Arizona State University

Senior Capstone Project

Eshu and Ender: Common Language Platforms for Multiple Command and Control Frameworks

Alexander Aviles, Yousif Alsabah, Alan Ingersoll, Xun-Yang Leong, Prateek Ravindran

Ira A. Fulton School of Engineering

Sponsored by

Nathan Smith and Matthew Keeley
ProDefense

Acknowledgments

Our team would like to express extreme gratitude to the ProDefense [1] team, Nathan Smith and Matt Keeley, for the opportunities, direction, and support they provided over the duration of this project. Their ideas, which spurred this capstone project, are novel and important. Our team recognizes the future prospect that platforms Eshu and Ender provide with regard to offensive Cybsersecurity. We also want to thank Arizona State University's Ira A. Fulton School of Engineering staff for the assignment to our apprenticeship under ProDefense. The stars aligned so that we could shed light on a new avenue of research and development in this Cyber Age.

Lastly, to Nathan Smith, our primary contact and mentor, who guided us through technological hiccups compounded by inexperience, may this paper make you proud.

Contents

List of Figures

Abstract

Platforms Eshu and Ender were created for red-team operators or other individuals who utilize multiple command and control frameworks to carry out their responsibilities. They are meant to serve as a common language platform, a utility that enables these operators to perform their exploitation and post-exploitation nearly completely agnostic of the various command and control (C2) frameworks they may be utilizing during a given engagement. This project is motivated by the growing dependence on multiple C2s for red teaming. C2 frameworks perform varying tasks in varying manners with varying outcomes, but the end goal is always the same: do some thing on some vulnerable machine, or two things, with three machines, and so on and so forth. Making these tasks easier benefits individuals who are limited by their active command and control set-up. The original scope of this project was to take three very common C2s: Metasploit, Sliver, and Havoc C2, and build a Python library that would interface with the operator between all three in an agnostic manner. This scope was reduced to two C2s, Metasploit and Sliver, after the original direction shifted, and academic time constraints arose before the start of development of the exploit engine. The following is a description of the problem, the development of solutions, and the results and future work of the projects Eshu and Ender.

1 Introduction

Arizona State University Computer Science majors are required to complete a Senior year capstone project. A list of available, sponsored propositions are assigned across teams in the student body. It is the team's responsibility to coordinate with the sponsor, establish a development plan, and execute that plan until a minimum viable product is produced to the appeasement of the sponsor. Our team, having received ProDefense as our sponsor for project "Eshu: Common Language Platform for Post-Exploitation Frameworks", were tasked with building Eshu for post-exploitation during the first semester, and Ender for active exploitation during the second semester. We began with post-exploitation as advised by our sponsor Nathan because he believed post-exploitation development would be easier to tackle since all of these technologies and methodologies were relatively new to us. Our team set a biweekly meeting schedule, utilized Jira for project management, and initiated the project by setting up a development environment. This environment was a networked Docker Desktop environment that consisted of the "Operator" service as our attacker machine, as well as the renowned Metasploitable 2 vulnerable machine.

Metasploit and Sliver are incredibly popular command and control frameworks used in the industry for both research and practical applications. It is these two frameworks that our common language platforms interface with and communicate via certain API's that will be discussed later in the paper. In truth, these two programs fall short of their full intended capabilities, but we believe heartily as a team in the implied practicality and usefulness of each program independently. Eshu and Ender are simply the seed of an idea that could send the Cybersecurity research industry into a thriving age. We leave this paper as a testament to our own research and development, which may lead someone to planting this seed and letting it grow beyond what we conceived to be capable.

2 Purpose

Powerful modern command and control (C2) frameworks are the heart of red-team research and exploitation. They are the vital organ of Lockheed Martin's sixth Cyber Kill Chain [2] tenet: 'Command and Control'. C2 frameworks, powerful tools for managing implants and agents on compromised machines, have a variety of options. Operators can choose such tools based on their needs and have resources like the C2 Matrix [3] to guide them. This flood of possibilities however can lead to complicated workflows, convoluted learning, and most importantly: inefficient operations. An operator's success is dependent on their efficiency and capability. Therefore, our team looked into solving the problem of maintaining efficiency while increasing capability within an operator's active campaign. Many C2s already incorporate "interfacing" in some way through their own APIs for other applications, or their own embedded system can be used in relation to a third party. For instance, Metasploit's Metasploit Framework (MSF) is capable of utilizing Sliver payloads using custom payloads [4]. Or in a reverse manner, Sliver supports Meterpreter staging protocols [5].

Understanding these behaviors, our team wanted to develop an expandable application that would enable agnostic control between the various C2 frameworks during an active operation. This post-exploitation platform would be capable of managing any number of active sessions, set up by operators without worrying about a specific C2. For instance, an operator using both Metasploit and Sliver would be able to query their exploits during post-exploitation without explicitly interacting with both Metasploit's and Sliver's user interfaces. This is strictly Eshu's proposed capability.

Additionally, active exploitation can be made agnostic as well, to a certain degree. Deciding which framework to use during exploitation is simply part of the necessary steps, however, creating a singular environment to perform operations between multiple C2 is Ender's proposed capability.

Considering the technology available to us, the ability to achieve these goals was a guarantee. These products are a quality of life improvement for red-team operators, simplifying multi-C2 management for in-house penetration testing, vulnerability analysis, and security consultancy. As long as the products were user-friendly, support the variety of testing needs customers may have, and improve general efficiency between multiple frameworks, we could say we achieved our immediate goals.

3 Eshu Methodologies and Solutions

Our team first developed Project Eshu. This was our solution to post-exploitation querying of active implants on vulnerable machines. The baseline goal was to enable operators to query and manage already established implants regardless of C2 choice. This meant standardizing commands that retrieved host information, could execute commands remotely, and handle the various sessions across the multiple frameworks that an operator may use during any given engagement. We minimized these requirements into a single core capability: execute shell commands remotely. This alone meant we would be able to "query" the target operating system as well as receive other details. Our team decided to restrict our developmental frameworks to just Metasploit and Sliver specifically after we set up our development environment. We also needed to understand which API options were available to us and that we could use for both Metasploit and Sliver to interact with Eshu. We were aware that Metasploit can be used to bootstrap Sliver exploits but wanted complete independence with respect to Eshu's capability to interface with a variable number of frameworks. We decided on Metasploit's RPC Client by using Pymetasploit3 [6] and Sliver's SliverPy [7] library which utilizes the gRPC protocol to communicate with both the Sliver server and Sliver client.

Furthermore, considering these were relatively new technologies for us, we had to understand an actual operator's workflows. We devised sequence diagrams to better understand how a red team operator would use these frameworks to perform their duties.

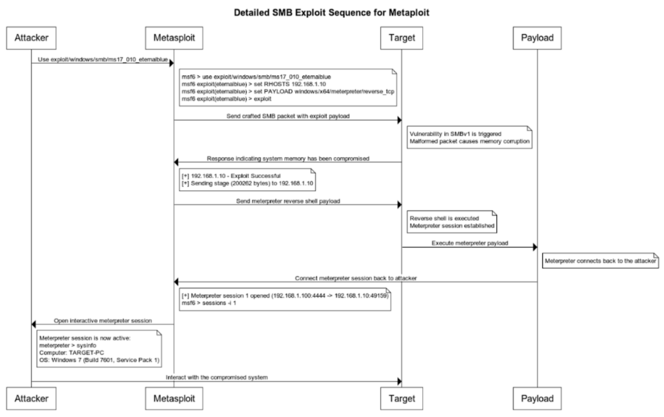
Detailed SMB Exploit Sequence for Metaploit
Attacker
Metasploit
Target
Payload
Use exploit/windows/smb/ms17_010_eternalblue
msf6 > use exploit/windows/smb/ms17_010_eternalblue
msf6 exploit(eternalblue) > set RHOSTS 192.168.1.10
msf6 exploit(eternalblue) > set PAYLOAD windows/x64/meterpreter/reverse_tcp
msf6 exploit(eternalblue) > exploit
Send crafted SMB packet with exploit payload
Vulnerability in SMBv1 is triggered
Malformed packet causes memory corruption
Response indicating system memory has been compromised
[*] 192.168.1.10 - Exploit Successful
[*] Sending stage (200262 bytes) to 192.168.1.10
Send meterpreter reverse shell payload
Reverse shell is executed
Meterpreter session established
Execute meterpreter payload
Meterpreter connects back to the attacker
Connect meterpreter session back to attacker
[*] Meterpreter session 1 opened (192.168.1.100:4444 -> 192.168.1.10:49159)
msf6 > sessions -i 1
Open interactive meterpreter session
Meterpreter session is now active:
meterpreter > sysinfo
Computer: TARGET-PC
OS: Windows 7 (Build 7601, Service Pack 1)
Interact with the compromised system
Attacker
Metasploit
Target
Payload

Figure 1: Sequence diagram illustrating the workflow of the Metasploit framework.

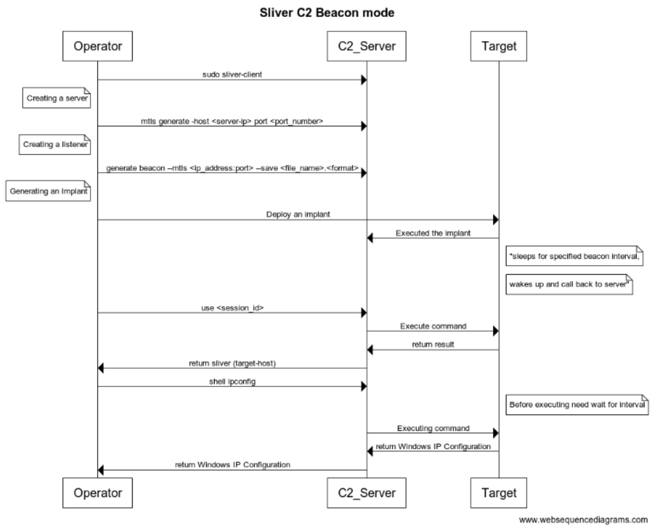
Sliver C2 Beacon mode
Operator
C2_Server
Target
sudo sliver-client
Creating a server
mtls generate -host <server-ip> port <port_number>
Creating a listener
generate beacon --mtls <ip_address:port> --save <file_name>.<format>
Generating an Implant
Deploy an implant
Executed the implant
"sleeps for specified beacon interval,
wakes up and call back to server"
use <session_id>
Execute command
return result
return sliver (target-host)
shell ipconfig
Before executing need wait for interval
Executing command
return Windows IP Configuration
return Windows IP Configuration
Operator
C2_Server
Target
www.websequencediagrams.com

Figure 2: Sequence diagram illustrating the workflow of the Sliver framework.

Originally, our team had explored building a REST API with Python Flask [8], however it limited our development capabilities. For one, we were limited in flexibility and expansion. The Flask app would have convoluted our designs unnecessarily be requiring us to adhere to web application standards. We wanted the C2 interactions to be native to the Operator, so they have full control over their environments. Therefore, we were forced to pivot in our development of Eshu but already had a lot of the groundwork necessary to complete the project finished, as well as our understanding of requirements.

Maintaining an rigorous testing and development cycle, we established a set of critical functionalities in Eshu. We built framework initiation in the backend in order to tie Eshu to the frameworks actively in use. Additionally, Eshu registers a C2 per session which maps which framework an exploit session is engaged with, and then has locally scoped session storage to recall the active sessions once accounted for by Eshu.

An Operator launches a C2 via Eshu's backend option for that C2, and then they must register the instance into Eshu using the Python line:

```
Eshu.Eshu().register(c2Instance).
```

Currently, only Metasploit and Sliver are built out in the backend per our project specifications. However, since each C2 is managed similarly, the structure and code necessary to add additional C2's is not hefty.

First is the get-hosts command which identifies and pulls the active exploits or sessions capable of being managed by Eshu. Second is the critical run-command function. This function takes the uniquely designated session id for the active sessions stored in Eshu, and a command or set of commands to run against the active session, as well as an optional OS parameter. The unique id uses nomenclature which specifies the C2 that the session is connected to and the count of that session for that C2. Although the names very therefore, the result is technically agnostic on the Operator's side.

This summarizes the current state of Eshu's main class and functionality.

As stated previously, strictly Metasploit and Sliver have been built out in the backend. Starting with Metasploit, Eshu needs to initiate the MSF RPC client. Then there are the two functions that work in sync with the Eshu class for each C2. First is the query-hosts function. This function retrieves the active sessions and certain metadata regarding each session from the Metasploit server, which get stored to display the information to the Operator. Secondly, is the send-command function. This function retrieves the session matching the given session ID and runs the command set provided against the target machine. Lastly, is Sliver's backend. Again the first step is to launch the Sliver C2 framework with Eshu so that it can be registered like earlier.

Then like Metasploit, Sliver accommodates the two primary functionalities of Eshu in its backend. The query-hosts function similarly identifies the active sessions and pulls Sliver specific metadata to store and display to the Operator for post-exploitation operations.

Next, Sliver's send-command function is built specifically to handle Sliver beacons, and not sessions it should be noted. Although we limited our testing of Sliver to beacons only, that is a capability we would like to eventually include. Although hopes for future work will be discussed later. The send-command function also takes an agnostic session ID and command set, finds the stored session and attempts to execute the commands against the target machine.

These backend C2 specific classes are critical to Eshu's functionality. And their independent nature allows for new additions to be included later, depending on whichever C2's the Operator chooses to use during a campaign. Please see the ProDefense Github [9] for references about the code.

4 Ender Methodologies and Solutions

Our team worked on Ender after completing Eshu's minimum viable product. First, we developed the actual user interface for Ender. Unlike Eshu, we realized that for active exploitation we would need a responsive command line interface or CLI to mimic typical C2 framework behavior because most C2's at least have a TUI or GUI that operators can rely on to interface with the framework firsthand. To mimic this behavior that meant deciding on an architecture to use. Thus, our team established the Ender-client and Ender-server. This client-server communication would enable a lightweight client to interact with the operator while the server performed all of the remote procedure calls to each independent C2 on the backend. Requirements for Ender included listing and/or searching for available payloads for both Metasploit's Exploit modules and Auxiliary modules, enabling an operator to configure the modules, and execute them, all while receiving feedback via the RPC in both Ender's client and server.

Utilizing Pymetasploit3, we first implemented Ender's Metasploit control before Sliver. This client-server architecture would also enable us to implement Sliver's communication in a more modular manner later on. The client is responsible for passing along and controlling the messages depending on keywords. Then the server's handle-message() function would receive the command and perform accordingly. Once the initial state of Ender was complete, the operator only need launch the Ender-client and Ender-server and enter the connect command.

Ender ensures programmatic querying once in the process of executing a Metasploit module to ensure the operator initiating the operation can input proper parameters depending on the exploit/auxiliary module's options or their necessities. This results in Ender's capability of running and executing any of the modules chosen by the operator on a target victim machine. The final development for Metasploit's end of Ender before attempting to code the agnostic session handling which is the end goal, was catching a Meterpreter shell.

We were able to catch a Meterpreter shell, which is a payload that establishes an interactive channel between the attacker and the victim. This is performed by setting a listener on the local host, triggering the payload, and then receiving the interactive session. Once the interactive session is complete, which establishes a basic shell, running the "post multi/manage/shell_to_meterpreter" command establishes the Meterpreter shell.

Implementing Sliver had its own difficulties, specifically when it came to Sliver's own client and server. This required us to ensure when interacting with Ender that the Ender client and server always stated connected to the Sliver client and server. Original implementation of Sliver behavior in Ender was consolidated to a singular function, however that was extrapolated to multiple modular functions. These functions work together to establish, maintain, and control the traffic between Ender and Sliver. Additionally, Ender's implementation allows an operator to establish profile config files, generate beacons, use multiplayer mode in the server, set up an http listener, and make general command queries.

The primary end goal for Ender's capability was enabling agnostic implant handling, or simply performing as a session handler. For instance, once an active session was established between Metasploit and the operator or Sliver and the operator, they could interchangeably transfer control of the implant between connected frameworks. This is currently not possible to our knowledge however. Considering we used Metasploit's RPC API and Sliver's gRPC protocols, the two handle their independent channel communications differently and therefore we were limited in our ability to handle these implants in a completely agnostic manner. Therefore, we attempted a more brute force methodology of enabling the operator to perform techniques that transfer control from one C2 to another by injecting new sessions or implants via the already in control C2. The two methods considered were Session Passing and "Bring Your Own Stager" or BYOS [10].

Like Eshu, please see the project code on the ProDefense Github [11] for reference.

5 Future Work Discussion

Unfortunately, the goal to enable an operator to perform "Bring Your Own Stager" session passing as a primary alternative and session passing as a secondary back up methodology were not fruitful in time for our Senior Capstone Exposition Day. These two implementations would have turned Ender into a near unified session management tool, however our final attempts only resulted in a grouping of template or skeleton code with a few functions that could work but did not result in an active session transfer.

Beyond the addition of more C2 frameworks such as Havoc or Cobalt Strike, what we believe would be valuable to include later are the handling of Sliver's sessions along with the beacons implementation already in place. Sliver beacons and sessions are similar but still behave differently enough that it would require additional programming to have full interoperability for the chosen implants and payloads on Sliver. Metasploit already works with both exploit and auxiliary modules. So it is important to consider what types of implants a framework is capable of generating and staging in order to incorporate full functionality for future frameworks.

Lastly, our team firmly believes this is a novel idea, and a novel project that may be capable of significantly impacting the security industry. Therefore, with approval from our sponsors, we have opened up Eshu and Ender for open source development. We understand these are not completely secure programs, or complete programs in any sense of the word. However, we are proud of what we did accomplish and what it implies for the capabilities of a project like this in the future.

6 Conclusions

In conclusion, Eshu and Ender are projects that could drastically change the nature of Cybersecurity. These platforms set a precedent for the future of this field, which is all about innovating ways to advance our mindsets and our capabilities as a common people. With all of this team's due effort in the final stretch before the project close, we were unsuccessful in implementing Ender's would-be standout characteristic of session handling in an agnostic manner. But we are proud of our work. We have developed products that work, are efficient, and consolidated. Eshu and Ender are stepping stones into the ways professionals can take advantage of the technology they use everyday, and re-purpose the way they approach their daily tasks. This is an ever-evolving field. Security practitioners cannot become complacent or they will be overtaken by the wave of innovation and discovery that occurs on a literal daily basis. We hope the right person can pick up where we left on in a future endeavor that sees total management from a single point of contact for multiple command and control frameworks come into existence.

7 Glossary

This glossary details the list of special terminology (abbreviated or not) that is referenced throughout the paper.

7.1 Technical Terms

C2 Command and Control Framework, a tool utilized in the context of Cybersecurity to coordinate incident response, perform penetration testing, and main systems control.

C2 Matrix a resource pool for red-team operators of command and control frameworks.

CLI Command Line Interface.

Docker Desktop Docker Desktop, application for running containerized services and applications.

Ender Common Language Exploit Engine for multiple Command and Control Frameworks.

Eshu Common Language Post-Exploitation Platform for multiple Command and Control Frameworks.

Jira Jira Software, a project management system.

Metasploit Metasploit Framework, a command and control (C2) framework.

Metasploitable 2 a vulnerable Ubuntu Linux virtual machine.

Meterpreter a Metasploit exploit payload that provides an interactive shell when executed.

pymetasploit3 a Metasploit automation library.

RPC Remote Procedure Call.

Sliver Sliver C2, a command and control (C2) framework.

SliverPy a client library for a Python gRPC to communicate with Sliver over Mutual TLS.

8 References

[1] M. Keeley. ‘Prodefense - about,’ Accessed: 23 Jan. 2025. [Online]. Available: https://www.prodefense.io/about

[2] L. Martin. ‘The cyber kill chain,’ Accessed: 26 Mar. 2025. [Online]. Available: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[3] J. Orchilles. ‘C2 matrix,’ Accessed: 26 Mar. 2025. [Online]. Available: https://howto.thec2matrix.com/

[4] Z. Goldman. ‘Byos: Bring your own stager,’ Accessed: 26 Mar. 2025. [Online]. Available: https://www.rapid7.com/blog/post/2022/09/16/metasploit-weekly-wrap-up-176/

[5] B. Fox. ‘Stagers,’ Accessed: 26 Mar. 2025. [Online]. Available: https://sliver.sh/docs?name=Stagers

[6] C. C. Team. ‘Pymetasploit3 – metasploit automation library,’ Accessed: 28 Mar. 2025. [Online]. Available: https://coalfire.com/the-coalfire-blog/pymetasploit3-metasploit-automation-library

[7] moloch. ‘Welcome to sliverpy’s documentation!’ Accessed: 28 Mar. 2025. [Online]. Available: https://sliverpy.readthedocs.io/en/latest/

[8] Pallets. ‘Byos: Bring your own stager,’ Accessed: 1 May 2025. [Online]. Available: https://flask.palletsprojects.com/en/stable/

[9] M. Keeley. ‘Prodefense,’ Accessed: 28 Apr. 2025. [Online]. Available: https://github.com/ProDefense/Eshu

[10] P. Projects. ‘Flask,’ Accessed: 15 Apr. 2025. [Online]. Available: https://www.rapid7.com/blog/post/2022/09/16/metasploit-weekly-wrap-up-176/

[11] M. Keeley. ‘Prodefense,’ Accessed: 28 Apr. 2025. [Online]. Available: https://github.com/ProDefense/Ender