Adversary Emulation Framework

Genesis Resto, Kiara Rivera, Carlos Roque

*Abstract*—

# Introduction

The Adversary Emulation Framework aims to address the constantly evolving nature of malware attacks by developing a framework that offers a structured and strategic approach to managing complex networked systems and offers various options to expose malware evasion techniques. The framework employs dynamic evasion, in-memory execution, and encrypted payloads to enhance its effectiveness and resilience. The implant uses various evasion tactics to avoid detection and collect victim information for a secure database. The future work includes creating a special tool called an implant that covers evasion techniques, collecting system information from the victim, and performing debugging to uncover new attack vectors for malware evasion. This framework will provide defense systems with better insight into the workings of harmful programs and empower them to counteract them more effectively.

## Code Execution Methods

Currently, Command and Control frameworks perform their code execution using Injection Techniques. This is largely since implants can be boiled down to shellcode instructions to be inserted into memory. Although this opens paths for initial access into computer networks, the operator needs a very deep knowledge of the internals of operating systems and assembly instructions to create shellcode. Moreover, the advantage of this approach is that the implant code resides inside the computer memory. Hence evading on-disk detection mechanisms.

## Evasion Techniques

As of 2023, the most prevalent evasion techniques that exists is Reflective DLL Injection. Using the Shellcode Reflective DLL Injection (sRDI) project [15], actors can convert compiled DLLs into position-independent shellcode for injection or loading mechanisms. A combination of this, coupled with obfuscation, shellcode encryption and detection services’ patching, the implant can pass through defenses and achieve execution.

## New Developments

In our framework, we take a different approach to loading code into memory. First, we use download cradles to download scripts from the internet. This download cradle is for the Microcontroller to get the implant code, coupled with Dynamic Language Runtime [14] we can load the C# implant which is encrypted, into memory and execute it. Now, this approach can also be expanded into compiled bytes of .NET Assemblies, and this lets us use any Dynamic Programming Language that can accept the .NET Framework for compilation or inline execution such as plain-text IronPython code.

## Evasion Vectors

Now, since we’ll also be implementing evasion just before the actual .NET code is loaded, we need to consider each detection mechanism step by step. Our approach was to encrypt the plain-text C# implant using AES-256 to avoid AMSI from scanning it and rendering it malicious. Next, patch ETW by setting up the correct bytes and writing them into the buffer of the loaded process where ETW exists and writing the bytes to immediately return the EtwEventWrite function. Next, we also patch AMSI by sending the correct bytes to AmsiScanBuffer just as we did with ETW and return a clean scan before the actual scan. When decrypting the implant code and loading into memory using a C# runspace, we check if there are commands on the server, if not (dormant state) we encrypted some of the heap sections in the implant buffer to evade memory scanners. When the server has a command for an implant, decrypt that heap buffer and execute the command. To which encrypts it again to lay dormant.

# Techniques and components

A Command-and-Control Framework that include the following techniques:

* Dynamic evasion is a defense technique where a system or network is capable of adapting and evading attacks in real-time, allowing for a faster and more effective response to constantly evolving threats.
* Syscall variables is a technique used to avoid detection and exploitation of vulnerabilities in operating systems by dynamically altering the available system calls in a program or system.
* Event Tracing for Windows (ETW) and AMSI Patching are two security mechanisms built into Windows operating systems used to monitor and detect malicious activities. ETW is an event logging tool that allows developers and system administrators to capture and analyze detailed information about system activity in real-time. On the other hand, AMSI is an interface that allows anti-malware applications to scan the content of scripts, macros, and other files that can be used to carry out malicious attacks.
* In-memory execution is another technique used by attackers to evade detection by traditional security systems, by executing malicious code directly in the system's memory.
* Download cradles are a cybersecurity technique used by attackers to evade detection by traditional security solutions and download malware onto a compromised system. This technique involves the use of malicious scripts or commands to download malware onto the system via legitimate servers or compromised websites.
* A microcontroller is a device that combines a microprocessor, memory, and input/output peripherals on a single chip.

# Diagram Description automatically generatedDesign

## One of the limitations of this project in the real world could be to find effective evasion techniques for all operating systems at the same time. Most of the evasion techniques are for computers running Windows operating systems. The vast majority of the techniques that are known today are assumed not to apply to Linux and this could present a limitation to this project. It is also complicated by the fact that software versions are always being updated and this allows that without realizing it we use obsolete software and may not work in the future. It is also important to keep in mind that the Arduino must be a protected one, because if it is confiscated it could create serious problems.

## Performance Criteria

Detection Rate: How accurately the framework can identify and expose malware evasion techniques used by attackers.

Evasion Coverage: The range of evasion techniques covered by the framework, such as dynamic evasion, in-memory execution, and encrypted payloads.

Resilience: The ability of the framework to maintain effectiveness even in the face of advanced defense mechanisms employed by target systems.

Data Collection: The framework's success in securely collecting and storing victim information, including system information and network traffic data.

Attack Vector Discovery: The framework's ability to uncover new attack vectors and evasion techniques through debugging and analysis.

Performance Overhead: The impact of the framework on the performance of target systems, striving for minimal resource utilization and system slowdowns.

Integration and Compatibility: The framework's compatibility with existing security infrastructure and tools, ensuring smooth deployment and interoperability.

Effectiveness of Countermeasures: The framework's ability to provide actionable insights and recommendations for enhancing defense systems against malware.

Scalability: The framework's capability to handle large-scale networks and systems for emulation and analysis purposes.

Continuous Improvement: The framework's future work, including regular updates to address emerging malware techniques and security challenges.

# Prototype and simulation

## The system should be built to create a special tool called an implant. If the operator already has this tool, the system will take it in and manage it. If the operator doesn't have one, the system will make one, manage it, and get it ready to run. Then, it sets up a way for the tool and the server to talk to each other. If it is the first time the implant registers, it should collect system information from the victim. If it is not the first time, the implant should wait for the server to send commands before proceeding with any actions. If the server has tasks for the implant, it gets directions from a part called the Module Handler, using a TaskID. The implant does the task and sends the results back to the server. This can be done as many times as the operator wants. If the server doesn't have any tasks, the implant just waits for new instructions.

##### V. References

[1] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, “A Survey on Malware Detection Using Data Mining Techniques,” ACM Computing Surveys, vol. 50, no. 3, pp. 1–40, Jun. 2017, doi: <https://doi.org/10.1145/3073559>.

‌[2]R. Tahir, “A Study on Malware and Malware Detection Techniques,” International Journal of Education and Management Engineering, vol. 8, no. 2, pp. 20–30, Mar. 2018, doi: <https://doi.org/10.5815/ijeme.2018.02.03>.

[3] D. Patten, “The Evolution to Fileless Malware 1 The Evolution to Fileless Malware.” Accessed: Mar. 23, 2022. [Online]. Available: <http://infosecwriters.com/Papers/DPatten_Fileless.pdf>

[4] E. Chaffey, D. Sgandurra, and R. Holloway, “Malware vs Anti-Malware Battle -Gotta Evade 'em All!,” 2020. Accessed: Apr. 29, 2023. [Online]. Available: <https://core.ac.uk/download/pdf/340199834.pdf>

‌[5]A. Kapellas, “A Thesis in Malware Development.” Accessed: Mar. 29, 2022. [Online]. Available: <https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/11190/Kapellas_mte1604.pdf?sequence=2&isAllowed=y>

[6] Georgios Ioannou, “Malware Development for Red Teaming Using Metasploit Petros Katritzidakis,” 2018. Accessed: Apr. 29, 2023. [Online]. Available: <https://repository.ihu.edu.gr/xmlui/bitstream/handle/11544/29159/Dissertation_Thesis_Petros_Katritzidakis_Malware%20Development%20for%20Red%20Teaming%20Using%20Metasploit.pdf?sequence=1>

[7] X. Hu, “Large-Scale Malware Analysis, Detection, and Signature Generation.” Accessed: Apr. 30, 2023. [Online]. Available: <https://deepblue.lib.umich.edu/bitstream/handle/2027.42/89760/huxin_1.pdf%3Fsequence%3D1%26isAllowed%3Dy>

[8] D. Miller, R. Alford, A. Applebaum, H. Foster, C. Little, and B. Strom, “Automated Adversary Emulation: A Case for Planning and Acting with Unknowns.” Accessed: Apr. 30, 2023. [Online]. Available: <https://apps.dtic.mil/sti/pdfs/AD1108001.pdf>

[9] A. B. Ajmal, M. A. Shah, C. Maple, M. N. Asghar, and S. U. Islam, “Offensive Security: Towards Proactive Threat Hunting via Adversary Emulation,” IEEE Access, vol. 9, pp. 126023–126033, 2021, doi: <https://doi.org/10.1109/access.2021.3104260>.

[10] B. Strom, A. Applebaum, D. Miller, K. Nickels, A. Pennington, and C. Thomas, “MITRE ATT&CK®: Design and Philosophy.” Available: <https://www.mitre.org/sites/default/files/2021-11/prs-19-01075-28-mitre-attack-design-and-philosophy.pdf>

‌[11]M. Carvalho et al., “Command and Control Requirements for Moving-Target Defense,” IEEE Intelligent Systems, vol. 27, no. 3, pp. 79–85, May 2012, doi: <https://doi.org/10.1109/mis.2012.45>.

[12] B. Strom, A. Applebaum, D. Miller, K. Nickels, A. Pennington, and C. Thomas, “MITRE ATT&CK®: Design and Philosophy.” Available: <https://www.mitre.org/sites/default/files/2021-11/prs-19-01075-28-mitre-attack-design-and-philosophy.pdf>

[13] TylerMSFT, “System Calls,” learn.microsoft.com. <https://learn.microsoft.com/en-us/cpp/c-runtime-library/system-calls?view=msvc-170> (accessed Apr. 30, 2023).

[14] adegeo, “Dynamic Language Runtime Overview - .NET Framework,” learn.microsoft.com. <https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/dynamic-language-runtime-overview>

(Accessed Apr. 30, 2023).

‌