been detected [11].

2.2.4   Caffeine Monkey

Caffeine Monkey contains a JavaScript engine "based on extensions to the open source Spidermonkey JavaScript implementation" [12]. It uses a MySQL database to store retrieved documents, the results of analysis, as well as organizing crawls [12].

Although the Caffeine Monkey focuses on JavaScript, similar ideas can be applied to other scripting languages. Techniques to obfuscate the true functionality of scripts are numerous. One obfuscation technique is called whitespace randomization. This simple method omits whitespace which causes the scripts to appear different when traversing the internet but retains the exact same functionality to be performed. This technique does not hide what the scripts are actually doing during execution but rather changes the raw data which could defeat filters that match content [12]. Similarly, comment manipulation serves a very similar purpose as whitespace randomization. It leaves the code unmodified but changes the binary representation of the script which could potentially fool systems put in place to detect malicious activity. Comments could also be used to bewilder a human analyst examining the code by giving inaccurate or misleading guidance [12].

String obfuscation is another abashment-inducing technique which ranges from various encoding methods, to XOR functions, to Caesar Ciphers. These techniques render signature based detection impractical as a result of the myriad number of possible combinations in which strings can be represented and still induce the same result [12].

Variable name randomization and function pointer reassignment reassigns objects to a different variable or function with the intent of obfuscating the true functionality of the actual variables and functions utilized. Real-time security devices would not be able

to tell the difference between the built-in functions of JavaScript and a user-created function of the same name [12].

Integer obfuscation bypasses security mechanisms looking for memory addresses. For example, the address "'0x04000000' could be expressed as 16,777,216 * 42, or any number of other ways" [12].

Block randomization changes the structure of a script's statements and changes the code syntactically but performs the same actions. This is a more sophisticated technique that alters if/else and loop constructs. The combination of various obfuscation techniques can make detection potentially very tedious and taxing [12].

Heritrix, an Internet-scale crawler developed by the Internet Archive, is used to collect JavaScript. The Heritrix crawler "collected approximately 225,000 web documents over a continuous period of about three and a half days, with a total yield of 7.9GB" [12]. Of these, 364 documents which comprised 4.5MB (0.2%) of the total data collected were JavaScript files [12].

Once the JavaScript documents were ready for analysis, they were submitted to the Caffeine Monkey JS engine and the runtime logs were examined but provided no malicious results. Four malicious examples were obtained from security researchers of which SecureWorks made requests. These results were scaled to the results from the MySpace crawl. There was an apparent difference in the ratios of the various actions performed by the benign MySpace JavaScripts and the known malicious JavaScripts. Benign scripts made much more use of the document.write() method while malicious scripts made more use of string instantiation and objects. In the malicious scripts, DOM

(Document Object Model) elements were created with a higher frequency and the eval() function was used less than in the benign scripts [12].

The Caffeine Monkey JS Engine hooked functions that were determined to be the most likely to be obfuscated in JavaScripts and created logs at runtime. Thus, the flow of execution was seen in the logs without the need for script debugging [12].

2.4   Payload Delivery Methods

The following sections describe some of the different ways that malware can be distributed to victim machines.

2.4.2   Clickjacking

Clickjacking occurs when a user is persuaded to mouse click on an unapparent element of a page that has been placed there by the attacker but is not noticeable to the user [13]. This kind of attack could lead to the unintentional and undesired consequences of transfer of funds, interacting with fraudulent advertising, posting messages, or other actions that could be triggered by the click of a mouse [14].

The most common known form of clickjacking uses an invisible iframe overlaid on top of the user's desired content [13]. For example, JavaScript can be used to align framed content in real time with the user's mouse cursor which allows an attacker to have the victim perform actions that require multiple clicks [14].

One of the most notable things about clickjacking is that it is not based on an exploitable vulnerability or a bug in a web-based application. It is simply an abuse of features inherent within HTML and Cascading Style Sheets (CSS) [14].

An extension named ClickIDS, which as the name suggests is a clickjacking intrustion detection system, in the No-Script browser plug-in, which prevents scripted elements from executing, detects if an event is triggered by an obscured or invisible element after a user has made a click. Currently, an annoying side effect of this plug-in is a large number of false-positives are generated [14].

ClickIDS detects if clickjacking is taking place within a test environment by identifying coordinates on a page where clickable elements exist and then manipulating the mouse to click on each of these individual elements. If two or more elements are overlapping during the click event, a suspicious behavior alert is generated. The ClickIDS examines both frames and iframes during the detection process [14]. Since it only interacts with elements that are deemed clickable. The possibility of detecting clickjacking that is occurring elsewhere within web pages cannot be done [14].

2.4.3 Drive-By Download

A drive-by download occurs when malicious software is installed on a victim's machine without his or her knowledge or consent [15]. These types of downloads are successful due to leveraging some vulnerability, which in many cases occur in a web browser.

In 2004, the drive-by download attack known as the "Download.ject attack" successfully compromised a large number of well known business websites. These compromised sites downloaded such software as key loggers as well as Trojan Horses in order to steal user credentials and private information [16].

2.5 *Malware Delivery and Execution Prevention*

The following sections examine different methods to prevent the successful delivery of malicious payloads to a user's system. When considering how to defend against malware, a lot of questions come to mind. One is how can software automatically be identified as malicious. A second question is what can be done to prevent both the system and the user from acquiring and executing malware. These and other such questions are quite difficult and there tends to be only partial solutions rather than complete answers.

2.5.1 *Data Execution Prevention*

Data Execution Prevention (DEP) technology, implemented in both hardware and software, examines memory to prevent malware from executing. If running in Physical Address Extension mode, the hardware version of DEP sets all of the memory regions within a process to be non-executable unless there is code that is recognized as executable. This eliminates malicious code that attempts to execute from these locations by blocking these attempts and throwing an exception [17].

Hardware DEP implementations vary by architecture (AMD utilizes NX, or no-execute page-protection while Intel utilizes XD, or the Execute Disable bit) but serve the same purpose. Usually, DEP is used on a per-virtual-memory-page basis by modifying a bit within the PTE (page table entry) [17].

The software implementation of DEP functions without a direct dependence on hardware. This version of DEP is limited compared to the hardware version. It can prevent Structured Exception Handler (SEH) overwrites and is generally used on

computers without the hardware features previously described. Software-based DEP is applied at compile-time and typically limited to Windows system libraries [18].

The main benefit of DEP is the prevention of code execution from pages in memory classified as data, such as the default heap, stacks, and memory pools. DEP will raise an exception when code attempts to run from these defined-data sections and if the exception is unhandled, will terminate the program [17].

There are four configuration options for DEP: OptIn, OptOut, AlwaysOn, and AlwaysOff. The OptIn option means DEP is enabled for only those applications that are explicitly specified, which by default are the Windows system files. In the OptOut option, DEP is always on and is only off for those applications specified. The AlwaysOn option enables DEP for each individual application, while the AlwaysOff option will not use DEP for any application [17].

Although DEP mitigates the effectiveness of malware, it cannot be solely relied on to protect a system. In previous research, ways were found to bypass DEP. One was to run code from sections of memory that are designated executable to modify the flags of the non-executable memory. This was using ret2libc to call NtSetInformationProcess and the ProcessExecuteFlags which would disable the non-executable support for the calling process [18].

2.5.1 Nozzle

Heap-spraying is a technique that manipulates even type-safe languages to an attacker's advantage. It creates a large number of objects containing an exploit within a process's heap. One common method uses a web browser and JavaScript. A website

containing malicious JavaScript uses the interpreter to allocate and place malicious objects in the process's heap [19].

It is quite difficult for standard signature based detection methods to detect heap-spraying as it is easy to represent the same functionality in a variety of ways via techniques such as polymorphism and encoding. Instead of using a signature-based method, Nozzle takes a two-level approach to detect heap-spraying. It scans "objects locally while at the same time maintaining heap health metrics globally" [19].

Nozzle scans objects locally by performing a sandboxed interpretation of heap objects as if they were code and seeing if there are any signs of malicious intent. One of the most notable cues of malicious intent is the detection of a NOP sled. These sleds can contain an arbitrary set of commands as long as their execution does not result in termination or alter the payload [19].

The types of instructions Nozzle deemed invalid are I/O or system calls, interrupts, privileged instructions, and jumps external to the current object's range. Nozzle attempts to find objects that modify control flow assuming an attempt to arrange flow so a seemingly random jump will execute the malicious code [19].

2.5.2 Gatekeeper

Gatekeeper enforces reliability and security policies for JavaScript code via a subset of acceptable and safe JavaScript. This subset is based on a collection of 8,379 JavaScript widgets rather than analysis of the language itself [20].

The number of items Gatekeeper prohibits is relatively small. In particular, it bans the use of *eval*, *Function*, *setTimeout*, *setInterval*, and *with*. These constructs accept

string parameters and then executes said parameters. The first four items present a problem because strings can be interpreted as code but this is not known until run time which makes it impossible for a static analysis to detect. Specifically, the symbol lookup scope could be altered. The reflective constructs of *Function.call*, *Function.apply*, and the arguments array are allowed since they can be statically analyzed. One of the most prevalent risky features of JavaScript that needs to be addressed at runtime are innerHTML assignments as well as filed references that are unresolved [20].

During runtime, the safe JavaScript subset and another Gatekeeper subset are used. The difference between the two is the Gatekeeper subset allows non-static field stores as well as innerHTML assignments. The widget is checked against the safe JavaScript subset. If it fails this test, it is checked against the Gatekeeper subset; if this fails the program is deemed unsafe and will not be considered any further. If either the safe or the gatekeeper subset tests pass, the program undergos pointer analysis and is checked to see if any established policies are broken [20].

2.6 History of Twitter Vulnerabilities

Although the main focus of this research is on the effects URLs external to Twitter may have on a user browsing said links, the Twitter service itself has been vulnerable to malicious attacks. The following sections examine some of these vulnerabilities.

2.6.1 SMS Authentication Vulnerability

A vulnerability in the Short Message Service (SMS) authentication service, which allows a user to access their Twitter account via text message, was leveraged by spoofing

caller ID's. This is because the caller ID was the only thing used to validate a user. This vulnerability made it possible for anyone who could spoof a caller ID to post a message to an account if they possessed the caller ID information that was tied to the specified account [21].

2.6.2 Clickjacking Vulnerability

In February 2009, a vulnerability was discovered on Twitter that propagated a button labeled "Don't Click" on a Twitter user's page without the user manually posting the URL themselves by using an invisible iframe to perform a clickjacking attack. This vulnerability was deemed a "prank" and did not cause the compromise of any accounts [22].

2.6.3 XSS Worms

In April 2009, many "worms" were found spreading throughout the Twitter service via a Cross-Site Scripting (XSS) vulnerability within the Twitter Profile CSS. Like the previously mentioned clickjacking vulnerability, these XSS worms automatically posted statuses to a user's page [23].

2.6.4 MouseOver Vulnerability

In September 2010, a "MouseOver" vulnerability allowed a XSS attack to occur. The vulnerability triggered when a user would move their mouse over a hyperlink posted within a status. This executed custom CSS or JavaScript which leaves open the possibility for the unintended visiting of URLs hosting malicious content or other such malicious activities [24].

2.7 Twitter's Malware Countermeasures

2.7.1 Malicious URL Filtering

Utilizing Google's Safe Browsing API, Twitter enabled a URL prescreening service that looks for URLs known to host malicious content. If a user were to attempt to post a URL that has previously been identified as malicious, the filter would display a prompt to the user stating "Oops! Your tweet contained a URL to a known malware site!" [25]. Including this filtering process in the submission of URLs is a positive step forward, but the problem with this approach is that it only detects known malicious URLs rather than both known and unknown malicious URLs.

2.7.2 Additional Filtering after Bit.ly Partnership

In 2010, Twitter and URL shortening service Bit.ly partnered. As part of this partnership, Twitter announced a new URL filtering mechanism. In-depth details on the algorithms and monitoring services used for this filtering are not public [26].

2.8 Summary

This chapter contains relevant definitions for terms related to this thesis, a review of various methods of crawlers that scan for malware, some previously known vulnerabilities found within the Twitter service, as well as measures that Twitter has taken to reduce the potential malware that is present. Using this information, a methodology is created in Chapter 3 to create a system that collects and examines URLs from Twitter for malware.

III. Methodology

3.1 Problem Definition

With the release of the Department of Defense's (DoD) Directive-Type Memorandum on 25 February 2010, social networking sites have been deemed accessible for use within the various DoD organizations and on the Non-Classified Internet Protocol Router Network (NIPRNET) [27]. Since these social networking sites have the potential to facilitate the spread of malware, as is described in Chapter 2, determining how great the risk is of contracting malware on these sites and also to develop a system that can potentially prescreen said sites to prevent the compromise of government information systems would be very beneficial.

3.1.1 Goals

The main goal of this research is to develop a system that can automatically gather URLs from statuses, which are commonly referred to as tweets, posted on Twitter, and analyze those URLs to determine whether malware is present at the specified locations. This goal will provide a prototype system to ascertain the risk of being exposed to malware when visiting URLs found using social networking services, and in particular to this research, Twitter.

Another goal of this research is to create a Twitter Malware Repository (TMR) which can be further examined in subsequent studies to identify the different classes of malware present, the methods of delivery of the malware, the specifics of the assumedly various functionality of the malware, and so on.

3.1.2 Approach

Twitter statuses are collected via a Python script. A custom web crawler, also written in Python, is used to determine the safety of the collected URLs. A database tracks the gathered statuses, the results generated by the crawler, as well as other statistical information.

The status collecting script sends requests to Twitter, using Twitter's Streaming and Search Application Programming Interfaces (API) to retrieve current statuses based on the most popular trends at the time of collection. The URL examiner determines whether the gathered URLs are using any of the various URL shortening services that are publicly available. URL shortening services are used for statistical tracking, making the sharing of URLs easier by only requiring a relatively short URL (generally the service's URL followed by a few characters), and also to make the most of the 140 characters that statuses are limited to on Twitter. If the URL examiner determines that a status-gathered URL is in fact utilizing a shortening service, it will store the "unshortened" URL in the database for analysis as the full URL is what is actually being visited, besides which the shortened URL has the potential to be reused and reassigned pointing to a different location depending on the service [28]. The custom made crawler visits the URLs contained within the gathered statuses to determine whether the site contains malware. The results of this study will primarily include drive-by download as described in Chapter 2 as simulated user-interaction is not within the scope of this study. That is, malware that does not require manual intervention or input is captured.

3.2 System Boundaries

The System Under Test (SUT), which is named the Twitter Malware Collection System (TMCS), is composed of two server-grade rack mounts that provide global Internet connectivity to Twitter and the URLs investigated. Its data storage ability and virtualization hosting platform allow simultaneous execution and management of a series of Virtual Machines (VMs) on which the different components of TMCS execute (Figure 1).

One of the rack mounts is designated strictly for collection and storage purposes. A lone VM running on this hardware is responsible for a number of functions. One of the Python scripts running on this VM, named the Twitter Status Fetcher (TSF), creates the API requests through the use of the PycURL library, which fetches objects identified by a URL [29]. The results from the trends query using the Search API are URL encoded and used as a query parameter for the Streaming API which provides statuses based on the given trending parameters. Twitter's Streaming API responds to this query with a stream of JavaScript Object Notation (JSON) objects. These objects are parsed within the Python script which determines whether the URLs within the collected statuses contain shortened URLs, composes e-mail messages consisting of the systems current progress and stores the collected URLs as well as the results generated by the crawler components in a file-share and a MySQL database which are housed on the second rack mount.

The Component Under Test (CUT) within the SUT, is named the TMCS ESCAPE Handler (TEH) module. The TEH module has two main parts: a Python script and the ESCAPE system. The Python script controls much of the actions performed by the VM client and the ESCAPE system traps the execution of malicious code. The parts

of the TEH module are described in more detail within the system services section of this chapter.

Figure 1: Twitter Malware Collection System
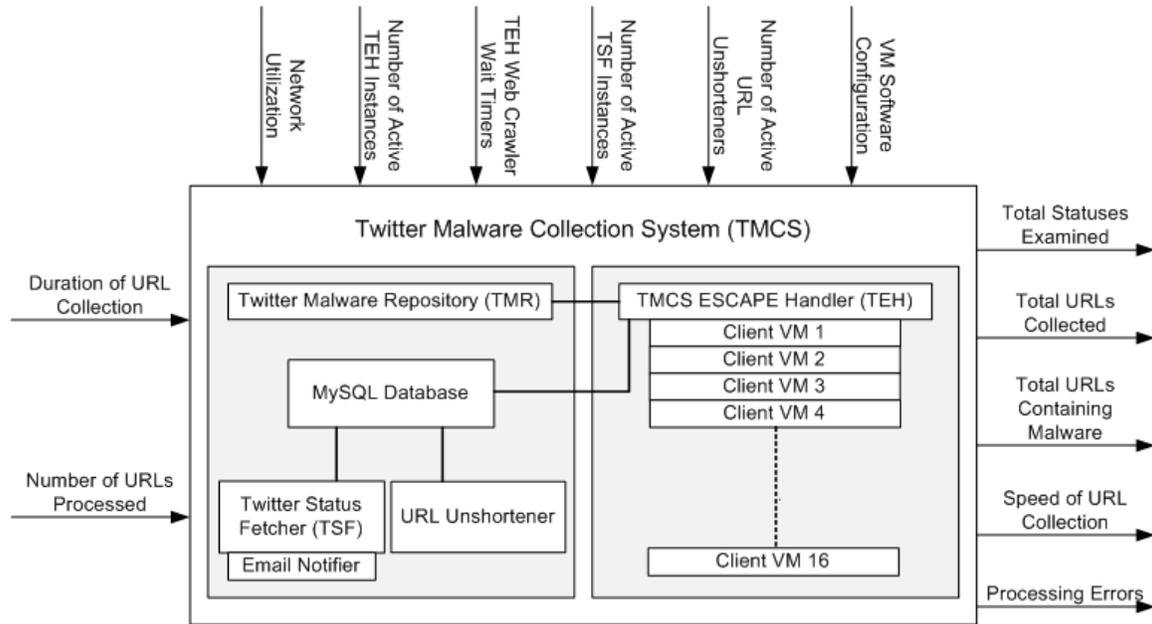
3.3 System Services

3.3.1 Status Collection

One of the services TMCS provides is the collection of statuses from Twitter. This is accomplished through the TSF module as previously described. The possible results of status collection are: 1.) successful status and 2.) formatting error. The formatting error results in the status being discarded due to malformation.

3.3.2 URL Extraction from Statuses

Once a status has been retrieved from Twitter, the TSF module parses the JSON data and extracts any URLs that are contained within. The outcomes of this service are: 1) URL(s) found and 2) No URLs are found.

3.3.3 Storage of Extracted URLs

Extracted URLs are stored within a MySQL database. In conjunction with these extracted URLs are: the Twitter ID associated with the status the URL was found in the URL itself, and the date the URL was collected. The outcomes of storing the extracted URLs include: 1) URL(s) successfully stored and 2) URL(s) already in database. When the second outcome occurs, a counter column, which is assigned to each URL, is incremented to keep track of the total times each URL is witnessed.

3.3.4 URL Unshortening

Since a large majority of URLs found in Twitter statuses use a URL shortening service, such as bit.ly, the URLs are "unshortened" to determine what location the shortened URLs point to. This process also allows additional analysis to be performed based on the URLs themselves. Initially, use of the various shortening services' APIs were considered, but since a myriad of these shortening services are in operation and are found within Twitter statuses, an alternative method was chosen. The URL Unshortener sends a Hypertext Transport Protocol (HTTP) request to the URL in question and if the response received contains an HTTP redirect status code, typically 301 or 302, it searches via a regular expression through that same response's header and stores the redirection destination URL. The possible outcomes of URL Unshortening are: 1) URL is

unshortened and stored and 2) URL is not unshortened. The first outcome occurs when the URL in question uses a URL shortening service or if an external redirection occurs (a redirection to a different domain). The second outcome occurs when a shortening service is not being used, the domain name cannot be resolved, or if a relative redirection occurs (a redirection using a relative path that is located within the local domain).

3.3.5   URL Processing through the TEH module

The TEH module is responsible for multiple tasks with regards to the processing of URLs gathered by the TSF module. The Python script portion of TEH performs the following actions:

- Requesting URLs from the TMCS database for processing

- Handling instances of Internet Explorer

    o   Launching Internet Explorer

    o   Visiting specified URLs for a specified time

    o   Closing Internet Explorer

- Scanning ESCAPE's output log for malware positive indicators

- Creating and storing dump files of Internet Explorer when malicious code has been detected

- Reverting the current VM to a clean snapshot and restarting through SSH

    o   When malware has been detected and collected

    o   When a preset timer has expired

The revert and restart timer value for the script of 30 minutes is chosen to prevent any potentially undetected malware from tainting results and also to clear out any other

unforeseen abnormalities that could be experienced while also minimizing the total overhead experienced when reverting and restarting a VM client.

The ESCAPE system portion of TEH actually determines whether or not a URL contains malicious content. The ESCAPE system is comprised of the following:

- A kernel-mode driver monitoring code execution

- A list of Hash-based Message Authentication Codes (HMACs)

- A configuration file

- An execution result log file

The list of HMACs is generated by the user on a trusted baseline system, that is, a system comprised of approved files for execution. These HMACs are signatures of the executable portions of the Windows programs allowed to execute. The HMACs are used by the kernel-mode driver to determine when suspected malware attempts to execute. The following flags produced by ESCAPE are stored in its log file are indicative of malware:

- When the Windows Error Reporting feature triggers (werfault.exe)

- When unsigned code that is not part of the signed executable attempts to execute

- When unsigned code that is part of the signed executable attempts to execute

- When a new process is created

The outcomes of the URL processing are: 1) No potential malware detected and 2) Potential malware detected.

3.4 Metrics

The metrics that assess the performance of TMCS are:

Total statuses examined: The total statuses examined determine how quickly TMCS is able to retrieve statuses to scan through from Twitter.

Total URLs collected: The total URLs collected determine the available pool of potential malware-hosting URLs to be scanned.

Total URLs determined to potentially contain malware: This metric is the one of most interest.

Speed of URL collection: This metric is largely reliant on throttling by Twitter.

Processing Errors: This metric measures the number of errors encountered while processing URLs.

3.5 Parameters

The following system parameters affect system performance including the metrics and system responses.

3.5.1 System Parameters. The system parameters are those parameters that when changed alter the system responses and/or the metrics.

3.5.1.1 Network Utilization. The amount of available network bandwidth consumed corresponds to the number of collected URLs and also the rate at which URLs are visited by the web crawlers. The available networking resources are changed with respect to other research being conducted due to a lack of an independent networking source. In addition, the availability of the social networking service, in this case Twitter, also affects the ability to obtain URLs, although this is not something that can be

28

controlled by TMCS. This parameter is closely linked to the number of active TEH, TSF, and URL Unshortener instances parameters.

    3.5.1.2   Number of Active TEH Instances.   Since this research is conducted with shared networking resources, the number of active web crawlers also determine how fast URLs are processed and results are gathered.

    3.5.1.3 TEH Web Crawler Wait Timers.   The time each VM waits on a particular visited URL determines the overall amount of time that must be allotted to process each URL collected. Choosing a length of time appropriate for each web crawler to pause at each URL is influenced by two factors. The first of is the time required to start up an instance of Internet Explorer. This value is manually calculated and 5 seconds was deemed adequate. The second factor is the time to load individual URLs. According to a report by Google, the average page consists of 320KB of data *[30]*. Given that there are an unknown number of other variables that could affect the wait time of the crawler, 10 seconds was chosen to allow data to download and any other processing, including execution delay timers, to occur when a URL is visited. Processing many include interpreting JavaScript or executing other outlets for malicious payload delivery. This, in total, sets the web crawler wait timer to be 15 seconds per URL visited. A longer timer could have been selected, but the length of time required to wait at each URL directly affects the total time required to process the entirety of the collected URLs.

    3.5.1.4 Number of TSF Instances Active.   As more instances of the TSF module are run, the total number of URLs scanned simultaneously increases. This number is capped based on the ability to obtain IP and account white listing privileges

from Twitter or the use of available IP addresses and accounts. For this research, a white listed status could not be obtained, and only one IP address and one Twitter account were used.

3.5.1.4 Number of URL Unshortener Instances Active. When more instances of the URL Unshortener module are running, the number of fully realized URLs increases at a greater rate.

3.5.1.5 VM Software Configuration. Since some malware may only affects a particular version of a web browser or a certain patch level of an Operating System, multiple configurations for the processing VMs are considered. In addition, different versions may benchmark differently than others in terms of speed which has the potential to impact the system in various ways.

3.5.2 Workload Parameters. The workload parameters, which consist of the characteristics of the service requests made to the system are listed below.

3.5.2.1 Duration of URL Collection. This is the total amount of time allotted for the purpose of gathering URLs. The more time available for collection results in more URLs gathered. The chosen length of time for URL collection from statuses was approximately 40 days. This time period allows a relatively large number of statuses to be processed while also allowing enough time for analysis once complete. This period also determines the total number of URLs that are collected and processed.

3.5.2.2 Number of URLs processed. As the total number of URLs increases, the number of URLs infected with malware also potentially increases.

3.6   Factors

3.6.1   Considered but excluded factors

Due to the fact that white listed privileges were not obtained for this research from Twitter, the number of TSF instances was not considered as a factor for testing. Therefore, the TSF was limited to one active instance as Twitter would only allow one streaming connection at a time. The number of active TEH instances was also considered as a potential factor, but to be courteous and fair to others using the shared networking resources, this value changed from time to time at the operator's discretion typically with fewer instances running during the day when utilization by others was apparent. Had this not been an issue, a set number would have been chosen to run throughout the entirety of the research process.

The factor chosen to test the performance of the system is:

3.6.2 VM Software Configurations.

Initially, three configurations were selected for the different VM Software Configurations. However, due to a configuration error, there was realistically only enough time to test two of the VM software configurations. The first configuration is comprised of Windows XP SP2, Adobe Reader 8.0.0, Internet Explorer 7.0.5730.13 Update Versions: 0, and Office 2007 version 1.2.0.4518.1014. The software versions were chosen because they contain known exploitable vulnerabilities. Windows XP is chosen as the Operating System for this configuration as it still has the largest share of Operating Systems in use, comprising approximately 40% of active Internet computers as identified by user-agents [31]. The second configuration chosen consists of Windows

Vista SP2, Adobe Reader X 10.0.1, Internet Explorer 9.0.8112.16421 Update Versions: RTM (KB982861), and Office 2007 version 12.0.6545.5000. Windows Vista SP2 was chosen as the operating system for this second configuration because at the present time it has been adopted alongside Windows XP for use throughout the Air Force. All versions of the software tested in the Vista configuration are fully patched and up-to-date at the start of testing (Table 1).

Table 1: Chosen Factor and Levels

VM Software Configuration 1	Windows XP SP2, Adobe Reader 8.0.0, Internet Explorer 7.0.5730.13 Update Version: 0, Office 2007 ver. 1.2.0.4518.1014
VM Software Configuration 2	Windows Vista SP2, Adobe Reader X 10.0.1, Internet Explorer 9.0.8112.16421 Update Versions: RTM (KB982861), and Office 2007 version 12.0.6545.5000

3.7 Evaluation Technique

The evaluation technique used for this research is direct measurement. Measurement is chosen for this research because models and simulations are not used, but rather a live, real-world social networking system is used.

The testing environment consists of two Dell PowerEdge R710 server-grade rack mounts. These rack mounts each have two quad-core Intel® Xeon® E5530 CPUs @ 2.40GHz, three 15k RPM hard disk drives giving a total usable storage capacity of 271 GB, and 32 GB or RAM. The VM running the TSF module on one of the rack mounts used Ubuntu 10.10, MySQL 5.1.41, and Python 2.6. The VMs running the TEH instances were described in the factors section. The VMs were managed and hosted using VMWare ESXi 4.1.0.

3.8 Workload

The workload for TMCS is the total number of URLs processed by the TEH module. This is ultimately determined by how long URLs are collected, which in this research is set at approximately five weeks.

3.9 Experimental Design

A full factorial design is chosen for this research. Since there is only one factor with two levels, two experiments are required. The expected variance in the responses of the system is unknown, and since such a large workload is used, only one replication is performed. This results in a total two experiments to be executed.

3.10 Methodology Summary

Through the use of Twitter's APIs, a system is developed to collect an extensive number of URLs, which are processed by a series of VMs to determine if malicious code is present at the given locations.

IV. Results

The metrics outlined in the previous chapter determine the performance of TMCS which consists of total statuses examined, total URLs collected, total URLs determined to contain malware, and processing errors.

4.1 Total Statuses Examined

After running for approximately 40 days and 16 hours, the final count of statuses examined by TMCS is 19,309,417. This is an average of 474,822 statuses examined per day, and 19,784 statuses examined per hour. While the total number of statuses examined may seem rather large, it in reality only represents a small fraction of the total number of statuses that are posted on Twitter. Based on statistics provided by Twitter, the total statuses examined by TMCS represents approximately 0.4% of the total number of statuses posted on Twitter during the time of collection [32]. Figure 2 below is a histogram showing the total URLs collected each day.
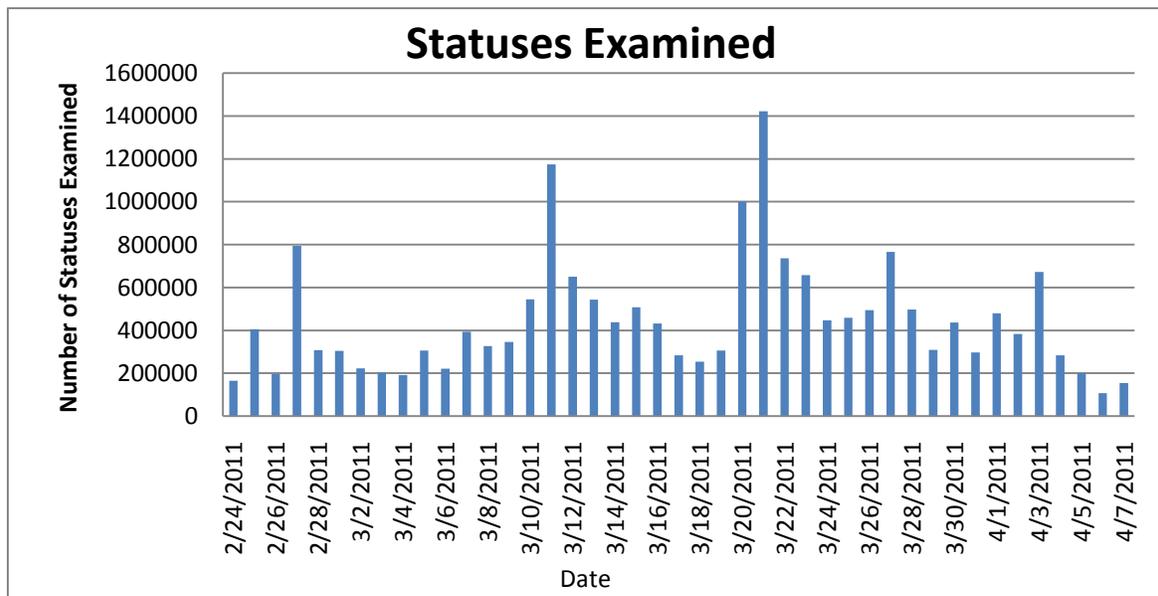

Figure 2: Total Statuses Examined

4.2 Total URLs Collected

The final total of unique URLs that were collected by TMCS is 466,237 URLs. The total number of URLs witnessed by TMCS, including duplicates, is 1,363,935. Therefore, only about 34.18% of the URLs gathered were unique. The total number of URLs witnessed came from approximately 1,315,077 separate statuses, indicating that multiple statuses included multiple links. This last figure is not an exact number as it was generated solely from the status update Emails, which were sent hourly, implying that some may have been missed during the final hour of collection since a total number of status containing URLs was not stored in any other fashion. Figure 3 below displays the URLs collected on a per-day basis.
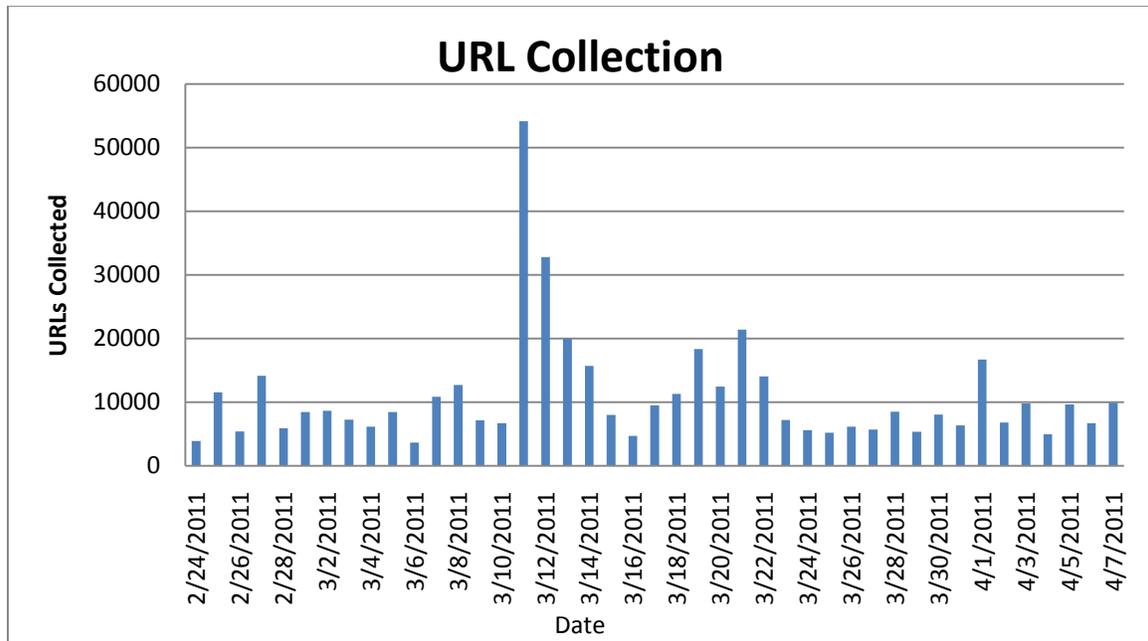

Figure 3: Total URLs Collected

35

4.3 Total URLs Determined To Contain Malware

The following sections display in detail the different types and also the totals for the URLs determined to potentially contain malware.

4.3.1 Windows XP Configuration Results

The number of URLs flagged as running suspected malicious code when visited by a Windows XP instance of the TEH module was 1,271. After examining the configuration for the Windows XP VMs, it was determined that the Windows Error Reporting feature was accidently disabled. This explains why there were zero results for this data set. Table 2 displays the different ways the ESCAPE system was triggered.

Table 2: Windows XP ESCAPE Results

Windows Error Reporting	0*
Unsigned code not from the executable	217
Unsigned code from the executable	878
New Process Creation	176
Total	**1,271**

Figure 4, below, shows the number of URLs identified as potentially malicious on a daily basis. An interesting point is the noticeable spike of URLs detected for the date of March 3, 2011. This was the day in which Tōhoku earthquake and tsunami struck Japan [33].
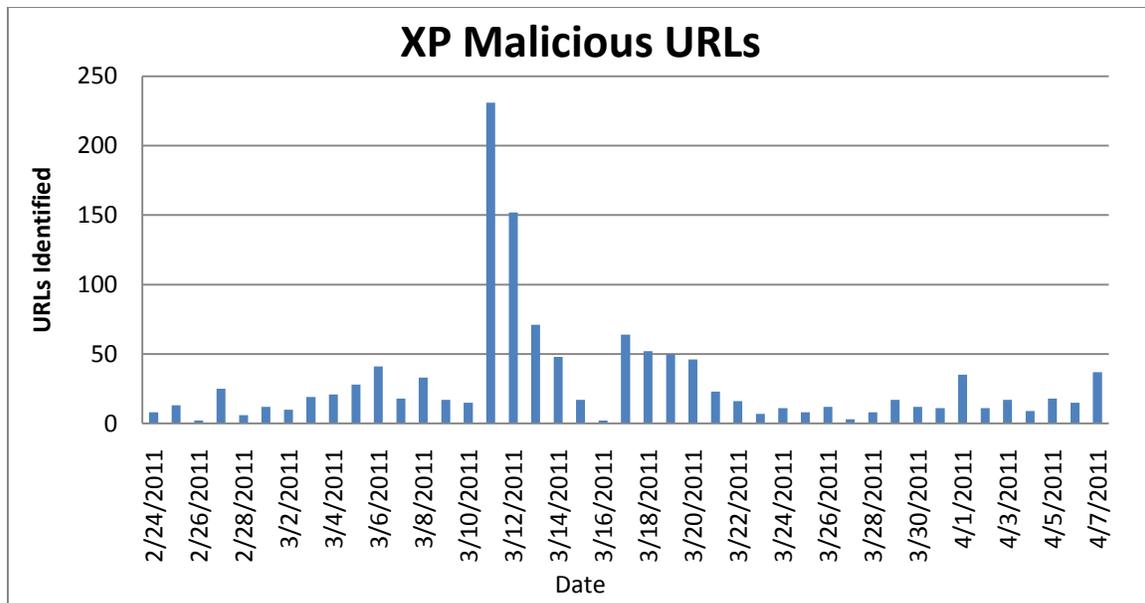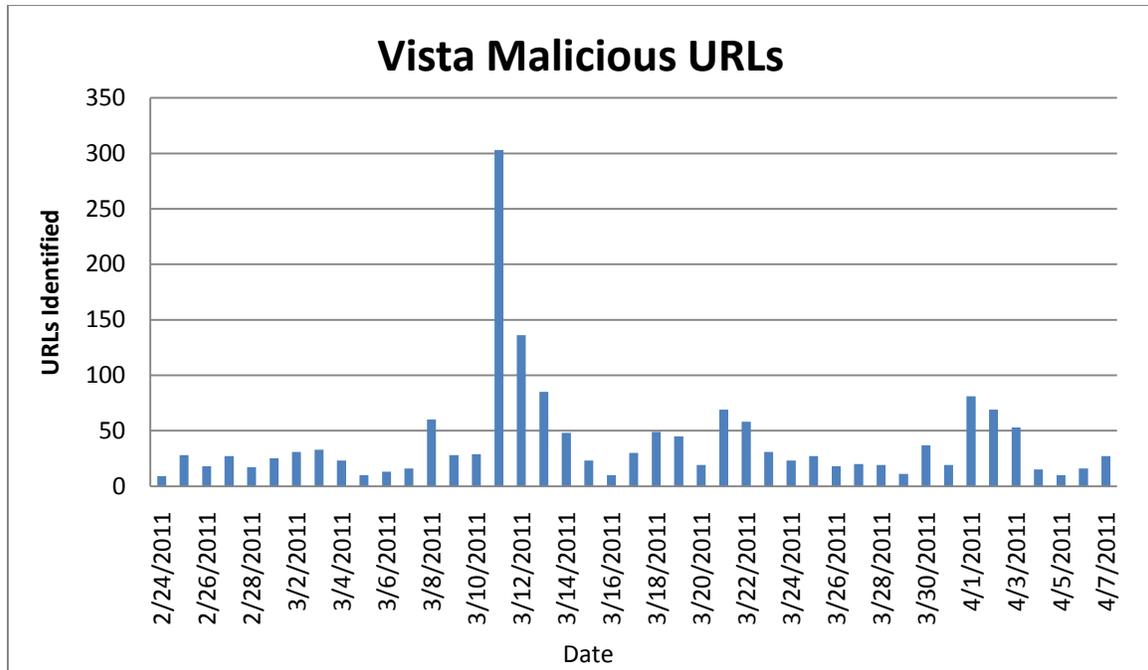
Figure 4: Potentially Malicious XP URLs

4.3.2 Windows Vista Configuration Results.

The number of URLs flagged as containing suspected malicious code when visited by a Windows Vista instance of the TEH module came to 1,718.   Table 3 displays how the ESCAPE system was triggered.

Table 3: Windows Vista ESCAPE Results

Windows Error Reporting	633
Unsigned code not from the executable	745
Unsigned code from the executable	3
New Process Creation	337
Total	**1,718**

Figure 5 below shows the number of URLs identified as potentially malicious by date collected.

37

Figure 5: Potentially Malicous Vista URLs

A similar pattern is shown when comparing the results of the Windows Vista trial run to the Windows XP trial run. The noticeable spike at March 3, 2011 is again present.

4.4 Examining Vista and XP Results

A t-test is conducted to assess whether the number of malicious URLs from both the Windows XP and Windows Vista runs were statistically different from each other. The resulting p-value of 0.2829 suggest that the two data sets are not different. Figure 6 displays a box plot of the two data sets.
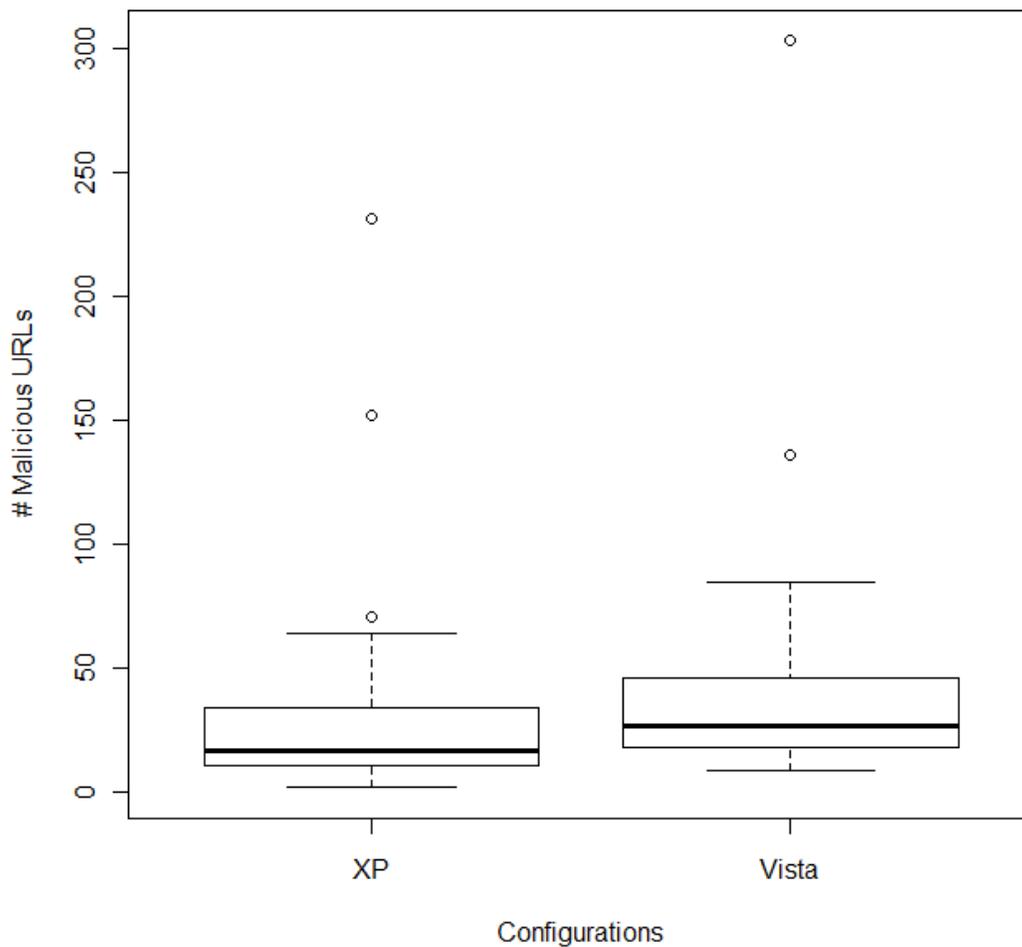
Figure 6: XP vs. Vista Box Plot

The ratios of malicious URLs to total URLs collected were examined to see if any day had a significantly higher percentage of malicious URLs than any other for both runs. A visible spike is seen on March 6, 2011 for Windows XP and April 2, 2011 for Windows Vista. Figure 7 shows the ratio of malicious to total URLs for the Windows XP configuration.
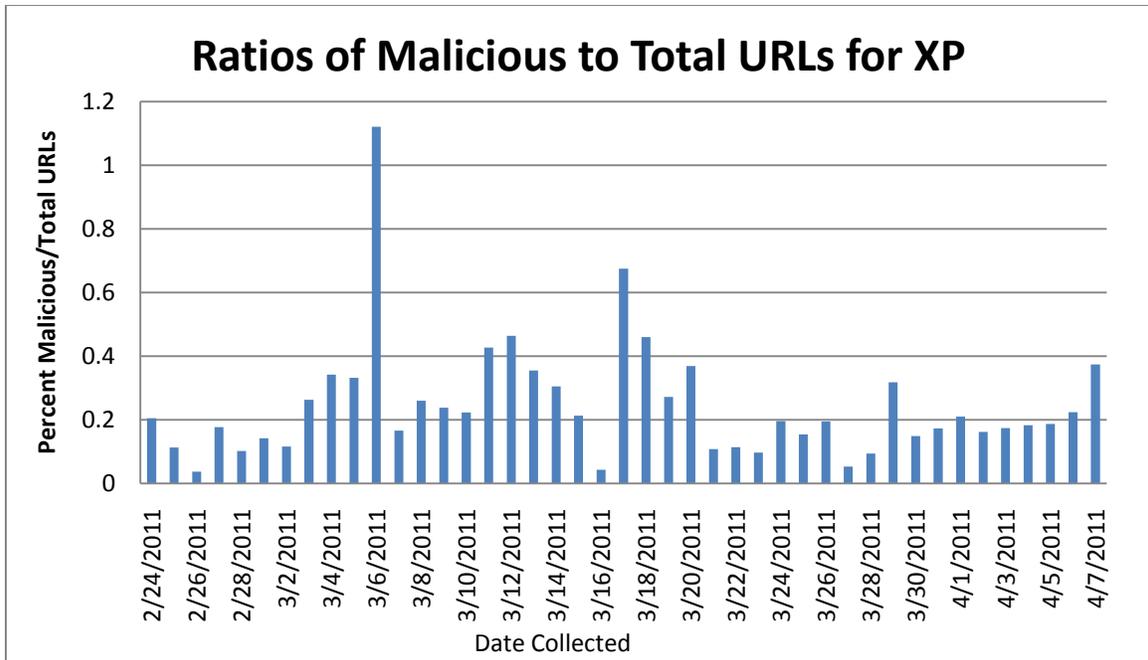
Figure 7: XP Ratios

Figure 8 is the ratio of malicious to total URLs for the Windows Vista configuration.
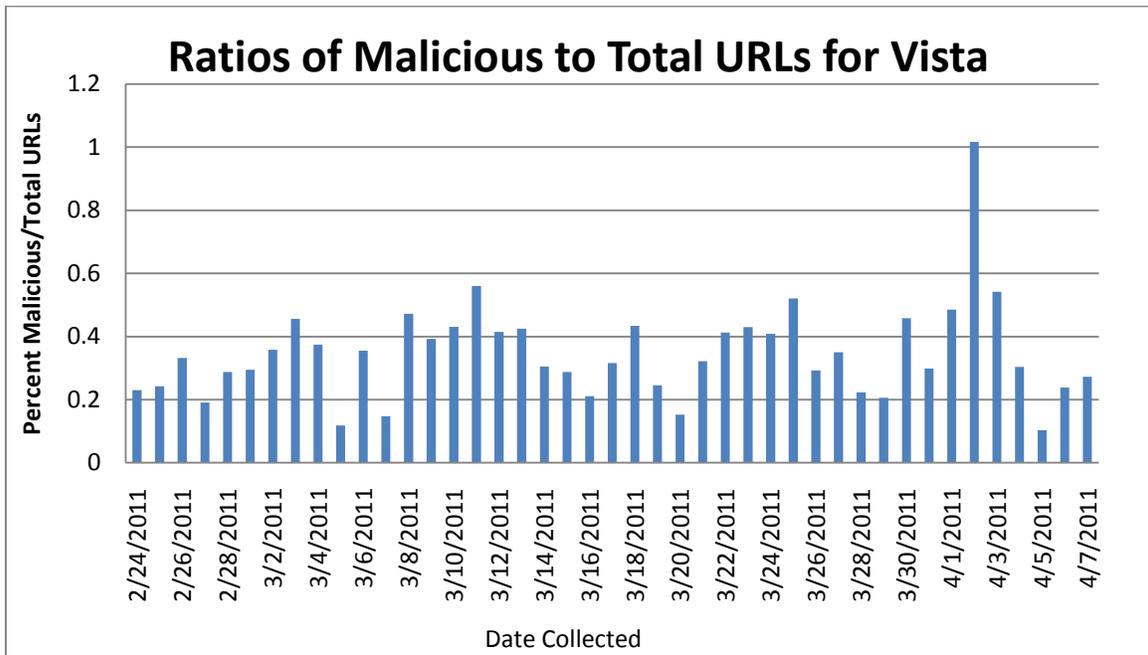

Figure 8: Vista Ratios

Visually, there are noticeable spikes on March 6, 2011 for the XP configuration and on April 2, 2011 for the Vista configuration. A t-test performed on the above ratios of malicious to total URLs results in a p-value of 0.007898. This p-value suggests that these two sets of data are in fact statistically different.

The URLs collected on the days with the noticeable ratio spikes were examined to see if there are any similarities between the two groups. This examination was inconclusive.

4.5 Processing Errors

The total number of tracked errors during this research is 20,376. These errors are comprised of HTTP connection failures during the URL unshortening process, unshortening parsing failures when a redirection URL could not be found due to relative redirects and Twitter feed errors when the TSF module encountered malformed JSON data sent from Twitter. The HTTP connection failures comprise a large majority of the total errors experienced. These connection failures were caused by such things as DNS timeouts. Table 4 contains the totals of each distinct type of error experienced through the research process.

Table 4: Processing Errors

HTTP connection failure	18,844
Unshortening parsing failure	833
Twitter feed error	699
Total	**20,376**

4.6 Manually Identified Malware

During the initial run of TMCS, a configuration error caused the result from the processed URLs to be ignored. During this run though, one of the processing VMs was found to be infected with malware. The malware that infected this specific VM has been categorized as the "Cycbot Trojan" [34]. It became apparent that the VM may have been compromised when an error window appeared stating that a VBScript did not have the correct permissions to execute. There was also a command prompt visible which was not manually launched. The third visible identifier was that the Windows Help window was opened. Figure 9 portrays the above scenario.
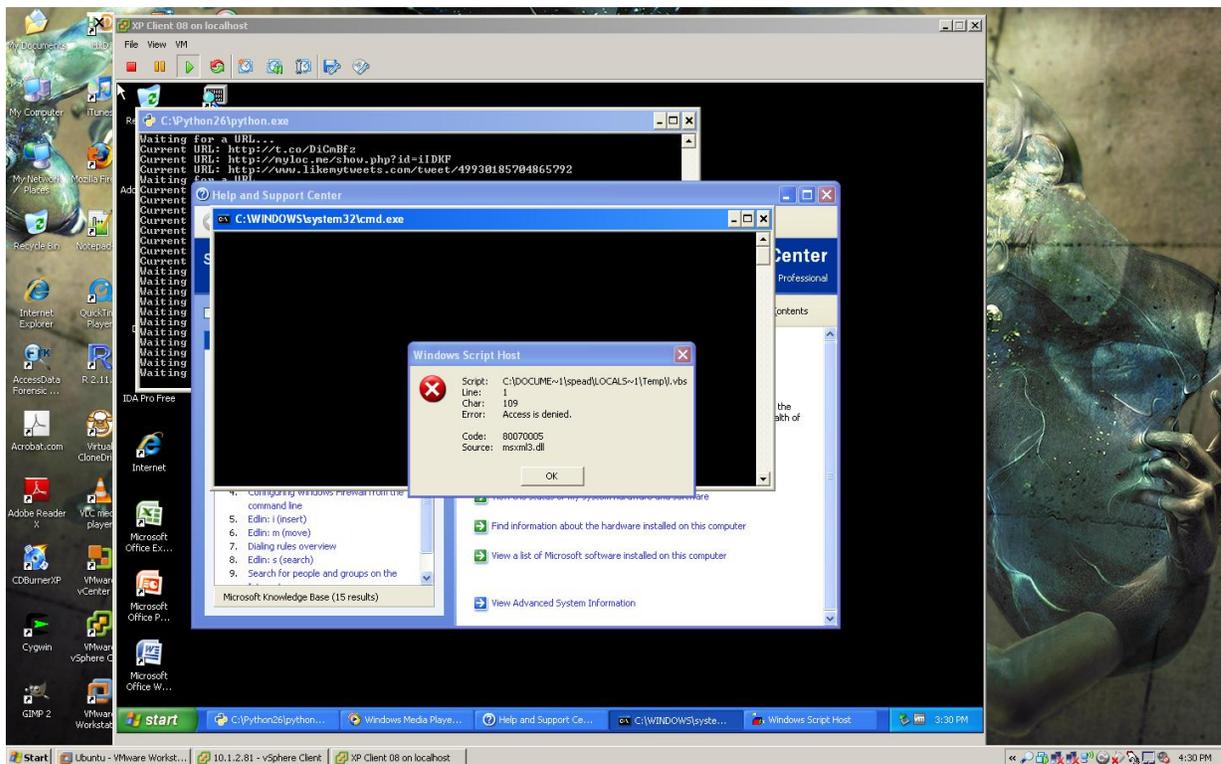

Figure 9: Cycbot Trojan

The distinguishing characteristics of this Trojan were identified while examining the VM. This malware attempted to make NetBIOS connections to the domain "xibudific.cn". Figure 10 below shows a packet capture of this described action.
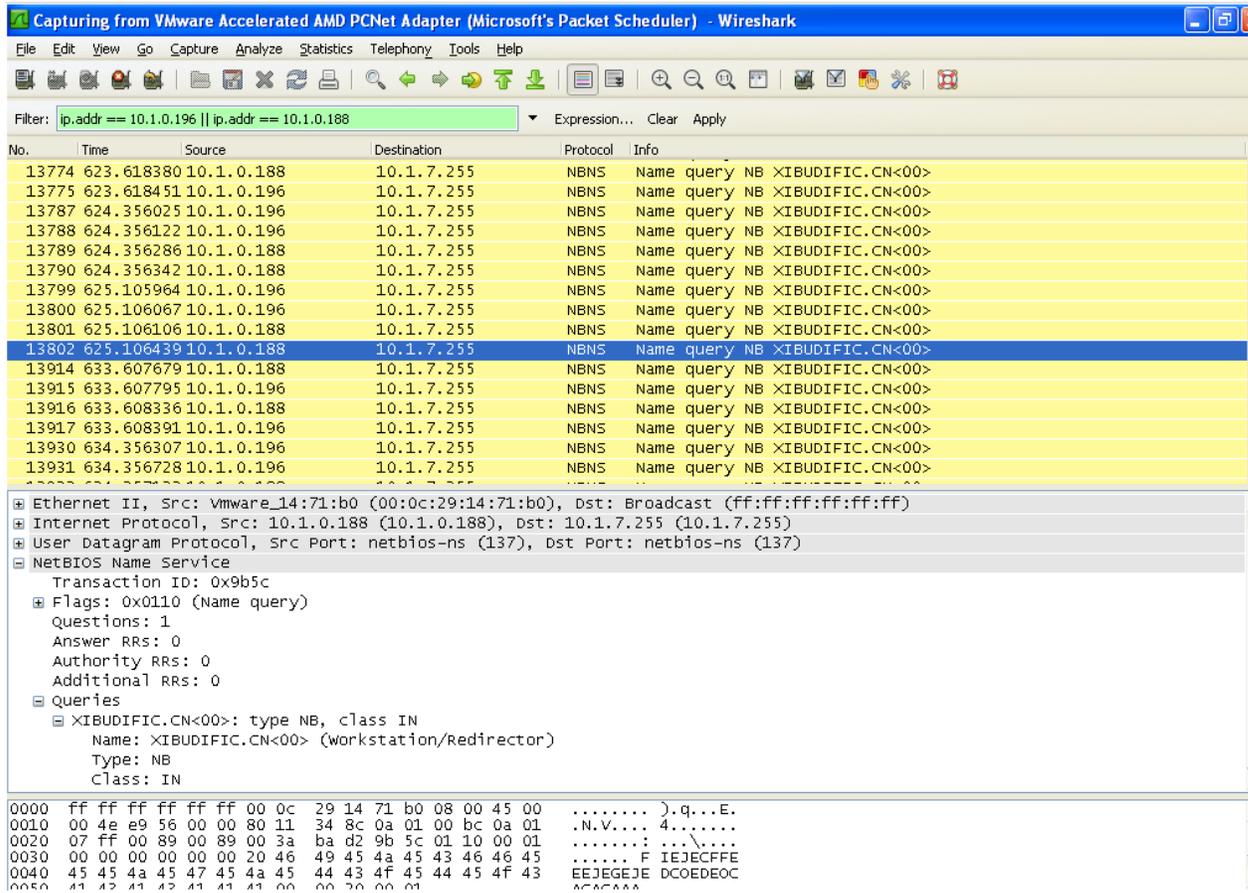
Figure 10: Cycbot Packet Capture

An executable associated with this malware was also found running and consuming a large amount of system resources. Figure 11 shows the "Cycbot" associated "conhost.exe" process running in Windows Task Manager. This finding provides substantial evidence, even without examining the collected dump files in detail, that URLs containing malware are able to bypass the Twitter filtering as discussed previously in Chapter 2.

Windows Task Manager

File Options View Shut Down Help

Applications | Processes | Performance | Networking | Users

Image Name	PID	User Name	CPU	Mem Usage	Page Faults	PF Delta
conhost.exe	1948	spead	00	543,252 K	1,413,559	0
alg.exe	1944	LOCAL SERVICE	00	2,876 K	744	0
VMwareUser.exe	1928	spead	00	5,688 K	3,795	0
VMwareTray.exe	1900	spead	00	4,384 K	1,135	0
wireshark.exe	1692	spead	00	62,504 K	140,222	17
svchost.exe	1464	LOCAL SERVICE	00	4,868 K	17,361	0
taskmgr.exe	1456	spead	00	1,936 K	1,483	0
svchost.exe	1392	NETWORK SERVICE	00	3,924 K	7,563	0
svchost.exe	1340	SYSTEM	00	23,140 K	517,106	0
VMUpgradeHelper...	1328	SYSTEM	00	3,664 K	1,933	0
explorer.exe	1232	spead	00	34,976 K	277,476	1
svchost.exe	1096	NETWORK SERVICE	00	5,016 K	3,132	0
svchost.exe	1016	SYSTEM	00	4,628 K	4,325	0
vmacthlp.exe	1000	SYSTEM	00	2,212 K	693	0
lsass.exe	840	SYSTEM	00	1,496 K	1,365,507	0
services.exe	828	SYSTEM	00	4,312 K	1,653	0
winlogon.exe	784	SYSTEM	00	3,680 K	6,118	0
csrss.exe	760	SYSTEM	00	1,296 K	772,707	0
smss.exe	688	SYSTEM	00	388 K	217	0
vmtoolsd.exe	652	SYSTEM	00	7,732 K	222,206	0
wordpad.exe	648	spead	00	1,632 K	1,757	0
dumpcap.exe	588	spead	00	4,304 K	1,122	0
cmd.exe	576	spead	00	164 K	1,868	0
helpctr.exe	408	spead	00	22,396 K	6,844	0
helpsvc.exe	348	SYSTEM	00	27,168 K	112,925	0
setup_wm.exe	328	spead	00	4,136 K	1,407	0
HelpHost.exe	240	spead	00	8,624 K	2,364	0
wscript.exe	212	spead	00	8,672 K	2,317	0
spoolsv.exe	148	SYSTEM	00	5,344 K	1,569	0
System	4	SYSTEM	00	236 K	3,843	0
System Idle Process	0	SYSTEM	99	28 K	0	0

☐ Show processes from all users   [End Process]

Processes: 31   CPU Usage: 0%   Commit Charge: 789M / 2460M

Figure 11: Cycbot Trojan's "conhost.exe"

4.7 Results Summary

The experimental results show TMCS provides the services of URL collection, URL examination, and potential malware storage as well as keeping track of statistical values and errors experienced. The methodology used and the analysis performed support the findings.

44

V. Conclusions

5.1 Accomplishments

The TMCS system successfully obtained and analyzed URLs contained within Twitter statuses. Using Twitter's APIs, a MySQL database, Python scripts, and multiple VMs running on VMWare ESXi, statuses are gathered, and URLs are extracted, stored, and examined. Dump files are successfully created for the URLs determined to be malicious by the ESCAPE system creating a repository of malware. These dump files can be further examined to determine the root of malicious activity. After processing a total of 466,237 URLs, TMCS found that 2,989 of them were deemed malicious between the two examination runs using the Windows XP and Windows Vista configurations.

5.2 Contributions

The TMCS system provides a means for researchers to automatically collect instances of malware found on the social networking service Twitter. This collection can be used to develop signatures, examine potentially novel methodology contained within the malware, and be used as testing data for new malware protection schemes. TMCS can also provide data on how prevalent URLs hosting suspected malicious content are on Twitter as shown in Chapter 4.

5.3 Future Work

5.3.1 Integration of User Input

By integrating user input, whether performed by actual people or simulated by some form of automated functionality, the TMCS system could capture more instances of

malicious activity. This additional input would trigger events that are not required by drive-by downloads, which were the sole focus of this research. Malware that may require user interaction for successful delivery and execution of a malicious payload, such as fake anti-virus rogueware products, could be witnessed and analyzed with this additional integration.

5.3.2 Prescreening Application

Extending the functionality for more practical purposes is not outside of the realm of possibilities for TMCS. It could be altered to examine links contained within Twitter statuses on the fly as a user requests them. This sort of functionality could be implemented in a proxy-like fashion, or as a host-based protection scheme.

5.3.3 Expanding Social Network Compatibility

As the focus of TMCS is on social networks, specifically Twitter, adding the ability to analyze URLs posted on various other social networking platforms, such as Facebook or newer Google+, would be beneficial. Since Twitter is not the only social networking service available, and since it only makes up a fraction of the total traffic generated by users for the purposes of social networking, this expansion would encompass a potentially larger number of URLs to process.

5.3.4 Incorporating Additional Detection Methods

The only system used within this research for detecting potentially malicious activity was the ESCAPE system. While this system generated a fair amount of results, it would be interesting from a research standpoint to compare and contrast the results that are generated by other methods of detection.

5.3.5 Automate Analysis of Captures

The dump files generated when a URL was deemed as hosting potentially malicious content are not examined in-depth because the manual analysis of these dump files requires a considerable amount of time. Implementing or creating a methodology to reliably examine these dump files automatically would significantly reduce the diagnosis process time.

5.3.6 Processing URLs in Parallel within a Single VM

The TMCS system is able to process multiple URLs at a time through the use of multiple VMs. With some refinement of the ESCAPE log parsing portion of TMCS, it could be possible to have multiple URLs checked at the same time within a single VM. This type of enhancement could significantly increase the speed at which URLs are processed.

5.3.7 Focus on Semantically Relevant Information

When collecting statuses to extract URLs, TMCS would request statuses that refer to the most popular trending topics. These topics were disregarded after using them as search parameters. Storing and examining these topics may be able to provide more insight into the distribution of malware on Twitter.

5.5 Conclusion

The TMCS system is a successful URL collection and examination platform. It is able to gather and process URLs from the social networking service Twitter, and store dump files of detected malware. The possibilities for future additions and applications of TMCS are promising. Malware research stands much to gain by using such a system for

47

the purposes of malware collection and examination. By applying this methodology,

social networking platforms can become a safer venue for communication.

Bibliography

[1] Graham Cluley. (2011, June) Simon Pegg is Twitter-hacked, warns fans of Trojan horse threat. [Online]. http://nakedsecurity.sophos.com/2011/06/26/simon-pegg-itwitter-hacked/

[2] Robert Moir. (2003, October) Microsoft TechNet. [Online]. http://technet.microsoft.com/en-us/library/dd632948.aspx

[3] SecureList. History of malicious programs. [Online]. http://www.securelist.com/en/threats/detect?chapter=77

[4] Rishabh Dangwal. (2010, January) Simplest Virus - Fork Bomb. [Online]. http://viralpatel.net/blogs/2010/01/simplest-virus-fork-bomb.html

[5] Nicolas Falliere, Liam O Murchu, and Eric Chien, "W32.Stuxnet Dossier," Symantec, 2011.

[6] Carnegie Mellon University. (1999, February) CERT (R) Advisory CA-1999-02 Trojan Horses. [Online]. http://www.cert.org/advisories/CA-1999-02.html

[7] Sean-Paul Correll and Luis Corrons, "The Business of Rogueware: Analysis of the New Style of Online Fraud," pp. 3-7, 2010.

[8] Yi-Min Wang, Doug Beck, Xuxian Jiang, and Roussi Roussev, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites that Exploit Browser Vulnerabilities," January 20, 2006. [Online]. http://research.microsoft.com/en-us/um/redmond/projects/strider/honeymonkey/

[9] Microsoft. (2005, July) Microsoft Security Bulletin MS05-037. [Online]. http://www.microsoft.com/technet/security/bulletin/ms05-037.mspx

[10] Alexander Moshchuk, Tanya Bragin, and Damien Deville, "SpyProxy: Execution-based Detection of MaliciousWeb Content," pp. 2-13, 2007.

[11] Mengjun Xie, Zhenyu Wu, and Haining Wang, "HoneyIM: Fast Detection and Suppression of," pp. 2-8, 2007.

[12] Ben Feinstein and Daniel Peck, "Caffiene Monkey: Automated Collection, Detection and Analysis of Malicious JavaScript," pp. 1-12, 2007.

[13] OWASP. Clickjacking. [Online]. https://www.owasp.org/index.php/Clickjacking

[14] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel, "A solution for the automated detection of clickjacking attacks," *ASIACCS '10 Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp. 2-

6, 2010.

[15] Microsoft. Security Intelligence Report. [Online].
http://www.microsoft.com/security/sir/guide/default.aspx#!section_7_1

[16] Mark A. Barwinski, Cynthia E. Irvine, and Tim E. Levin, "Empirical Study Of Drive-By-Download Spyware," Naval Postgraduate School, Monterey, California, USA, 2006.

[17] Microsoft. Microsoft TechNet. [Online]. http://technet.microsoft.com/en-us/library/cc738483%28WS.10%29.aspx

[18] Skape and Skywing, "Bypassing Windows Hardware-enforced Data Execution Prevention," *nologin*, pp. 2-8, 2005.

[19] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn, "NOZZLE: a defense against heap-spraying code injection attacks," 2009.

[20] Salvatore Guarnieri and Benjamin Livshits, "GATEKEEPER: mostly static enforcement of security and reliability policies for javascript code," *SSYM'09 Proceedings of the 18th conference on USENIX security symposium*, pp. 151-168, 2009.

[21] Scott Gilbertson. (2007, April) Twitter Vulnerability: Spoof Caller ID To Take Over Any Account. [Online].
http://www.webmonkey.com/2007/04/twitter_vulnerability_spoof_caller_id_to_take_over_any_account/

[22] Daniel Sandler. (2009, February) Twitter: Don't Click (Explained). [Online].
http://dsandler.org/outgoing/dontclick_orig.html

[23] Lynne Pope. (2009, April) Mikeyy Twitter XSS Mutates & Continues to Attack. [Online].
http://lynnepope.net/mikeyy-twitter-xss-mutates-continues-to-attack

[24] Mike Butcher. (2010, September) Warning: Mouseover tweets security flaw is wreaking havoc on Twitter [Updated]. [Online]. http://eu.techcrunch.com/2010/09/21/warning-mouseover-tweets-security-flaw-is-wreaking-havoc-on-twitter/

[25] Lucian Constantin. (2009, August) Malicious URL Filtering on Twitter. [Online].
http://news.softpedia.com/news/Malicious-URL-Filtering-on-Twitter-118308.shtml

[26] Brian Prince. (2010, March) Twitter Fights Phishing, Malware with Link Scanning Service. [Online]. http://www.eweek.com/c/a/Security/Twitter-Fights-Phishing-Malware-With-Link-Scanning-Service-556715/

[27] Deputy Secretary Of Defense, "Directive-Type Memorandum (DTM) 09-026 - Responsible and Effective Use of Internet-based Capabilities," 2010.

[28] bitly. Does bitly ever re-use links? [Online]. http://bit.ly/pages/help#i_1_4

[29] PycURL. (2010, January) PycURL. [Online]. http://pycurl.sourceforge.net/

[30] Barry Schwartz. (2010) Google Web Report: Average Page Size 320 KB. [Online]. http://searchengineland.com/google-web-report-average-page-size-320-kb-46316

[31] Wikipedia. Usage Share of Operating Systems. [Online]. http://en.wikipedia.org/wiki/Usage_share_of_operating_systems

[32] Twitter. (2011, March) #numbers. [Online]. http://blog.twitter.com/2011/03/numbers.html

[33] Wikipedia. (2011) 2011 Tōhoku earthquake and tsunami. [Online]. http://en.wikipedia.org/wiki/2011_T%C5%8Dhoku_earthquake_and_tsunami

[34] ThreatExpert. (2010, December) ThreatExpert Report: Backdoor.Cycbot, Backdoor.Cycbot!gen2, Backdoor.Win32.Gbot.bs. [Online]. http://www.threatexpert.com/report.aspx?md5=178a0b1875a1007349793b29f2e69580

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 15-09-2011	2. REPORT TYPE Master's Thesis	3. DATES COVERED *(From - To)* Sep 2009 - Sep 2011

4. TITLE AND SUBTITLE

Twitter Malware Collection System: An Automated URL Extraction and Examination Platform

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Kuhar, Benjamin B, Mr

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCO/ENG/11-07

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research develops the Twitter Malware Collection System (TMCS). This system gathers Uniform Resource Locators (URLs) posted on Twitter and scans them to determine if any are hosting malware. This scanning process is performed by a cluster of Virtual Machines (VMs) running a specified software configuration and the execution prevention system known as ESCAPE which detects malicious code. When a URL is detected by a TMCS VM instance to be hosting malware, a dump of the web browser used is created to determine what kind of malicious activity has taken place and also how this activity was allowed.

After collecting over a period of 40 days, and processing a total of 466,237 URLs twice in two different configurations, one consisting of a vulnerable Windows XP SP2 setup and the other consisting of a fully patched and updated Windows Vista setup, a total of 2,989 dumps were created by TMCS based on the results generated by ESCAPE.

15. SUBJECT TERMS

malware; malicious URLs; Twitter; social networking; web crawling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Rusty O. Baldwin, ENG
U	U	U	UU	63	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565 x4445 Rusty.Baldwin@afit.edu