DeadDrop

a clever command and control framework for penetration testers

Software Requirements Specification (P2)

Jann Arellano, Brian Buslon, Keaton Clark, Lloyd Gonzales
Team 08

Advisor:
Shamik Sengupta
Professor
University of Nevada, Reno

CS 425, Fall 2023
Instructors: Sara Davis, Devrin Lee, Vinh Le

Department of Computer Science and Engineering
University of Nevada, Reno
November 3, 2023

Table of Contents

I. Introduction

DeadDrop is a command and control (C2) framework used in post-exploitation activities and penetration testing to create malware payloads, manage compromised devices, and generate operational reports. DeadDrop's primary purpose is to aid legitimate security professionals (the "red team") in an activity known as penetration testing, which assesses the company's ability to defend against attacks and protect its data.

In a penetration test, the red team identifies vulnerabilities in an organization's devices, using these vulnerabilities to install malware that allows them to remotely control and communicate with that device. By leveraging the "trust" of this compromised device, the red team can then plant malware on more valuable devices with sensitive information (some of which may not be directly connected to the public internet), with the goal of exfiltrating this sensitive data out of the network without being caught.

Penetration tests are often coordinated using C2 frameworks, which greatly simplifies the task of managing infected devices while ensuring accountability for actions taken by the red team. The use of a C2 framework in testing greatly benefits security engineers (the "blue team"), as they can identify holes in detecting malicious activity through the reporting and logging functionality provided by the C2 framework.

However, unlike existing frameworks that communicate directly with attacker domains, DeadDrop focuses on leveraging features in legitimate websites such as YouTube and Wikipedia to communicate with devices and exfiltrate data, masking its activity within the noise of popular websites. The novelty of this approach is valuable to offensive and defensive cybersecurity professionals alike, as corporate networks and firewalls often "trust" the traffic of large, well-known websites.

By abusing this trust, DeadDrop provides security engineers with new insight into identifying covert attacker techniques and security weaknesses. In doing so, it encourages network security engineers to take new approaches to identifying covert communication methods that could be used by skilled threat actors. Ultimately, DeadDrop provides organizations with the tools needed to develop and test their defenses against these techniques through a highly configurable and extensible framework.

No changes in core functionality, architecture, or scope were made between Project Assignment 1 and the completion of this document. However, the insight gained from our stakeholder interviews has allowed us to better define and prioritize our requirements. In particular, we note the following statements stressed by *all* of our stakeholders:

- The scope (of what is allowed to be tested) must be well-defined to avoid disrupting critical systems in an organization. While not explicitly stated before, we intend to implement "guardrails" in DeadDrop that allow attackers to define machines that cannot be attacked under any circumstance.

- Every communication and action taken must be logged in such a way that they can be associated with the user who performed the action, providing accountability.
- In most cases, there is no urgency in executing commands; it may be several hours before an agent executes a command issued by the server. However, it should still be possible to complete time-sensitive commands.

In addition to defining requirements and use cases, we also provide initial snapshots of DeadDrop's web interface, which will be used by penetration testers to interact with the framework itself. We also provide a glossary of technical terms common in the field of penetration testing, as well as technical resources that we believe will be helpful in architecting and implementing DeadDrop.

II. Stakeholder Interviews

Questions Provided

Each stakeholder was provided with the following context and set of questions.

As background, DeadDrop is a C2 framework used to facilitate penetration tests. In a C2 framework, penetration testers use a C2 server to communicate with malware planted on exploited devices in a network. The malware - more commonly known as a C2 agent - can be used to run arbitrary commands on the infected machine and forward communications from C2 agents that are in better-protected layers of the network (and may not be directly connected to the internet).

The novelty behind DeadDrop is that it uses services such as YouTube, Wikipedia, and Reddit to avoid direct communication between the C2 server and its agents, which can make malicious traffic more difficult to initially detect. This stands in contrast to many existing C2 frameworks, which typically directly communicate with an attacker-held domain over the internet.

With this context and your cybersecurity experience in mind, please answer the following questions in as much or as little detail as you'd like:

1. In your experience, what are some common vulnerabilities that are often identified in penetration tests?
2. What are some common precautions taken by penetration testers to avoid accidental disruption to production systems?
3. How do penetration testers ensure that a particular action is within scope (or has otherwise been authorized by the target)?
4. What are some legal considerations when performing penetration tests?
5. Given the importance of documentation in penetration testing, what logging and reporting functionality should be provided by the framework?
   → *For example: what actions and communications should be logged? Should the framework be capable of auto-generating reports, and if so, what should it include?*
6. What functionality should always be available in every C2 agent?
   → *This refers to common tasks taken by penetration testers, such as dumping a list of all credentials in memory, stealing session cookies from browsers, or enumerating the file system that would not require more than a simple command to execute.*
7. What is an acceptable delay for communication between the C2 server and C2 agents?
   → *In other words, what is an acceptable delay between a command being issued by the C2 server and a response being received from the C2 agent?*
8. What are some common ways that malware or malicious communications might be caught by network or endpoint security?
9. What features or qualities of existing penetration testing tools do you like (or dislike)?
   → *Some examples might include a simple user interface, built-in support for scripting (such as in Ghidra), or being lightweight.*
10. How important is it that discovered vulnerabilities can be mapped to particular attack models, such as MITRE ATT&CK, for the purposes of documentation?

11. What visualizations would be helpful throughout the course of a penetration test?
    → *Visualizations might include a graph view of the network as machines are discovered, or a directory tree listing of files on a filesystem.*
12. How important is it that individual users are held accountable for any actions they take using a C2 framework, and why?

Optionally:

1. Based on your knowledge of cybersecurity, how might you detect the covert communications facilitated by DeadDrop (or defeat the framework altogether)?
2. Are there any particular malware techniques that you think are clever or are otherwise unexpected?

Stakeholders Interviewed

Bill Doherty is an adjunct professor at the University of Nevada, Reno. He was chosen as a stakeholder due to his extensive cybersecurity background in both industry and education, giving us insight into both long-term cybersecurity trends as well as specific recommendations for our offense-oriented project. Bill was given the questions above as a PDF and provided his responses via email.

Shamik Sengupta is a professor at the University of Nevada, Reno. In addition to his role as the project's advisor, he was interviewed because of his expansive research background. This allowed us to explore recent advances and novel techniques described in literature, as well as discuss current industry practices. Shamik was interviewed via Zoom in a 30-minute meeting.

Lloyd Gonzales is a member of the team developing DeadDrop. He was interviewed due to his cybersecurity internship experience and history in cybersecurity capture-the-flag competitions, which allowed him to blend his industry knowledge with novel offensive techniques. Lloyd provided his answers as a PDF.

Summary of Stakeholder Responses

The following is a summary of the collective responses of each stakeholder interviewed. The summaries provided reflect statements that all stakeholders were in agreement with, as well as any particular comments of note.

In your experience, what are some common vulnerabilities that are often identified in penetration tests?

Everybody noted that the answer would vary based on the type of organization being tested, but common vulnerabilities included network misconfigurations, unpatched public vulnerabilities in software and firmware, vulnerable APIs, hardcoded or otherwise easily accessible credentials, and buffer overflows. Some of these could be automatically discovered or tested, while others need a human to reliably discover.

For certain services - such as containerized applications or web applications - common vulnerabilities are largely detailed in models such as the OWASP Top 10 or the Microsoft Kubernetes Attack Matrix.

What are some common precautions taken by penetration testers to avoid accidental disruption to production systems?

Broadly, all the stakeholders agreed that the scope (the services and devices that could be tested) should not only be well-defined, but it should be carefully defined with the advice of the client to ensure that any critical systems are excluded from testing. Additionally, when vulnerabilities are found, they do not actually need to be exploited; demonstrating that a network is vulnerable to ransomware does not require actually encrypting the data.

Bill noted that certain classes of vulnerabilities, such as those related to buffer overflows, should generally not be exploited on production systems. Additionally, scripts should be used to monitor the status of a system under attack to allow attackers to quickly bring the system back up if needed.

Shamik also noted that where possible, any testing should be done in an isolated replicaof the environment; of course, for large or complex organizations, testing on anything other than the production environment is not always feasible. However, when penetration testing individual systems (rather than whole networks), this is often more reasonable.

How do penetration testers ensure that a particular action is within scope (or has otherwise been authorized by the target)?

Everybody agreed that a detailed agreement with the client (the organization being tested) must be developed; where there is *any* doubt that an action is within the scope of that agreement, the client was to be contacted immediately. Additionally, any action taken should be intentional - any automated tests should be carefully scoped, and “broad” automated tests should be avoided unless there is good reason to do so.

What are some legal considerations when performing penetration tests?
Every stakeholder stressed that as part of the client agreement, there should be (reasonable) liability waivers for the penetration testers. The fact is that no amount of preparation can guarantee that a penetration test will go perfectly, but a penetration tester should still be able to demonstrate due diligence in trying.

In particular, Bill mentioned that disclosure of customer information was a concern, which can happen unintentionally (such as by storing data in cloud services while testing). This is particularly relevant for organizations that are subject to regulations such as HIPAA and FERPA, in which case both the client and the testers must agree on the procedure taken when sensitive data is discovered or logged.

Given the importance of documentation in penetration testing, what logging and reporting functionality should be provided by the framework?
All stakeholders agreed that everything that can reasonably be logged should be logged; this includes every piece of communication between the server and the clients, as well as any actions executed by individual testers. The details that should be attached to each log item include the date, time, function executed, and source/destination where applicable.

Bill suggested that with respect to reporting, a summary of all tests performed would be useful; having every single action logged is important, but not necessarily useful for leadership unless conclusions can be drawn from the logs. Manual annotations should be supported as well.

In contrast, Shamik explained that automated reporting was relatively rare; a large part of penetration testing is using the experience of a human to adapt to the unique environment of an organization being tested. All the data necessary for a report to be created should be available to testers, but it should not be up to the framework itself to create these reports.

Lloyd noted that most mature C2 frameworks do provide a mechanism to log every action, though with varying degrees of granularity and configurability. Debug logs could be used in reporting, though are less reliable due to their lack of a standardized format.

What functionality should always be available in every C2 agent?
The answers here varied quite widely. Since Shamik had limited hands-on experience using a C2 framework, he provided similar suggestions to prior questions. Bill noted that this would depend on the actual host being tested, but helpful functionality included:

- Automatic discovery of devices on the network
- Automated vulnerability scanning for known CVEs, based on the service or device being tested
- “Live” remote session tools like SSH, RDP, or telnet
- Password/credential dumping tools

Lloyd had similar thoughts on the functionality that should be included, based on commands available in Mythic and Cobalt Strike (two mature C2 frameworks). Most frameworks included dedicated functionality for directory listings, single file downloads, and executing shell commands.

What is an acceptable delay for communication between the C2 server and C2 agents?
Bill and Shamik agreed that real-time communication (on the same level of latency as a live SSH connection) was generally not necessary; if a penetration tester is doing deliberate, methodical testing as they should, there should generally be no need for time-sensitive responses from a client. Indeed, most real-world attackers hide in environments for months, so it is not unrealistic to assume that a response delay of several hours is acceptable.

Lloyd similarly noted that real-time communication was likely not usually necessary, but a delay on the order of several hours or minutes would be unacceptable if an emergency action needed to be taken (perhaps to halt an automated test that is placing stress on a service), and would be impractical in live demonstrations.

What are some common ways that malware or malicious communications might be caught by network or endpoint security?
The answers here varied widely as expected, since there are just as many security solutions as there are types of malware. Everyone agreed that general solutions, such as intrusion detection systems and endpoint security systems (like antivirus solutions) were common in virtually every network. However, the specific methodology of how these were implemented or contributed to an organization’s security varied greatly:

- Lloyd suggested that enforcing all organizational traffic to pass unencrypted through a proxy would allow the traffic to be reliably categorized and inspected for suspicious activity (in theory).
- Bill suggested that externally-initiated connections (such as one from the C2 server) are more likely to be scrutinized than internally-initiated connections (such as one from a C2 agent).
- Shamik discussed how many organizations use some form of baselining, identifying IP addresses or domains that are not typically accessed by the network.

What features or qualities of existing penetration testing tools do you like (or dislike)?
Both Lloyd and Bill agreed that “lightweight” agents would be preferable over agents that are feature-filled but require too many dependencies or high hardware requirements. An ideal agent should be relatively small in size (to help make transferring it easier) and capable of running on a variety of devices, including IoT devices. Additionally, the easier a tool is to use, the more time an analyst can spend actually evaluating results instead of configuring or manually editing the tool.

Shamik stressed that quality-of-life features should not be a focus of the (short-term goals of the) project; things like nice user interfaces are good for demonstration, but would not make an attack any stealthier. Again, with limited hands-on experience, he reiterated statements made in earlier questions.

How important is it that discovered vulnerabilities can be mapped to particular attack models, such as MITRE ATT&CK, for the purposes of documentation?
Both Bill and Shamik agreed that mapping specific vulnerabilities to attack models could be helpful in reporting findings to leadership, but generally was not a high priority. Bill stated that a model like MITRE ATT&CK would be better suited to strategizing rather than executing an attack, though it could aid in identifying gaps in the testing itself.

Lloyd noted that such functionality would likely not be important for the project, although some mature frameworks (such as Mythic) did support mapping discovered vulnerabilities to the MITRE ATT&CK framework.

What visualizations would be helpful throughout the course of a penetration test?
Every stakeholder agreed that visualizations like graph views of the discovered network or neatly-formatted directory listings would be nice to have, but not essential for the project to function. Again, the purpose of the project is to aid in penetration testing, and while ease of use can only help a penetration tester, it is not the main feature that a penetration tester looks for.

How important is it that individual users are held accountable for any actions they take using a C2 framework, and why?
Every stakeholder agreed that this was essential; accountability is particularly important if something goes wrong in a penetration test, as it makes it considerably easier to identify the source of the issue and fix it. From a legal perspective, it also demonstrates to both the client and the penetration testers that the testing agreement is being followed; in the event something does go wrong, it provides both the client and the testers with the necessary evidence to determine who is at fault.

In Bill's words, "the C2 framework is just a tool, and maybe like a gun; if not used correctly, the results can be catastrophic."

Based on your knowledge of cybersecurity, how might you detect the covert communications facilitated by DeadDrop (or defeat the framework altogether)?
The responses provided by the stakeholders all generally took the same strategies; a virus scanner or anything host-based would be likely to succeed in finding DeadDrop's agents, which require privileged system calls to function. Although DeadDrop does use legitimate services, it's still possible to determine anomalous traffic relative to a particular machine or user; a user might not regularly use YouTube, or they might not use it at a particular time of day, or they might not visit a particular channel, and so on.

Every stakeholder also provided similar answers to those in the more general question for catching malware or malicious activity in a network.

<u>Are there any particular malware techniques that you think are clever or are otherwise unexpected?</u>
The following two techniques were discussed by all three stakeholders interviewed:

- Data could be hidden in normally benign protocols, such as "hiding" or slowly transmitting data using ICMP pings or hiding information in HTTP headers.
- Malware could be encrypted to avoid detection by any network-based protections and *some* endpoint security solutions, then decrypted at runtime to allow the code to actually run.

Additionally, Bill added the technique of using steganography to deliver malware, in which a legitimate "cover file" is used to hide underlying malicious data that can be extracted on the target device. For example, perhaps the least significant bits of an image can be combined to form an ASCII-encoded command.

We will note that the responses provided gave us two additional specific ideas, with respect to potential functionality:

- It would be helpful to tag/categorize individual assets on the network, either automatically or manually, to help identify their purpose and criticality to the organization's operations.
- DeadDrop should allow the user to provide a set of devices (by MAC address, IP address, etc.) that are *not* allowed to be interacted with, even if they are discovered.

On a similar note, DeadDrop should also allow the user to provide an "allowlist" of devices that can be tested, warning the user if a particular discovered device is not on the allowlist.

III. Technical Requirements

The keywords "must", "must not", "shall", "shall not", "should", "should not", and "may" in this section are to be interpreted as described in RFC 2119.

The terms “agent” and “server” are interchangeable with “endpoint” throughout this section – that is to say, an endpoint may be either the C2 server or a particular C2 agent.

Functional Requirements

ID	L	Description
FR01	1	DeadDrop shall implement a user authentication system.
FR02	1	DeadDrop shall allow the user to register and authenticate new agents compiled for a specific platform.
FR03	1	DeadDrop shall allow the user to remove and remotely uninstall existing agents.
FR04	1	DeadDrop shall attempt to reach registered agents at a user-defined interval.
FR05	1	DeadDrop shall implement a covert communication protocol that communicates solely over a popular service.
FR06	1	For covert protocols, DeadDrop shall allow the user to visit the URLs being used as a dead drop in a protocol.
FR07	1	DeadDrop shall allow the user to remotely execute shell commands through registered agents.
FR08	2	DeadDrop shall implement a “guardrail” system, preventing the framework from acting on user-defined hosts.
FR09	2	DeadDrop shall implement an “allowlist” system, forcing the framework to act only on user-defined hosts.
FR10	2	DeadDrop agents shall be able to forward messages from other agents that do not have direct internet access.
FR11	2	DeadDrop shall perform a user-defined action in the event an agent is unreachable for a specified number of attempts.
FR12	2	DeadDrop shall record all actions initiated by a user or an agent in a standard format.
FR13	2	DeadDrop shall record all data transferred via communication protocols in a standard format.
FR14	2	DeadDrop shall allow users to export logs to a standardized format.
FR15	2	DeadDrop agents may implement convenience commands for common but complex actions.

ID	L	Description
FR16	2	DeadDrop shall allow users to remotely connect to the server and interact with agents.
FR17	2	DeadDrop shall allow users to register "virtual" agents that represent discovered machines with no agent installed.
FR18	2	DeadDrop shall allow users to unregister virtual agents.
FR19	3	DeadDrop shall allow users to display and filter logs without requiring the logs to be exported.
FR20	3	When presenting well-formatted responses from agents, DeadDrop may automatically visualize the response as a graph, chart, tree, or other visual listing.
FR21	3	DeadDrop agents shall automatically discover accessible devices from the current device.
FR22	3	DeadDrop shall allow users to open a live terminal session with an agent using a standard communication protocol.
FR23	3	DeadDrop shall allow users to write reusable scripts to execute sets of shell commands on a particular agent.

Non-functional Requirements

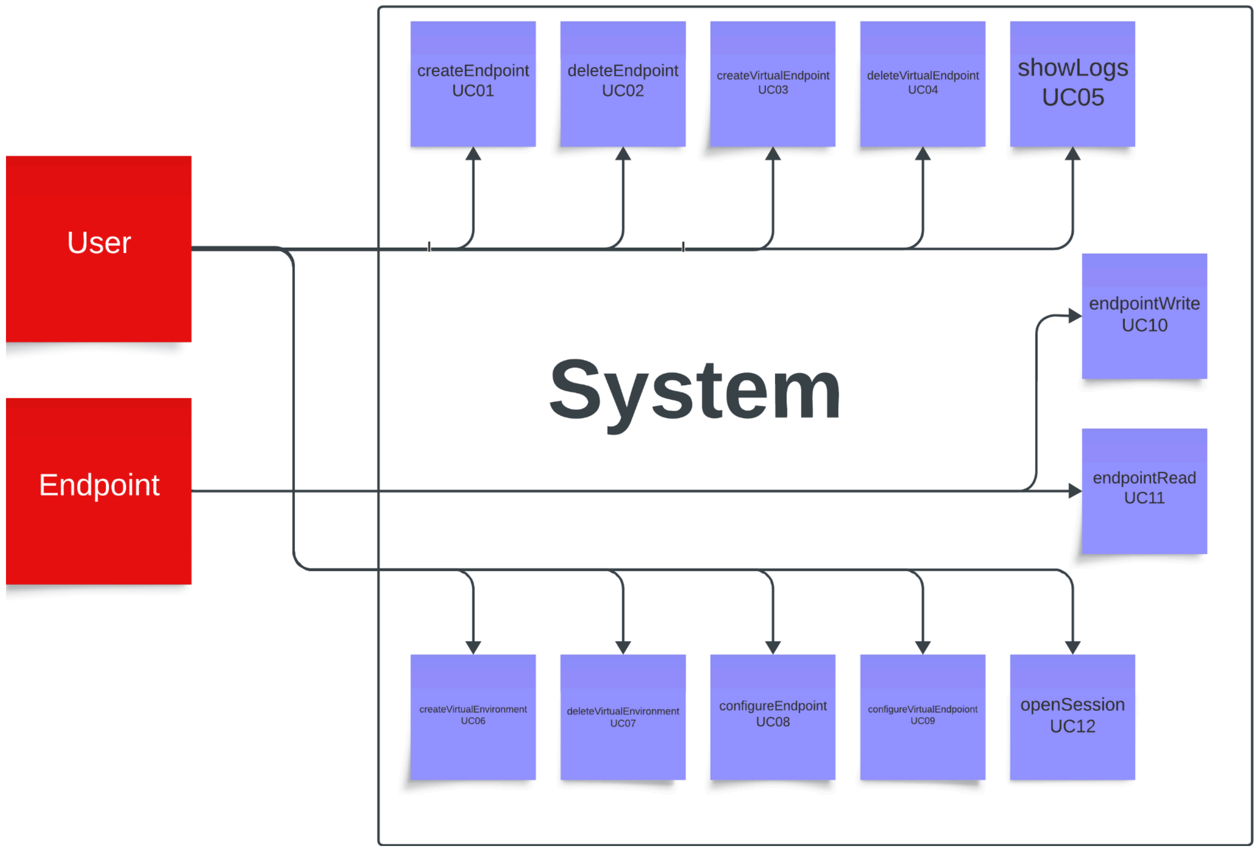
ID	L	Description
NFR01	1	DeadDrop must generate and store the information necessary to create a detailed penetration testing report.
NFR02	1	DeadDrop software written in Python shall adhere to PEP8 and must pass static type checks.
NFR03	1	Covert communication protocols shall be implemented independently of DeadDrop and must include a set of unit tests.
NFR04	1	DeadDrop agents should not perform irreversible actions on a device.
NFR05	1	DeadDrop shall implement at least one novel communication protocol.
NFR06	1	DeadDrop shall not violate terms of service on public websites.
NFR07	1	DeadDrop shall adhere to the virtues of ethical hacking.
NFR08	1	DeadDrop must be easy to troubleshoot and configure.
NFR09	2	DeadDrop shall be built in a platform-agnostic manner.
NFR10	2	DeadDrop agents shall not require internet connectivity to function.

ID	L	Description
NFR11	2	DeadDrop should be easy to use for users with experience in existing major C2 frameworks.
NFR12	2	DeadDrop should expose an API that allows users to easily automate functionality normally available through the GUI.

IV. Use Case Modeling

The terms “agent” and “server” are interchangeable with “endpoint” throughout this section – that is to say, an endpoint may be either the C2 server or a particular C2 agent.

Use Case Diagram



Detailed Use Cases

An overview of each of the user cases defined in the *Use Case Diagram* section is given below.

ID	Use Case	Description
UC01	createEndpoint	To generate a new agent, the user may command the server to compile or generate a new agent. The agent should be uniquely identifiable and should register with the server upon installation.
UC02	deleteEndpoint	The user may destroy an existing, registered agent. The agent should uninstall itself, and any dependent agents, from their machines.
UC03	createVirtualEndpoint	Either the user or an automated process may register a “virtual” agent, which is a known, accessible machine

		that does not have an agent installed.
UC04	deleteVirtualEndpoint	The user may delete any registered "virtual" agents, effectively removing these machines from view. They may be rediscovered if necessary.
UC05	showLogs	The user should be able to view (and export) the logs that have been collected by the server.
UC06	createVirtualEnvironment	The user should be able to create virtual network segments (environments) for the framework to operate in. These can be used to group controlled devices, restrict framework activity, or enforce agreements between the client and the testers.
UC07	deleteVirtualEnvironment	The user should be able to delete virtual network segments, effectively removing restrictions on the "real" network.
UC08	configureEndpoint	The user or server may request that an agent modify its behavior, such as by changing the communication secret key, the protocol used, its tags, or the name it is internally identified by.
UC09	configureVirtualEndpoint	The user or server may modify "virtual" agents in a similar manner to the above. Since these are loosely defined, this can encompass anything from changing user-written notes to changing their connections in a graph view.
UC10	endpointWrite	The server or agent may interact with each other over a specified communication protocol. Usually, this involves placing the command at the agreed-upon dead drop on the legitimate service, at which point it is up to the recipient to retrieve it.
UC11	endpointRead	The server or agent may retrieve (or simply check) new messages placed by the sender over a specified communication protocol. It is up to the recipient to decrypt and decode the message as needed.
UC12	openSession	The currently logged-in user to the C2 server may change, with all logging functionality and permitted actions changing accordingly.

Templated Use Cases

We have identified 4 use cases that are particularly important to DeadDrop's functionality, which have been explained in greater detail below.

Use Case: createEndpoint	
ID	UC01
Actor(s)	User, Server
Precondition(s)	1. A user must be logged into the C2 server 2. The user must have the permissions necessary to create new agents
Flow of Events	1. Server verifies that the currently logged-in user has the required permissions to create a new agent 2. User chooses agent template (which may only work on specific platforms, or require specific dependencies) 3. Server asks user to specify configuration variables for the agent 4. Server generates secret keys and identification variables for communication 5. Server compiles agent as standalone package and makes it available to the user to download via the Payloads interface
Postcondition(s)	1. A new C2 agent is generated and registered according to the user's configuration 2. The C2 server will now listen at the specified dead drop location for messages from the installed C2 agent

Use Case: endpointRead	
ID	UC11
Actor(s)	Time/User, Endpoint (either the server or a single agent)
Precondition(s)	1. The endpoint must be configured to communicate over a particular communication protocol, including the secret key for communication 2. One of the following: a. Sufficient time has passed for the endpoint to check the dead drop again (in which case time is an actor) b. The user explicitly requests an on-demand read of the endpoint (in which case the user is an actor)
Flow of Events	1. Endpoint checks specified dead drop location for new data 2. Endpoint verifies metadata a. If the message is a multipart message, retrieves all message pieces based on metadata 3. Endpoint uses the shared secret key to decrypt the message 4. Endpoint converts the message to the standardized DeadDrop message format before passing it to an internal message handler

	5. Message handler interprets standardized message and executes commands accordingly
Postcondition(s)	1. Upon reading a message, one of three things can happen: a. A new task is initiated for the agent; upon completion, the results are transmitted (using endpointWrite) b. New configuration settings are applied to the endpoint c. Nothing (such as when the server is simply checking that the agent is still alive)

Use Case: endpointWrite	
ID	UC10
Actor(s)	Time/User, endpoint (either the server or a single agent)
Precondition(s)	1. The endpoint must be configured to communicate over a particular communication protocol, including the secret key for communication 2. One of the following: a. Sufficient time has passed for an agent to announce to the server that it is still alive - a "heartbeat" (in which case time is an actor b. The user explicitly executes an on-demand command that requires communication with an endpoint
Flow of Events	1. Endpoint retrieves stored protocol configuration data 2. Endpoint forms message according to the DeadDrop standard message format 3. Endpoint encrypts message using sharedsecret key 4. Endpoint encodes message for the target dead drop platform (e.g., a video for YouTube, an image for Instagram, a comment for Reddit) 5. Endpoint generates requests to the target service to place encoded message at agreed dead drop location
Postcondition(s)	1. Message is now ready for receiving endpoint to view and decode (as per endpointRead)

Use Case: showLogs	
ID	UC05
Actor(s)	User, Server
Precondition(s)	1. User is logged into C2 server 2. User has permission to view server logs (especially if the logs may have sensitive user information)

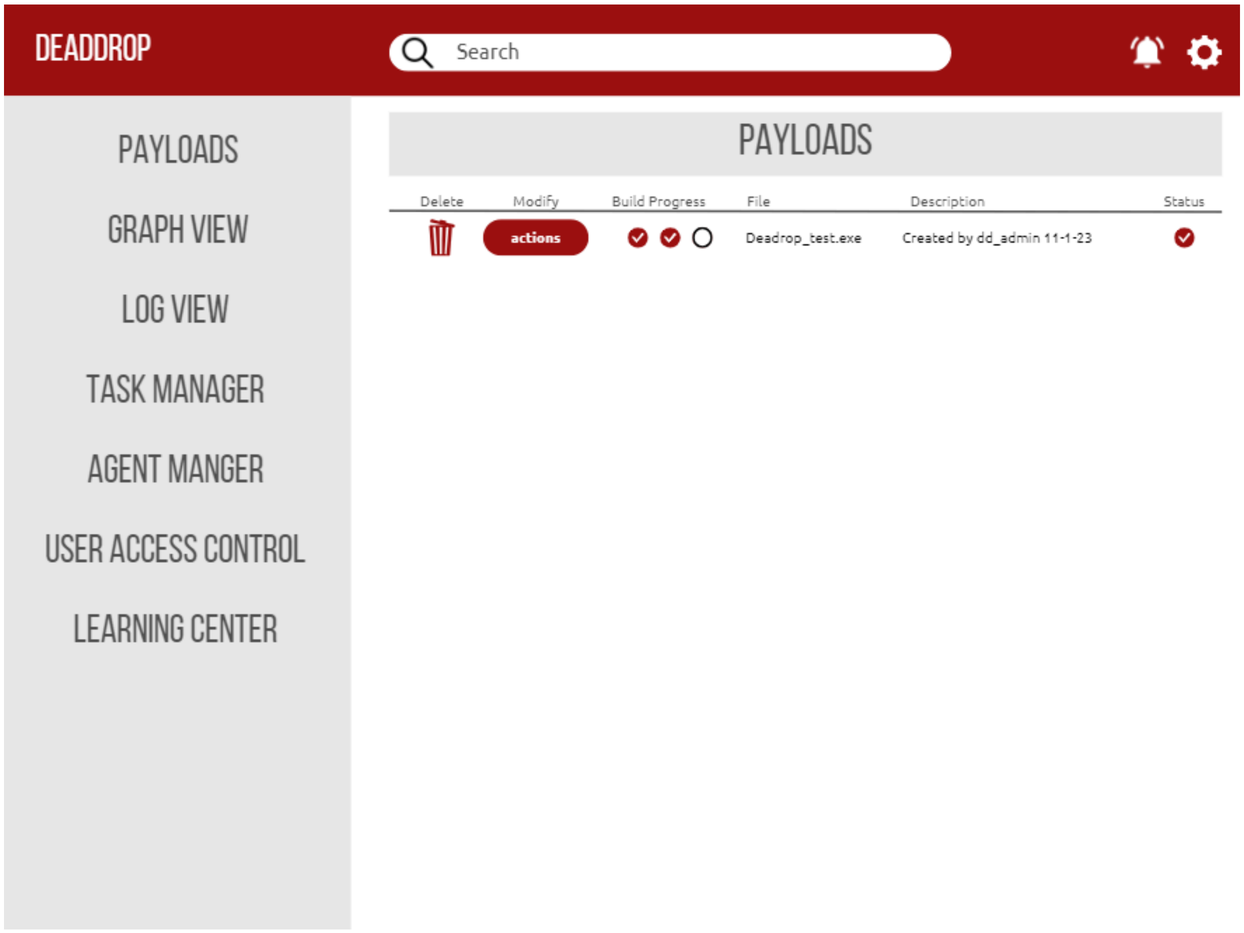
Flow of Events	1. User issues request for logs, possibly with filtering and configuration options 2. Server compiles requested log sources (if multiple) into one set 3. Server applies filtering options to log items 4. Server formats remaining log items for viewing according to the configuration options 5. Server presents log results through GUI
Postcondition(s)	1. Logs are now viewable and downloadable in the requested format

V. Requirement Traceability

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12
FR1												🟥
FR2	🟥									🟥	🟥	
FR3		🟥										
FR4										🟥	🟥	
FR5										🟥	🟥	
FR6	🟥							🟥		🟥	🟥	
FR7										🟥	🟥	
FR8			🟥			🟥	🟥		🟥			
FR9			🟥			🟥	🟥		🟥			
FR10	🟥									🟥	🟥	
FR11								🟥				
FR12					🟥					🟥	🟥	🟥
FR13					🟥					🟥	🟥	🟥
FR14					🟥							
FR15	🟥									🟥	🟥	
FR16												🟥
FR17			🟥						🟥			
FR18				🟥								
FR19					🟥							
FR20										🟥	🟥	
FR21	🟥											
FR22	🟥									🟥	🟥	
FR23	🟥									🟥	🟥	

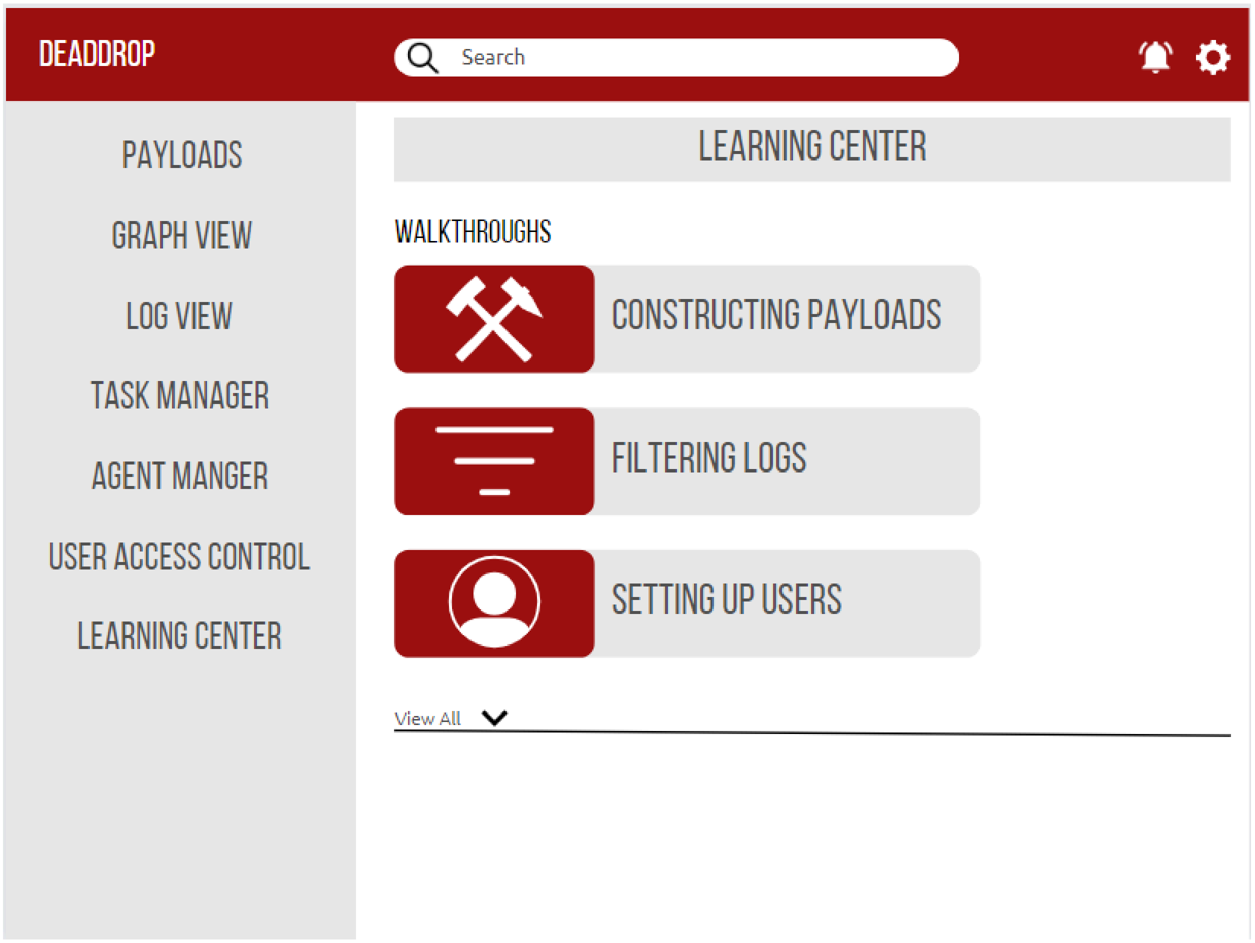
VI. UI Snapshots

Here, we present five snapshots of DeadDrop's server UI through which penetration testers can interact with the framework. Each of these provides some way to interact with the required features of DeadDrop as described in the *Technical Requirements* section.

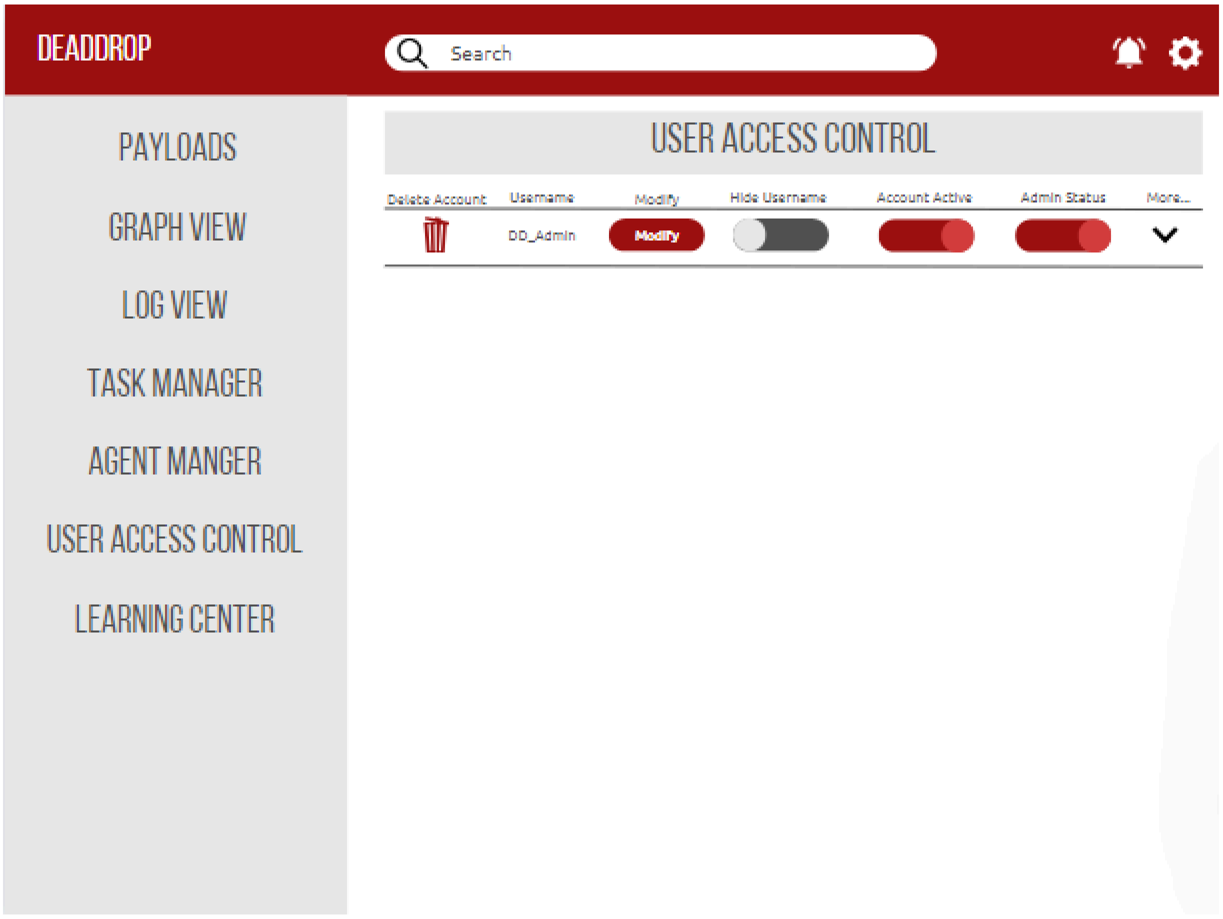


Delete	Modify	Build Progress	File	Description	Status
🗑	actions	✓ ✓ ○	Deadrop_test.exe	Created by dd_admin 11-1-23	✓

DeadDrop payload management: Users will be able to generate and download new agents (or payloads that install agents) through this interface. Where applicable, users will also be able to view build logs for the agent and reconfigure agent build variables as needed.

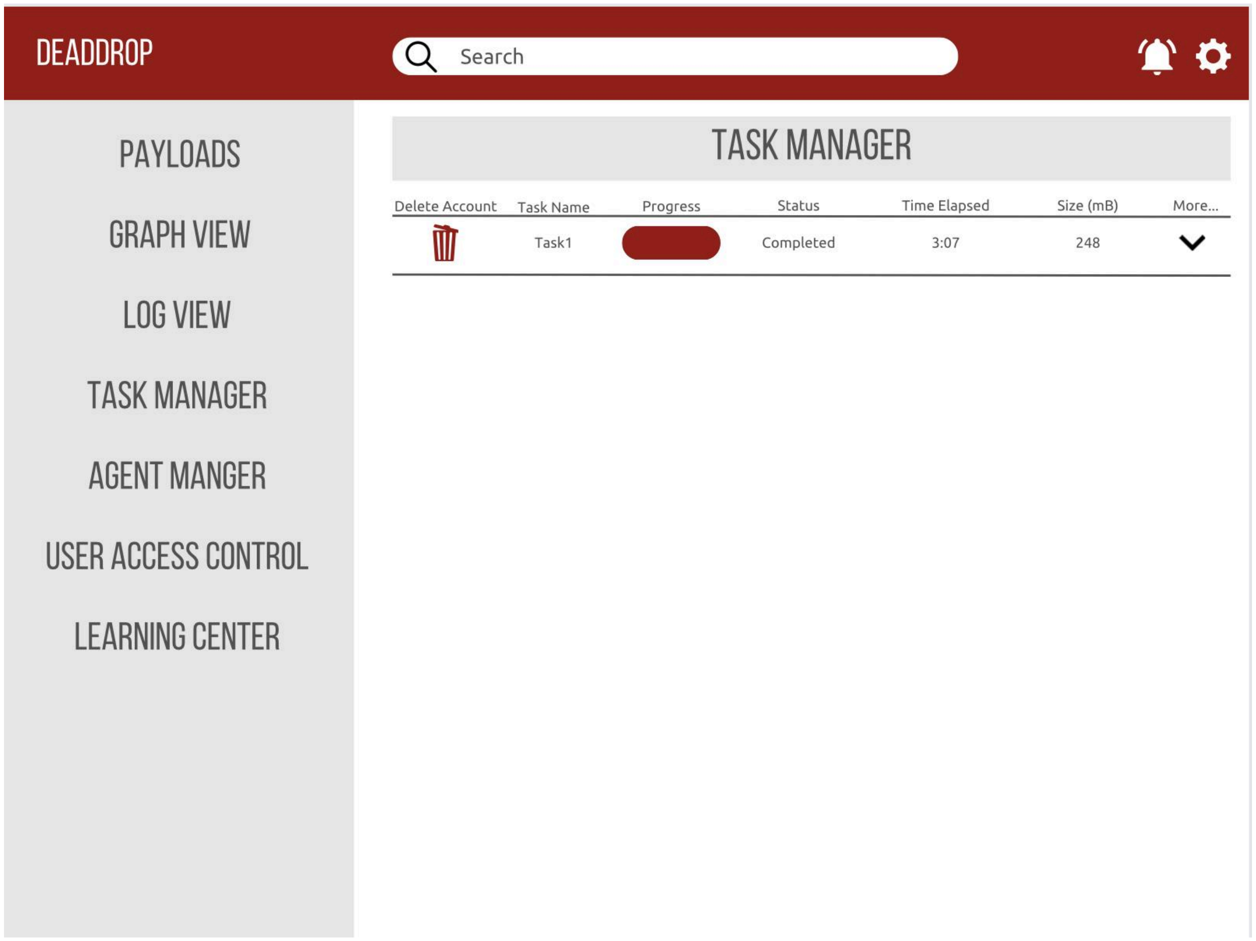


DeadDrop tutorials: As part of DeadDrop's goal of being easy to use, DeadDrop will provide users with training content for features that greatly aid the process of penetration testing.



Delete Account	Username	Modify	Hide Username	Account Active	Admin Status	More...
	DD_Admin	Modify				

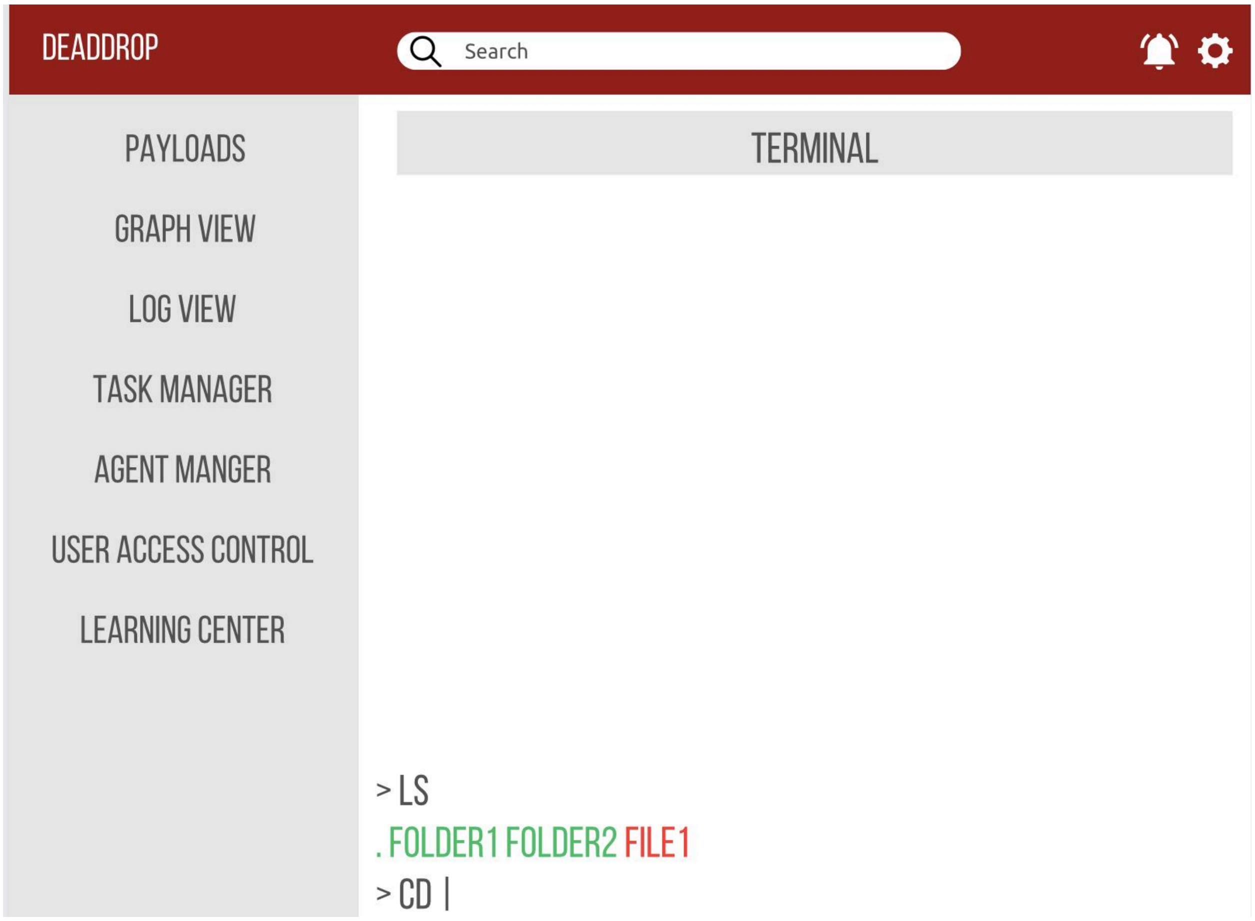
User access controls: Because penetration testing is often a collaborative process, DeadDrop will support a user authentication system. However, not all users should have the power to do everything, as certain actions may involve sensitive user data; thus, DeadDrop provides a user access control system.



TASK MANAGER

Delete Account	Task Name	Progress	Status	Time Elapsed	Size (mB)	More...
	Task1		Completed	3:07	248	

Task manager: Agents can be assigned "tasks", which are long-running commands that do not require instant responses. In most cases, these are scripts that perform common but complex tasks, such as dumping all the credentials in the volatile memory of a particular device. Users can view past, running, and scheduled tasks from this view.



```
> LS
. FOLDER1 FOLDER2 FILE1
> CD |
```

Agent terminals: In addition to running preconfigured commands on hosts, agents will also support users directly running shell commands through the DeadDrop interface. Since this is abstract of any communication protocol, providing a terminal frontend through the GUI allows the server to mediate complex protocols where needed.

VII. Glossary

Terms that are referenced elsewhere in the glossary are ***bolded and italicized*.**

#	Term	Definition
1	advanced persistent threat	Refers to well-funded attackers (threat actors), often nationally-backed groups, that engage in stealthy, complex attacks. They may be undetected for a long period of time, and often target national industries. Frequently referred to as an APT.
2	antivirus evasion	Any malware mechanism that allows the malware to go undetected by antivirus solutions such as Windows Defender. This is often achieved through ***polymorphic code***.
3	attack surface	Refers to all of the possible points where an attacker can attempt to ***exploit*** a potential ***vulnerability***. For many organizations, this can include the company website, any public-facing servers, and individual user machines.
4	beacon(ing)	Commonly refers to the process of a ***C2 agent*** communicating with its associated ***C2 server***. In a strict sense, it refers to the agent periodically reaching out to the server for new commands.
5	black box testing	Refers to penetration tests in which the attackers have virtually no knowledge about the organization or do not have any existing entry points into the organization. This better simulates real-world attacks, but makes it more difficult to discover internal vulnerabilities.
6	blue team	Collective term used to refer to cybersecurity professionals that defend a (particular organization's) network. Contrasts with the ***red team***.
7	C2 agent	A piece of software (***malware***) installed on a device that allows an attacker to remotely control that device as part of a ***C2 framework*.** It may execute arbitrary commands and forward messages to other agents installed in a network.
8	C2 framework	Software used to administer and organize penetration tests (and illegal cyber attacks). Comprised of a ***server*** hosted by the attackers and one or more ***agents*** planted on target devices, such that the server communicates in a standardized manner with the agents.
9	C2 server	Software used by an attacker as part of a ***C2 framework*** to administer and organize a penetration test. It is primarily used to compile, distribute, and communicate with ***agents*** installed on remote devices.
10	Common Vulnerabilities and Exposures (CVE)	A free, public ***vulnerability*** database often used by organizations to track and address vulnerabilities in software used by the organization. Integrates with many other platforms and resources for

#	Term	Definition
		testing older platforms and identifying vulnerable software.
11	Cyber Kill Chain	Attack framework developed by Lockheed Martin that describes the typical "life cycle" of an attack, detailing the steps that attackers complete to ***exfiltrate*** data out of a network.
12	domain	In this context, refers to a publicly accessible network that is identified by a unique name. Attackers will typically set up new domains to host a ***C2 server***; traffic to the new domain may be suspicious due to the longevity of most legitimate domains.
13	endpoint security	Software that contributes to the security of individual devices on an organization's network. Common endpoint security solutions include antivirus (Microsoft Defender), live threat analysis (CrowdStrike), and device log collection (osquery).
14	exfiltration	The act of taking sensitive information outside of an organization's network (where it is intended to be kept secure). The goal of most cyber attacks is to exfiltrate valuable data and either sell it or hold it for ransom.
15	(to) exploit	The act of abusing a ***vulnerability*** (flaw) in software to cause it to behave in an unintended manner. Can be used to install malware on a vulnerable device.
16	intrusion detection system	Refers to any hardware or software that monitors a network for malicious activity (usually indicating the presence of an attacker within a network).
17	malware	Malicious code or software that executes on a computer typically without the user's knowledge, performing arbitrary actions as designed by the attacker. Typically, malware is delivered by ***exploiting*** a ***vulnerability*** in one or more systems. In this document, malware is generally interchangeable with ***C2 agent***.
18	MITRE ATT&CK	A publicly available set of models that detail common attack techniques used by ***threat actors*** that have been observed "in the wild". Sponsored by the MITRE Corporation.
19	payload	Broadly refers to any malicious code that executes a specific action on a target system; payloads may be delivered through a variety of means (email, SSH, etc.), and may take a variety of forms (a Word doc, an executable, a shell script, etc.). Common payloads allow attackers to run arbitrary commands on a target or install malware.
20	persistence	The quality of a running program (typically malware) to survive a

#	Term	Definition
		restart of the machine. Typically, these programs leverage operating system features to allow the program to be run again on startup. In the case of malware, this also implies that it can survive a restart of the machine without detection.
21	polymorphic code	Any code (or software) that is capable of changing the instructions that it executes without changing its functionality. By doing so, the software is able to obfuscate malicious code that might already be known to an antivirus solution, contributing to ***antivirus evasion***. Sometimes achieved through decrypting instructions at runtime.
22	red team	Collective term used to refer to cybersecurity professionals associated with attacking a network and its services and identifying gaps in an organization's security. Contrasts with the ***blue team***.
23	threat actor	Refers to any individual or group that engages in malicious activity (and represents a risk to a particular organization). Threat actors vary greatly in skill and funding.
24	traffic mirroring	The act of inspecting, copying, storing all (unencrypted) traffic that travels in and out of an organization's network. Often used in both forensic investigations (i.e. after an attack has occurred) and proactive identification of suspicious activity.
25	vulnerability	Commonly used to refer to any flaw in a piece of software that can be used to cause the software to behave in an unintended manner. May also refer to any general weakness in computer systems, procedures, or configurations that could be used for malicious gain.
26	white box testing	Refers to penetration tests in which the testers have complete access to and knowledge of an organization's software and network. Typically, this includes network diagrams and application source code. These are more likely to discover internal vulnerabilities, but often do not accurately represent an external attacker's knowledge.

VIII. Reference Material

Problem Domain Book

Learn Penetration Testing is an introductory book to both the technical and non-technical principles of penetration testing. It covers not only the specific techniques that attackers use to identify and exploit vulnerabilities, but also the overall "life cycle" of an attack. This is particularly important; although the novelty of this project comes from its technical implementation of a covert communication protocol, the project as a whole is only useful if it can be used to orchestrate an attack from start to finish. This book allows us to identify specific requirements in functionality and reporting.

Reference Articles

Each of the publications listed below details penetration testing at the organizational level; that is, rather than focusing on specific technical approaches, these articles (primarily) detail the general procedures and ethics that teams follow when conducting penetration tests in a variety of different environments.

- *An overview of vulnerability assessment and penetration testing techniques* discusses trends in vulnerability discovery, particularly with respect to the Common Vulnerabilities and Exposure List, a public catalog of vulnerabilities administered by the MITRE Corporation. It also discusses attack frameworks used in penetration tests against specific industries, describing the risks of using these attack models and the precautions that must be taken to execute them.
- *A study on penetration testing process and tools* details the organizational outcomes of penetration testing, as well as specific parts of an organization that can be tested. (For example - testing an organization's internal network is different from testing its physical security or its public-facing applications.) It also covers certain non-technical techniques, such as social engineering. It also briefly mentions different "scopes" of testing, in which the testers have a varying amount of information about the services and network of an organization when engaging in testing.
- *MTC2: A command and control framework for moving target defense and cyber resilience* details an interesting deviation from classical C2 frameworks, in which the behavior of the framework is dictated primarily by software developers. In contrast, MTC2 is more policy-oriented and decentralized, in such a way that it avoids a single point of failure and provides inherent "guardrails" to the penetration testers by abstracting away certain parts of the framework.

Additional Resources

These resources primarily focus on penetration testing at the technical level, providing information on implementing specific attack techniques and detailing the infrastructure used in existing large C2s.

- The Mythic C2 documentation details payload and agent development for the Mythic C2 framework, which details Mythic's requirements for what new agents must implement. This is particularly helpful, as it helps inform us on what data is needed for communication between the server and multiple agents, as well as what their message format looks like. It also contains an extensive amount of general documentation on Mythic's architecture, components, and configuration.

- A list of various open-source C2 frameworks is available on GitHub, ranging from fully-fledged frameworks (like Mythic) to proofs of concept (usually just two scripts, one for the server and another for the agent). This allows us to evaluate what functionality is common among many different frameworks (and therefore desirable), as well as how other well-designed frameworks accept new functionality.
- The Sliver C2 documentation details both the architecture of Sliver as well as various other resources related to implementing specific penetration testing techniques (such as antivirus evasion) that are not covered in the Mythic documentation. This includes links to the Veil Framework and PEzor, both of which are automated solutions to developing payloads that evade common antivirus solutions. Existing integrations between Sliver and other external libraries could reasonably be adapted to DeadDrop to extend its functionality.
- A list of resources for implementing malware persistence (that is, malware that survives a restart of the computer) and detecting and analyzing persistent malware techniques, is available on GitHub, along with a general overview of how persistence techniques work. This can be particularly helpful in implementing our C2 agents in a "real world" context; although VMs in the cloud might not restart often, real user machines do. These resources allow us to explore existing industry techniques for developing stealthy malware.

IX. Time Worked on Document

Note that the recorded time includes time spent writing and formatting these sections in addition to learning about cybersecurity topics related to the project. It also includes any group discussions, time spent engaging in stakeholder interviews, and overhead associated with project management.

Team Member	Hours	Sections Contributed
Jann Arellano	7.0	- UI Snapshots (*includes design discussions and code prototyping*)
Brian Buslon	7.0	- UI Snapshots (*includes design discussions and code prototyping*)
Keaton Clark	6.0	- Technical Requirements - Use Case Modeling - Requirement Traceability
Lloyd Gonzales	10.0	- Introduction - Stakeholder Interviews - Technical Requirements - Glossary - Reference Material