Ransomware Creation

legacv dedbit3 byork-bork

legacv@proton.me - -

ABSTRACT

The final project within the Systems Programming course involved the creation of a program that utilized an external application programming interface (API). This project intends to utilize the creation of ransomware as a means of exploring the Windows API and its capabilities, and two separate programs were written to accomplish the task – one for process injection and one for file encryption. The project was mostly successful in terms of both programs' functionality; however, there is not yet a way to undo the file encryption of a user's system. In addition, encryption of the entire filesystem has not been tested due to time and resource constraints.

C CONCEPTS USED

External API implementation; Arrays; Strings; For-loops; While-loops; Programmer-defined functions; Shellcode

1. INTRODUCTION

C programming has been the focus of the Systems Programming course at Purdue University, specifically programming within the Linux environment. As a final project for the course, students were tasked with developing a program that incorporates an external API in addition to several C concepts that were used throughout the semester. The project parameters were open-ended, and due to the background and interests of the authors, a piece of ransomware was chosen as the focus in order to better understand the mindset of bad actors and their tools. The ransomware was chosen to be Windows- rather than Linux-based to add to the complexity of the project as well as provide an adequate set of APIs and API documentation to pull from.

2. BACKGROUND

Ransomware is a form of malware that infects the user's system, encrypting all files on it. The attacker will demand a ransom from the user in order to restore their system and obtain the key to decrypt their files.[1]

2.1 Related Works

Several other pieces of malware and ransomware were referenced in the creation of this project. The malware repository and information source VX Underground proved useful in both learning fundamental malware operation cycles as well as referencing concrete pieces of code.[2] In particular, the HelloKitty ransomware source code (first deployed in 2020 and leaked in 2023)[3] was used as a reference for spidering methods, blacklisted files and folders necessary for system operation, and file encryption. Several online resources and creators, such as the YouTube personality Crow, were referenced in early stages for base malware concepts, the handling of threads, and usage of the Win32 API.

3. USE OF EXTERNAL API

Two similar but distinct APIs were used for creating the first stage of the ransomware – Win32 and Windows Native. A third API, OpenSSL, was used for the secondary encryption portion.

3.1 Win32 API

The Win32 API (Application Programming Interface) is an integral part of the Windows operating system, providing a variety of functions and data structures that enable developers to create Windows applications This includes window management, user input control, file management, and system communication. At its core, the Win32 API acts as a bridge between an application and the operating system, facilitating communication and enabling developers to access system resources and perform various tasks. Developers use the Win32 API by calling functions that have specific parameters to request services from the operating system. These function calls can manipulate windows, handle user input, manage files and directories, communicate with hardware devices, and perform many other tasks required to run complex Windows applications. This is the preferred API Microsoft urges developers to use and its functions are standard across all Microsoft Windows versions and builds. It is also extensively documented on Microsoft's website.[4]

[1] *What is ransomware?* 2024. Microsoft.

[2] VX Underground. 2024.

[3] Abrams, L. 2023. *HelloKitty ransomware source code leaked on hacking forum*. BleepingComputer.

[4] *Get started with desktop Windows apps that use the Win32 API* 2021. Microsoft.

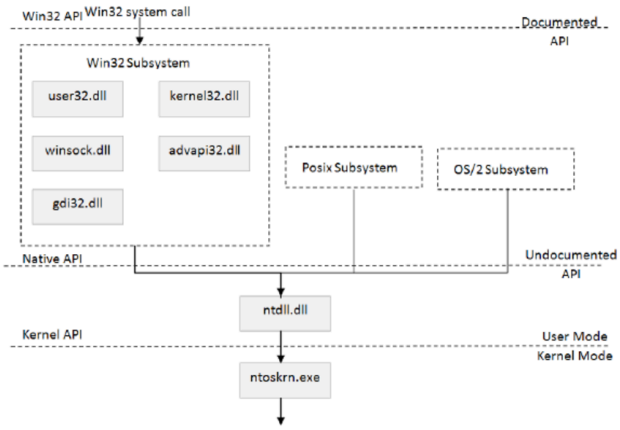

Figure 1. Windows 32 Architecture

3.2 Windows Native API

The Native API, also known as the NT API or NTDLL, is a low-level interface provided by the Windows NT kernel for system-level programming. It acts as the main interface for user-mode applications that communicate directly with the kernel and other system components. Unlike the Win32 API, which focuses more on user-level application development, the Native API provides deeper control over system features and performance. Developers can use the Native API for application and thread management, memory allocation and conversion, I/O performance, and system configuration. Although the Native API provides powerful capabilities, it is more complicated and less portable than the Win32 API, as it is tightly coupled to the Windows NT kernel architecture Developers working with native APIs often need an understanding deep in Windows and kernel-mode configuration options. Every call to the win32 API will eventually pass through the NT API before being passed to the kernel for execution. The NT API is undocumented and Microsoft does not want developers using it to build applications. Its function names change depending on the Windows release and build version, making it hard to work with and needing a lot of research to be able to accomplish the developers intended tasks. However the benefit of using the Windows Native API in malware development is its lower level capabilities and due to that the ability to more easily evade antivirus solutions.[5]

3.3 OpenSSL API

OpenSSL is a free, open-source cryptography project built to provide a toolkit for a wide array of encryption needs.[6] It also provides an API for C and C++ developers to be able to encrypt and decrypt data with a wide array of modes. For this project, the simplicity of AES 128-bit CBC mode encryption was preferred. Two headers in particular were necessary from the OpenSSL library; rand.h and evp.h. Rand.h is a header that allows a programmer to use the pseudorandom generation built into OpenSSL for the purpose of creating encryption keys. Evp.h is what essentially allows all other functionality: key and IV handling as well as encryption. The main function at work from the OpenSSL API is EVP_EncryptInit_ex(), which initializes encryption, using EVP_aes_256_cbc() as a parameter in order to specify the desired encryption mode. In future implementations, using the evp.h header to implement decryption functionality would be desired. However, in initial tests, decryption of the encrypted files was not successful.

[5] *Inside Native Applications*. 2021. Microsoft.

[6] *Welcome to OpenSSL*. 2024. OpenSSL.org.

4. APPROACH AND METHODS

As mentioned briefly in the introduction, two separate programs composed the project to simulate how ransomware would function in a real-life attack – one program utilized process injection and the second program utilized file encryption. In a real-life scenario, an initial malicious download would inject itself into a running process, then contact a command-and-control (C2) server in order to download the second stage of the payload (in this case, filesystem spidering and encryption) and execute it.

4.1 Process Injection

Process injection is an advanced technique used in software development and security exploits to inject code at the address of a running process. Process injection techniques usually involve modifying the memory of the target process and inserting custom code or payload. The most common methods of process injection are DLL injection, where a dynamic link library (DLL) is placed in the address space of the target process, and code injection, in which machine code is inserted directly into process memory. Furthermore, protection against process injection attacks requires strong security measures such as code signing, process monitoring, and access control to prevent unauthorized code and preserve system integrity.

For the project, a process injection was performed using several functions that were a part of the NT API. These functions had to be defined in a separate header file to lay the groundwork for each specific functionality; for example, the NtCreateThreadEx function was defined within the header file so that it could be used in the main program to create a thread that allows the shellcode to be injected into the process of choice. Some additional functions were pulled from the Win32 API for the processes that could not be accomplished using the NT API.

4.2 File Encryption

File encryption is a method of converting readable data into an unreadable form, called ciphertext, to protect against unauthorized access. Cryptographic algorithms and keys can be used to encode the contents of this file, making it incomprehensible without the associated decryption key. File encryption is widely used for purposes such as protecting sensitive information, ensuring confidentiality of data when transmitted or stored, and compliance with privacy laws. Common encryption methods include symmetric encryption, which uses the same key for encryption

and decryption, and asymmetric encryption, which uses two keys: the public key for encryption and the private key for decryption. Encryption algorithms such as the Advanced Encryption Standard (AES), Rivest-Shamir-Adleman (RSA), and Elliptic Curve Cryptography (ECC) do a great job of protecting data. In the case of malware file encryption can be used to encrypt sensitive and important data on a system and ask for a ransom in exchange for the decryption keys, forcing the victim to either comply or lose their data forever. As previously mentioned, AES 128-bit CBC mode encryption was used for this project due to its symmetric nature and resistance to being "cracked" (or having its key correctly guessed) even by modern technology.

5. RESULTS

The process injection, or first phase, had several problems during the testing, namely problems with access being denied when attempting to run the shellcode. However, once the process was streamlined, the code was able to run by injecting shellcode into the MS Paint process. This shellcode successfully allowed for the listener device to connect to the host computer, where commands that were run from the listener executed on the host machine. There were also hurdles regarding Windows Defender blocking execution of the first stage of the program upon detection of shellcode, though this was eventually remedied.

The code to be run from the listener was the encryption program, which was successful in encrypting all important files on the host's system. The most glaring problem with this was that the encryption could not be undone once the files were encrypted. This would be fine if the intention was to create actual malware that would work to destroy the user's system in an irreversible manner, however for ease and safety of testing, it was preferable to allow for the encryption to be undone. For this reason, total filesystem encryption was not tested; though, it is theorized that if the base path within the code was changed from a testing directory to the root of the filesystem, it would have proceeded successfully.

6. CONCLUSIONS

Both programs ended up being fairly successful by the end of the project. Of course, neither of them were perfect in how they executed or their functionality; however, they both were able to do the main tasks that they were intended to perform. A lot of knowledge and experience was also gained through using the Win32 API and the NT API, as all of the authors now have a better understanding of the processes and elements that go into each component of the Windows operating system.

6.1 Future Work

The first course of action regarding additional work with the code would be to implement a way for the user to "pay off" the ransom in some way so that the file encryption can be undone if the user complies. This would also make testing of the code much more manageable without the need of a virtual machine. It is likely that a separate program would need to be built for this such that it could be "distributed" to the victims by the authors of the ransomware, rather than baked into the encryption program for easy reverse-engineering. Some more work could also be done to further simplify the code to make it easier to hide the code within another process (if the group wanted to have the code begin executing from within another program the user runs).

Though total filesystem encryption was not tested, it is hypothesized that due to the size of a typical Windows filesystem, it would have taken several minutes to complete once run. This leads to more possible future work in creating multiple threads or processes in the second stage of the payload so that the encryption works through the filesystem faster, decreasing its chances of being halted in its tracks.

7. ACKNOWLEDGEMENTS

The authors would like to thank the professor and TAs for guiding their studies throughout the semester. Casey would like to thank the authors of the HelloKitty ransomware, as well as the fine people behind VX Underground, for providing insights into the world of black-hat hacking.

7.1 Individual Contributions

legacv – Designed and wrote the filesystem spidering and encryption code
byork-bork – Wrote test programs for testing the APIs and the process injection code
dedbit3 – Designed and wrote the process injection code

8. REFERENCES

[1] *What is ransomware?* 2024. Microsoft. https://www.microsoft.com/en-us/security/business/security-101/what-is-ransomware#:~:text=What%20is%20ransomware%3F%201%20Ransomware%20defined%20Ransomware%20is,a%20ransomware%20attack%20...%204%20Ransomware%20protection%20

[2] VX Underground. 2024. https://vx-underground.org/

[3] Abrams, L. 2023. *HelloKitty ransomware source code leaked on hacking forum*. BleepingComputer. https://www.bleepingcomputer.com/news/security/hellokitty-ransomware-source-code-leaked-on-hacking-forum/

[4] *Get started with desktop Windows apps that use the Win32 API*. 2021. Microsoft. https://learn.microsoft.com/en-us/windows/win32/desktop-programming

[5] *Inside Native Applications*. 2021. Microsoft. https://learn.microsoft.com/en-us/sysinternals/resources/inside-native-applications

[6] *Welcome to OpenSSL*. 2024. OpenSSL.org. https://www.openssl.org/