ISSN: 2834-4928

Research Article

Journal of Electrical Electronics Engineering

Beyond Traditional Testing: VMs and Abstraction in Correlation-Based IDS

Hung Anh Vu*

Electrical and Computer Engineering, University of Maryland, US

***Corresponding Author**
Hung Anh Vu, Electrical and Computer Engineering, University of Maryland, US

Submitted: 2023, Dec 04; **Accepted:** 2024, Jan 02; **Published:** 2024, Jan 31

Citation: Anh Vu, H. (2024). Beyond Traditional Testing: VMs and Abstraction in Correlation-Based IDS. *J Electrical Electron Eng, 3*(1), 01-06.

Abstract

A key innovation, the C2 abstraction layer, was developed to facilitate a comprehensive testing environment that produces a myriad of attack scenarios. Current methodologies employ comprehensive malware analysis using machine learning and deep learning techniques [1,6]. However, this project aims to develop a comprehensive testing environment that allows for the generation of diverse malware attacks. As of now, I have created an automated environment where simulated attacker and victim machines interact in real time, serving as a realistic backdrop to assess the proposed IDS. Accompanying this is meticulous documentation on malware operations and the abstraction layer's code. The current implementation can be found at https://github.com/HungAnhVu/C2abstraction.

Keywords: Cybersecurity

1. Introduction

With the current computing power, real-time file scanning by endpoint security tools has become very advanced. This capability mitigates the threat of attackers introducing malicious malware through file dropping for code execution. Yet, a significant vulnerability persists in the form of process injection attacks. These attacks deploy malicious 1code into the memory of trusted programs (such as Adobe), allowing the attacker to blend in with benign activities. Such stealthy tactics often fly under the radar of existing security mechanisms because they exploit the trusted status of legitimate applications and produce minimal events in the attack chain. This threat necessitated a refined approach to detection and countermeasures that rely on correlation-based analysis of each active processing thread which will differentiate uncommon behaviors.

My project aims to rigorously test and enhance Intrusion Detection Systems (IDS) through a systematic approach that uses real-world malware frameworks to simulate genuine attack scenarios. As each malware is executed, the objective is to conceptualize a mechanism for its automated run in controlled settings, such as virtual machines, tailored for IDS testing. By abstracting the complexities of various Command and Control (C2) handlers, I can present a unified interface for testing. This is particularly beneficial when dealing with a diverse set of C2 systems. Instead of crafting unique tests or setups for each, I have one interface that can handle multiple C2 environments. Additionally, as new C2 handlers emerge or as existing ones evolve, abstraction removes the need to redesign the testing setup from the ground up. Other benefits of abstraction in programming are discussed in [7].

The paper is organized as follows. The setup of my experiments are in section 3, where I outline the setup and the core functionalities of the C2Configuration class within the testing environment. The new algorithm is in section 4, which details the mathematical formalization and operational procedures of the C2Client. Experimental results are in??, demonstrating the effectiveness of the system in simulating attack scenarios and the performance of the IDS. The conclusions follow in section 6, where we summarize the findings and implications of this research.

2. Background

Command and Control (C2) systems are the cornerstone of cyber operations, providing a structure for attackers to maintain communication with compromised systems and coordinate their actions. The effectiveness of these systems lies in their ability to evade detection and provide persistent access to a target network. I would also like to acknowledge the Electrical and Computer Engineering department at UMD for providing me with this opportunity.

Framework 2.1 Metasploit Framework
1: **Input**: Target system information, vulnerability database
2: **Output**: Exploitation success probability, shell access
3: **Initialization**: Load MSF console, import target data
4: **Exploit Selection**: Match target system with potential exploits
5: **Payload Crafting**: Generate suitable payloads for the exploit
6: **Execution**: Deliver exploit and payload to the target
7: **Post-Exploitation**: Establish persistent access, extract data
8: **Reporting**: Log activities, generate reports

Framework 2.2 Covenant Framework
1: **Input**: Network access, post-exploitation objectives
2: **Output**: Command execution results, data exfiltration
3: **Initialization**: Deploy Covenant server, configure listeners
4: **Grunt Deployment**: Create and insert Grunt into the target network
5: **Communication**: Secure channel establishment with the Grunt
6: **Tasking**: Send tasks to Grunt, execute commands
7: **Data Handling**: Receive, process, and store exfiltrated data
8: **Cleanup**: Remove traces, terminate Grunt

Metasploit is an open-source C2 framework widely used for penetration testing and exploit development. Its modular architecture allows for the seamless integration of custom or pre-existing exploits and payloads [3]. Metasploit facilitates the automation of the various stages of an attack, from reconnaissance to the establishment of a C2 channel.

Covenant is a .NET C2 framework designed for post-exploitation scenarios [8]. It is a versatile toolkit for red teams that enables complex attack chains through an intuitive web interface. Covenant's standout feature is its Grunt—a lightweight, multi-platform implant that communicates with the Covenant server.

Both frameworks exemplify the dual-use nature of C2 systems: they serve as invaluable tools for security professionals in testing and strengthening cyber defenses, yet they also of fer attackers a way to take advantage of vulnerabilities. By analyzing these frameworks, researchers can uncover the tactics, techniques, and procedures (TTPs) employed by adver saries, which in turn informs the development of robust intrusion detection systems (IDS).

Remark 2.1 (C2 Frameworks' Commonalities and Differences). Metasploit and Covenant, while serving similar purposes, differ in implementation and usage. Metasploit's extensive database of exploits and its compatibility with various platforms make it a universal tool for vulnerability testing. Covenant's focus on stealth and post-exploitation tactics provides red teams with advanced capabilities in simulating sophisticated cyber threats. Another major difference is that Metasploit's is handled through a terminal while Covenant requires a web GUI.

Meterpreter	Covenant	Merlin	Cobalt Strike
- Ease of use - Really focused on Exploitation and not just Post - Very stable especially compared to other frameworks - Open-source - Intuitive and easy to use	- Extensible features - Web GUI - Docker - Open-source	- Implants have good AV evasion - Extensible features - HTTP/2 support - Open-source	- Collaborative - Well Tested - Highly Capable - All-in-One - Recon, Phishing, C2, Post-Exploitation
- Limited GUI, CLI driven - Building new modules can be complex	- Building Tasks can be time consuming - Written in C#, not the most userfriendly language	- Can be buggy	- Windows Only - Closed source

Table 1: Advantages and disadvantages between some C2 Frameworks

Other frameworks exist, and there are a lot of them. Below is a table that illustrates the pros and cons of Meterpreter and Covenant compared to two other popular frameworks. Understanding the frameworks is essential for the development of effective IDS that can recognize and mitigate the threats posed by C2 activity. This research project aims to dissect these frameworks to inform the design of a detection methodology that is sensitive to the subtleties of process injection and memory-based execution techniques commonly employed in advanced C2 operations.

3. Setup

The C2Configuration class centralizes command mappings tailored to different malware types. This abstraction layer ensures flexibility in dealing with various malware without altering the core command execution logic.

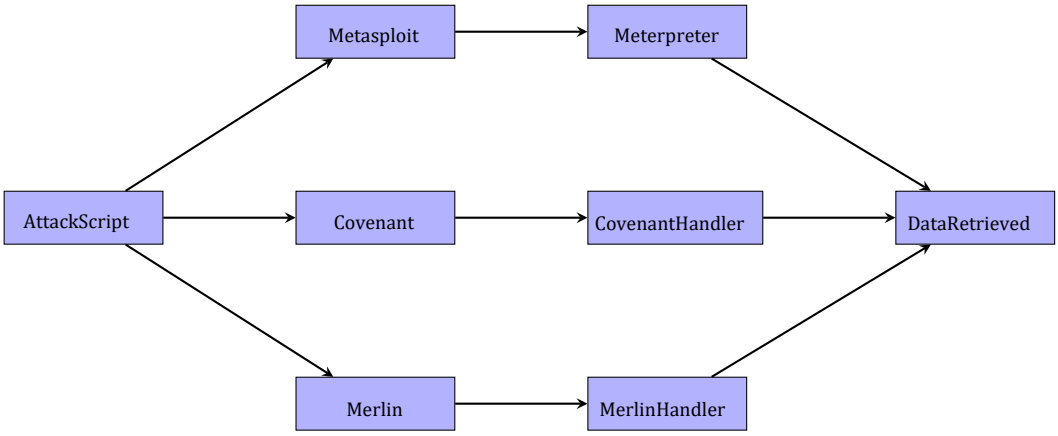

Figure 1: shows a diagram that outlines the structured workflow of the C2 framework.

Once I finish writing an attack script (such as Figure 2), I can run the program which will translate the attack script to the desired C2 handler. The next crucial step involves initializing both the configuration settings and the client for the C2 operation. This initialization could encompass a range of parameters from target IP addresses and authentication details to protocols and ports. However, with SSH, I was able to automate this process. It must be noted that the initialization requires the IP addresses of the virtual machines. To fix this problem, I have created a set address for the victim and attacker machines.

Once the foundational setup is ready, the framework progresses to establish a secure connection with the intended target through Connect via SSH. SSH, or Secure Shell, facilitates secure remote logins and other encrypted network operations over potentially insecure networks. If the connection is successful, the system might deploy a specialized payload (Malware) if the situation warrants it. This payload, renowned in cybersecurity and penetration testing circles, provides the operator with a heightened degree of control over the target, leading to potential further exploitations.

After the necessary configurations, the C2 framework is primed to Execute Com-mands on the target system. These commands could range from data extraction to further exploitation, depending on the specific capabilities of the framework. However, after the designated operations are complete, it's of importance to terminate the connection securely. Thus, the framework advances to the Disconnect from SSH phase, ensuring a traceless and clean disengagement from the target.

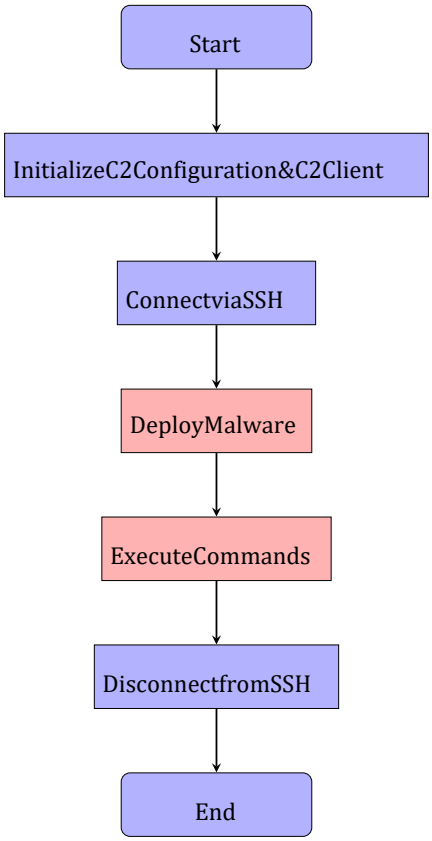

Figure 2: C2Abstraction functionality Framework

The steps colored in blue are the parts that are fully automated through Python. However, there are still aspects of deploying the attack that require manual input. My current progress is focusing on finding a solution to make the testing environment fully autonomous. I am currently researching aspects of IT automation to see if I can apply a similar methodology in the progression of the abstraction layer. If no probable solution exists, then I will resort to using SCP for automatic file execution.

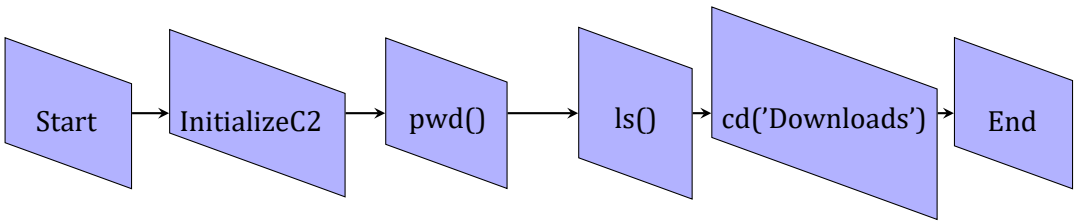

Figure 3: SampleBasicTerminalAttackScript

The virtual machine that I am using is Kali Linux. Previous work has used it for WiFi penetration testing and web application security analysis [2,4]. Preliminary testing has shown that IT automation may work with Kali Linux but further testing is required.

4. Algorithm

The C2Client setup and execution process is primarily influenced by the network operations (SSH connections, payload deployment) and the number of configurations and commands. The time complexity can vary significantly with network-related operations, while the space complexity is largely dependent on the size of command mappings and the number of commands executed. The most time-consuming operation is likely to be the payload deployment due to network latency and the size of the payload. Meanwhile, the space complexity is modest and mainly dependent on the data structures used to store the commands and their mappings.

The analysis leads to the algorithm in Algorithm 4.1.

Algorithm 4.1 Formalized C2Client Command Execution Process
Require: M = $\{m_1, m_2, ..., m_n\}$, a non-empty set of malware configurations.
Ensure: Sequential execution trace of C2 operations. **for all** $m \in$M **do**
Define C(m) as the C2 configuration for malware m.
Let C be a new C2Client instance with configuration C(m).
Define $S_{attacker}$ and S_{victi}m as SSH sessions for attacker and victim.
Establish Sattacker and Svictim. **if** m is ”Meterpreter” **then**
$P \leftarrow$ DeployPayload(); Set up Metasploit with P.
else if m is ”Covenant” **then**
InteractWithUI(); Automate Covenant setup via PyAutoGUI.
end if
Execute $\Gamma = \{\gamma pwd, \gamma ls, \gamma cd, \gamma ps\}$ over Sattacker.
Terminate Sattacker and Svictim. **end for**

5. Experimental Results

In the pursuit of creating a comprehensive testing environment for Intrusion Detection Systems (IDS), various methodologies were employed to simulate attack scenarios. The primary focus was on the automation of malware deployment and execution within this environment. Initial experiments were conducted to automate these processes using IT automation tools. However, these attempts did not yield the desired level of reliability and control required for precise IDS evaluations. Thus, the decision was made to utilize Secure Copy Protocol (SCP) for the deployment of attack scripts and malware payloads.

Subsequent experiments involved the use of SCP to transfer malicious files to the target virtual machines. This method proved to be both robust and reliable, ensuring that the exact contents of the payload were delivered without modification, a critical factor in the success of the simulated attacks. Moreover, SCP allowed for the preservation of file permissions and attributes, which is crucial for the execution of certain types of malware.

To evaluate the effectiveness of the IDS in detecting and responding to various C2 activities, a complete correlated analysis IDS system is being developed to be tested. There are telemetry collections that must be implemented before the full testing scheme can be run.

6. Conclusions

The research presented in this paper has contributed a significant step forward in the development of robust Intrusion Detection Systems (IDS) by providing a comprehensive testing environment capable of simulating a wide range of malware attacks. The establishment of a C2 abstraction layer has proven to be a pivotal innovation, enabling the execution of diverse attack scenarios against which the effectiveness of IDS can be measured and enhanced.

Despite initial attempts to implement IT automation tools for malware deployment, the findings underscored the necessity of using Secure Copy Protocol (SCP) to ensure precise and reliable setup for IDS evaluation. The transition to SCP not only bolstered the reliability of the testing environment but also preserved the integrity of the attack simulations, a critical 158 aspect for the accurate assessment of IDS capabilities.

The experiments conducted have laid the groundwork for a complete correlated analysis IDS system, with telemetry collections in place for future testing schemes. The research has illuminated the strengths and weaknesses of current IDS technology, specifically highlighting the challenges in detecting in-memory execution and advanced persistent threats that leverage process injection.

As I move forward, it is clear that continuous enhancements in IDS technology are required to address the evolving landscape of cyber threats. The research outcomes also suggest that a more autonomous testing environment could further streamline the evaluation process, potentially through the application of more sophisticated IT automation techniques or the 168 development of new tools.

In conclusion, this research has not only advanced the understanding of how various C2 frameworks can be leveraged to improve IDS but also provided a solid foundation for future work in the field of cybersecurity. The ongoing development of the C2 abstraction layer and the testing environment promises to yield further insights into effective strategies for detecting and countering cyber threats and defeating rule-based IDS and their shortcomings [5].

Acknowledgments

I would like to thank you Dr. Rajeev Barua for his unwavering support. He is a great mentor who introduced many great opportunities to me. I was able to learn about what it means to be a graduate student by participating in weekly group meetings and sharing my research.

References

1. M. S. Akhtar and T. Feng. (2022). Malware analysis and detection using machine learning algorithms. Symme180 try, 14.
2. Babincev, I. M., & Vuletić, D. V. (2016). Web application security analysis using the Kali Linux operating system. Vojnotehnički glasnik, 64(2), 513-531.
3. Kennedy, D., O'gorman, J., Kearns, D., & Aharoni, M. (2011). Metasploit: the penetration tester's guide. No Starch Press.
4. Lu, H. J., & Yu, Y. (2021). Research on WiFi penetration testing with Kali Linux. Complexity, 2021, 1-8.
5. Macedo, E. J. S. (2022). Signature-Based IDS for Encrypted C2 Traffic Detection.
6. Maniriho, P., Mahmood, A. N., & Chowdhury, M. J. M.

(2022). A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges. Future Generation Computer Systems, 130, 1-18.
7. Mirolo, C., Izu, C., Lonati, V., & Scapin, E. (2021). Abstraction in Computer Science Education: An Overview. Informatics in Education, 20(4), 615-639.
8. Sen, U., & Sinturk, G. (2018). Normalizing Empire's Traffic to Evade Anomaly-based IDS.