DeadDrop

a clever command and control framework for penetration testers

Revised Concept and Project Management (P1)

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

Advisor:
Shamik Sengupta
Professor
University of Nevada, Reno

CS 426, Spring 2024
Instructors: David Feil-Seifer, Devrin Lee, Sara Davis

Department of Computer Science and Engineering
University of Nevada, Reno
February 2, 2024

Table of Contents

I. Abstract

We describe the continued implementation of DeadDrop, 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. 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. Our work for this semester encompasses the design and implementation of DeadDrop's server and web interface, a containerized agent, and video-based covert communication protocols. This document outlines the design, rationale, and significance of this project, as well as scheduled work and progress since the fall prototype demo.

II. Project Description

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.

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. That is, the use of C2 frameworks is critical to penetration testers and security engineers alike.

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. Messages ("dead drops") can be placed on external services by one device, which can be retrieved by the other device by accessing the external service. By abusing the "trust" behind large, well-known websites, 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 attackers.

Like many C2 frameworks, DeadDrop provides the following major features through its web interface and internal APIs:

- *Payload generation:* Users can build and configure new instances of agents, leveraging containers to allow platform-agnostic building.
- *Messaging:* Users can communicate with agents to execute commands on remote devices and receive results.

- *Reporting and administration:* The server provides permissions systems and logging for developing organizational reports and ensuring accountability.

Each of the three high-level modules of the project (as described in Project Responsibilities) leverages distinct technologies:

- The **server** is composed of a Svelte frontend and a Django backend, which uses Celery for asynchronous task management, Docker for payload generation, and several libraries for building the API endpoints needed for Svelte and Django to communicate.
- The **agent** is a containerized Python program with numerous libraries for task management, message encryption and signing, and logging. It uses PyInstaller to bundle itself as an executable on specific platforms.
- Both the server and the agent incorporate one or more **communication protocols**, which are implemented in Rust and are compiled as shared libraries.

Note that agents and protocols may be implemented in any language or platform, so long as they adhere to the standard internal messaging formats. Since an arbitrary number of agents and protocols may be developed for DeadDrop (i.e. it is infinitely extensible), the agent and protocols described above reflect short-term implementations.

DeadDrop's offensive nature makes security and safety paramount to its design. As described by one of our stakeholders, DeadDrop is like a gun; in addition to being handled only by qualified professionals, it must incorporate features that prevent misuse of the framework and limit the impact of mistakes. To achieve accountability, DeadDrop uses Django's built-in authentication to provide users with fine-grained permissions and generate logs for actions throughout the framework. Furthermore, certain planned features reduce the likelihood of disruption to an organization, including a "guardrail" system that prevents payloads from being executed on systems that do not meet an IP address or hostname whitelist. Individual communication protocols abstract the mechanisms used to achieve message reliability, such as timeouts and message reordering.

In short, DeadDrop provides organizations with the tools needed to develop and test their defenses against these techniques through a highly configurable and extensible framework.

III. Significance

DeadDrop is a valuable project from not only an industry and research standpoint, but also in our own professional growth. Through its use of covert communication protocols over trusted services, it directly attacks existing industry defenses in a manner that (to the team's knowledge) has not been implemented in a framework suitable for large-scale penetration testing. Although not all of us are cybersecurity-focused students, we found this an exciting opportunity to explore "hacking" in a controlled environment. Furthermore, we believed that this could lay the groundwork for future graduate research at UNR, whether undertaken by our team or passed on to another student.

From a project management perspective, we found that we could split the project in a way that allowed each of us to work on roles directly related to our career goals. Although the final product requires skilled cybersecurity professionals to operate, its implementation still requires web designers, low-level programmers, and full-stack developers - all roles we could assign to our team. In turn, despite only having one cybersecurity-focused member, all of us could significantly contribute to the project and develop our own interests.

The innovative characteristics of DeadDrop can best be described through an analysis of existing C2 frameworks. Products such as Cobalt Strike [1] and Mythic [2] serve the role of mature C2 frameworks that are actively used in industry to conduct penetration tests, while proof of concepts such as Gcat [3] and Gdog [4] leverage trusted services such as Gmail for stealthy communication. However, no project has combined both with the goal of creating a robust framework with a focus on covert communication streams. While products such as Cobalt Strike employ stealth at the "application" level – such as through self-encrypting malware – no product has truly explored stealth at the "network" level, avoiding detection as malicious traffic passes over a network.

In particular, the use of YouTube, Zoom, and Slack as a method of covert communication is highly innovative; many network-based defenses, such as intrusion detection systems and application firewalls, depend on their ability to identify malicious strings in plaintext. This is much more difficult when a video streaming protocol is used, as only human observation of the underlying video would reveal anything unusual about the video itself.

As a result, DeadDrop possesses significant market potential and offers substantial open-source value. It aims to offer all of the features of major commercial and open-source C2 frameworks while introducing a unique approach to evading detection. As part of our mission to provide organizations with the tools needed to defend against these attacks, DeadDrop is a permissively licensed open-source project. Furthermore, our modular architecture, platform agnostic code, and documentation promote the rapid development of new agents and protocols tailored toward novel techniques well beyond the scope of a capstone project. Our existing code can be freely forked to develop an ecosystem of interchangeable packages (and even new servers) through the use of standardized interfaces with well-documented messaging formats.

Though DeadDrop has a limited environmental impact, we believe that the novel (and perhaps "flashy") nature of our project will raise awareness about modern security threats, allowing organizations to improve their cybersecurity practices and better protect the data of users.

IV. Legal and Ethical Aspects

The inherent nature of DeadDrop, as both a free and open-source command and control framework intended for penetration testing, raises a host of legal and ethical concerns regarding the potential abuses of its functionality. In particular, there are many concerns associated with simply testing and developing our communication protocols, even before their use in the general public.

The protocols being developed for DeadDrop are intended to utilize large, whitelisted platforms such as Twitter or YouTube to bypass security measures and communicate covertly between the server and infected devices. In turn, developing and testing these communication protocols involves using these platforms for unintended purposes, likely skirting or violating their terms of service. Simultaneously, another major concern is that the free and open-source licensing of DeadDrop makes the source code readily available to both well-intentioned and malicious actors. It is entirely possible for a malicious user to utilize or build off DeadDrop's code to exploit vulnerabilities in a system.

To mitigate the risks of utilizing platforms such as Twitter or YouTube in an unintended manner, as much of the development process as possible is conducted through fake services to simulate larger platforms. "Real" testing is conducted in a discrete manner through alternate accounts and a small number of devices set up in a post-infiltration scenario, where the agent is already installed on the device. Addressing the potential for misuse, DeadDrop agents are constructed in an open manner, meaning that agents deployed by DeadDrop are not obfuscated and leave behind logs of all activities undertaken. That is, DeadDrop is aimed at evading network-level defenses, not evading endpoint defenses or causing destruction; agents can be easily found and reverse-engineered on devices.

Furthermore, the repository doesn't come with any tools for infiltration – only the ability to execute commands from the server on an already infiltrated device. This leaves out the possibility for any "out-of-the-box" exploitation, increasing the difficulty of misusing the framework.

V. Changes and Progress Since Initial Project Concept

Since the project concept was introduced in October 2023, we have built a complete proof of concept in which arbitrary two-way remote communication can be achieved over YouTube. To be specific, it is possible to issue an arbitrary command from the web interface, insert it into a standardized JSON document, encode the document into a video, and upload this video to YouTube. The agent can then check the agreed-upon YouTube channel for new videos (signifying a new message from the server), download the video, decode the document and command, and then execute the command asynchronously. The same flow is then used by the agent to generate responses to the server.

This demonstrates that the workflow is not only possible, but that it can be implemented in full. No major changes in the design of this workflow have occurred since October 2023, and our future tasks primarily involve making DeadDrop more usable as an industry product (as opposed to a series of proofs of concepts).

However, various planned changes have been made to the original design of the project as a whole; while this includes the addition of various features relevant to the industry (based on the feedback of our stakeholders), we have also made minor architectural changes. For example, we are currently considering a rework of how we manage the individual dependencies of agents and communication protocols, which are wholly independent of the server; while we previously planned on using a Git-oriented approach, we are now considering implementing our own package manager that handles dependencies without relying on Git. This is the result of research and analysis into other how other open-source C2 frameworks deal with interchangeable agents, such as Mythic.

In terms of implementation milestones, we have successfully implemented the server-side backend, leveraging Django Rest Framework to expose DeadDrop's internal API to the frontend. We also integrated plugins for stability and portability; an example is Django CORS Headers, which simplifies communication between the frontend and backend by enabling trusted host settings. Another notable addition is Zod integration, a robust schema validation tool that improves data input on Svelte Superforms, ensuring safer and more reliable code.

The successful integration of our first Superform was a major milestone, demonstrating that our design for the frontend and backend was viable despite the challenges we encountered. Additionally, we have completed implementing our interfaces between the frontend and backend, allowing the frontend team to start implementation on rendering the data provided by the backend APIs.

VI. Project Responsibilities

The project can broadly be split into three modular segments: a server, one or more agents, and one or more communication protocols. At the highest level, DeadDrop follows a client-server model, whereby a single attacker-controlled server interacts with one or more agents (clients) installed on remote machines via an arbitrary protocol.

The assignments for each component (and any subcomponents) are summarized below:

- C2 Server
    - Framework interface
        - Frontend: **Brian Buslon**
        - Backend: **Jann Arellano**
    - Task management: **Lloyd Gonzales**
    - Messaging: **Keaton Clark**
    - Payload generation: **Jann Arellano**
- C2 Agent(s): **Lloyd Gonzales**
- Communication protocols: **Keaton Clark**

The server provides a *team* of attackers with a user-friendly interface to interact with compromised devices, generate new payloads (instances of agents), and log all actions taken. The server can be further divided into four components, each assigned to someone responsible for its research, design, and implementation:

- The *framework interface*, which validates user actions, stores operational logs, and provides a web interface for penetration testers. This is composed of a Django backend (**Brian Buslon**) and a Svelte-managed "frontend" (**Jann Arellano)**, which communicates with Django via a RESTful API.
    - **Brian Buslon** is responsible for the Svelte frontend (both its visual design and implementation) due to his graphic design and web development experience.
    - **Jann Arellano** is responsible for the Django backend due to his full-stack development and data analytics experience.
- The *task management module*, which asynchronously executes tasks initiated from the framework interface. This is responsible for initiating tasks in the messaging and payload construction modules, as well as managing bookkeeping and callback actions on their completion or failure. **Lloyd Gonzales** is responsible due to prior Django design experience.
- The *messaging module*, which handles communication with agents over arbitrary protocols. This includes both message generation and message fetching, retrieving and sending messages over a particular protocol. **Keaton Clark** is responsible due to his role in developing communication protocols as described below.
- The *payload construction module*, which generates new agent instances that can be installed on a target device. This is achieved through one or more Docker containers that contain the necessary dependencies to build and bundle an instance of an agent with user-specified

configurations. **Jann Arellano** is responsible due to his interest in containerized applications.

Each agent provides the attacker with the means to interact with the computer the agent/malware is installed on. The agent may range in complexity from a simple Python or Powershell script to a feature-filled application capable of dumping credentials from memory, stealing session cookies, and opening a reverse shell to the machine. Agents are very abstract in form, and may be implemented in arbitrary languages for arbitrary systems; the only underlying requirement is that they respond to standardized DeadDrop messages issued by the server. **Lloyd Gonzales** is responsible for the implementation of agents due to his experience in malware research and general cybersecurity knowledge.

Finally, the protocols are standalone binaries included with the server and agents that allow binary messages to be encoded, passed through a trusted service such as YouTube, Zoom, or Slack, and then decoded by a receiver. Each protocol defines its own mechanism for message reassembly, acknowledging messages, and so on. This forms the heart of DeadDrop - each protocol makes it possible to sidestep network defenses that identify malicious traffic based on domain or the underlying “contents” of the message. **Keaton Clark** is responsible for the research and development of each protocol, in large part due to his low-level programming and hardware experience.

VII. Project Monitoring and Risks

Project Management

Our primary platform for project management is GitHub, which provides a variety of features for creating tasks, assigning issues, and determining the status of the project relative to the end of the semester. Currently, we have a large backlog of features to implement for each component of the project. Each feature/topic is expressed as a distinct Github issue, with due dates and expected effort assigned to each issue.

Progress towards each issue is measured and implemented through commits against a feature branch, such that each feature branch is (ideally) scoped to exactly one issue. Issues are considered complete when a pull request covering the issue has been created and all members of the group have approved the changes. The issue is transitioned to a finished state and updated in GitHub accordingly. These progress-tracking practices will ensure the project is completed on time, mitigating the likelihood and impact of **RP 07 –** the general risk of time-related issues described in the risk register in Figure 1 below. The use of GitHub for project management makes it easy to identify where extra support or effort is needed and act accordingly.

Risk Id	Risks	Current Risk			Status	Owner	Raised	Mitigation Strategies	Residual Risk		
		Likelyhood	Impact	Severity					Likelyhood	Impact	Severity
RP 01	Violating TOS of platforms	★★★★★	★★☆☆☆	10	Open	Keaton Clark	10/16/2023	- Read into terms of service before implementing a protocol - Create mock websites when necessary	★★★★☆	★★☆☆☆	8
RP 02	Ensuring all actions on both the server side and target are thouroughly logged	★★☆☆☆	★★★★★	10	Open	Lloyd Gonzales	10/16/2023	- Impliment logging policies in a useful format	★☆☆☆☆	★★★★★	5
RP 03	Properly containing agents being used in testing	★★☆☆☆	★★★★☆	8	Open	Lloyd Gonzales	11/5/2023	- Use containerization - Restrict agents access to the internet	★☆☆☆☆	★★★☆☆	3
RP 04	A third party misusing our project	★☆☆☆☆	★★★☆☆	3	Open	Jann Arellano	1/27/2024	- Due to the open source nature of the project there is not much that can be done besides keeping it open source	★☆☆☆☆	★★★☆☆	3
RP 05	Agents performing actions they are not permitted to do	★★★☆☆	★★★☆☆	9	Open	Lloyd Gonzales	11/24/2023	- Thouroughly log actions to reduce impact of rouge agents - Multiple points of failure - All members review code	★★☆☆☆	★★☆☆☆	4
RP 06	University IP being blocked from providers such as YouTube	★☆☆☆☆	★★★★★	5	Open	Keaton Clark	12/15/2023	- Graduate before it happens - If we get TOSed, stop	★☆☆☆☆	★★★★★	5
RP 07	Time Constraint	★★★★★	★★★☆☆	15	Open	Jann Arellano	10/16/2023	- Adhere to timeline - Continually update timeline and expectations	★★★☆☆	★★★☆☆	9
RP 08	Proper Open Source license usage	★☆☆☆☆	★☆☆☆☆	1	Open	Brian Buslon	1/9/2024	- Educate ourselves on common open source licenses	☆☆☆☆☆	★☆☆☆☆	0

Fig. 1
Risk register outlining the identified risks of DeadDrop's implementation this semester.

All other risks fall under two main categories: risks associated with terms of services/licensing, and ethical risks associated with conducting penetration testing.

Terms of Service and Licensing

The most immediate risk to this project is the potential for violating the terms of service (TOS) of each platform we use for our communication protocols, – an unavoidable risk due to the nature of the project, as detailed in **RP 01**. Due to the complexity of these legal documents, it is infeasible for us to eliminate this risk; we simply do not have the legal resources or knowledge needed to vet our project on each platform we intend to use. However, we have several procedures in place to mitigate this risk, primarily by setting up mock websites that are nearly (or completely) equivalent to the true service the project is intended to be used with. Other procedures include limiting testing and manual review of the TOS, which aim to reduce our exposure to this risk.

Fortunately, the most likely outcome of this risk becoming an issue is an automated account ban. In contrast, **RP 06** describes the worst-case outcome we have considered as a result of repeated TOS violations, which is a blanket ban against the university's network from accessing the platforms we will be using. While highly unlikely, the extreme impact of this risk makes it important to consider. In the case of a true malicious actor, this is a non-issue; the risk of an IP ban pales in comparison to the risk of other actions likely taken by an attacker. While the comedic solution is to graduate before this risk manifests into an issue, the true solution is to either avoid using these services entirely or form an agreement with the associated vendor.

Finally, **RP 08** describes the improper use of open-source software by violating its associated licensing terms, such as requirements for attribution or restrictions on commercial usage. This is a low-impact issue that is unlikely to occur, given proper education and due diligence on our part.

Ethics and Best Practices for Penetration Tests

One of the largest concerns in penetration testing is the potential for taking down business-critical services; in turn, organizations and penetration testers agree on the "rules of engagement", which outline what is and what is not allowed. Despite this, mistakes still occur, and organizations need to know that penetration testers are aware of any remaining risks.

One potential risk is the inability to revert any changes made to machines as a result of penetration testing, typically due to a lack of logging. This is described by **RP 02**, which describes the impact of logging failures in a penetration test. Such failures can be catastrophic, especially if the organization lacks backups or any form of disaster recovery; in turn, we plan on implementing common policies that enforce logging throughout the project.

Agents must be contained in two different ways - they must not be allowed to p erform actions outside of those explicitly allowed, and they must not be installed on devices that are "out of scope" based on the rules of engagement. These are represented by **RP 05** and **RP 03** respectively, which address the issue of scoping. In order to mitigate this, we will test DeadDrop in containerized or sandboxed environments where possible and require team reviews of any changes made to agents. Furthermore, the "guardrail" system, which enforces a whitelist of the devices that agents can be run on, will further avoid any accidental execution of agents on unintended devices.

Finally, as with any open-source project, there exists the unfortunate reality that DeadDrop may be misused by a third party, as described by **RP 04**. We have no way to restrict the use of this project to legitimate penetration testers (and even paid products such as Cobalt Strike suffer from this); that said, the impact of any such use is inherently mitigated by the source being available. Legitimate security professionals can explicitly design defenses against DeadDrop-like attacks in their own environment before used by an attacker. Furthermore, certain components of DeadDrop are designed to be "attackable" by defenders if needed, such as the absence of obfuscation in compiled agents (unlike Cobalt Strike).

VIII. Accessibility

The primary concerns regarding accessibility are largely tied to the web interface of the C2 server; all other interactable components of the framework involve source code development or API calls. For components outside of the web interface, their accessibility is dictated largely by the user's choice of program for software development.

Developing with web accessibility in mind is largely dictated by the Web Content Accessibility Guidelines (WCAG), which express these guidelines as functional and non-functional requirements. Applicable functional requirements for DeadDrop include keyboard navigation for interactable components, text alternatives for non-text content (such as figures), and focus indicators for keyboard-focused components, which are essential to the "mechanical" elements of accessibility – that is, the website's support for interacting with page elements. Non-functional requirements for DeadDrop include rigorous browser compatibility, high color contrast, and responsive design, which contribute to a seamless experience across different users. These are typically higher-level requirements that are not implemented on a page-by-page basis, but rather across the website as a whole.

As with most modern JavaScript frameworks, SvelteKit (the framework in use for the frontend), can leverage Accessible Rich Internet Applications (ARIA) attributes to enhance the accessibility of dynamic content and UI components. SvelteKit's ARIA features, in addition to openly available prebuilt components, help maintain a high degree of accessibility throughout our development cycle. Furthermore, we intend to use openly available accessibility analyzers, such as SiteImprove and Google Lighthouse, to identify potential accessibility issues and track compliance with specific items of the WCAG standards.

IX. Time Worked

Note that the recorded time includes time spent performing research, writing code, and implementing the modules described in this document (since the fall prototype demo). This also includes meetings to discuss the architecture and design as a team, verifying that the systems are still reasonably designed and achievable.

Team Member	Hours	Sections Contributed

Jann Arellano	15	- Code (during Winter Break)
		- Document
		  - Significance
		  - Changes and Progress Since Initial Project Concept

Brian Buslon	4	- Document:
		  - Legal and Ethical Aspects
		  - Accessibility

Keaton Clark	5	- Project Website (https://unr-deaddrop.github.io/) - Document: - Changes and Progress Since Initial Project Concept

Lloyd Gonzales	9	- Document:
		  - Abstract
		  - Project Description
		  - Significance (and References)
		  - Project Responsibilities

X. References

[1] Fortra, “Features | Beacon, C2 Profiles, Attack Packages, and More,” Cobalt Strike. Accessed: Oct. 16, 2023. [Online]. Available: https://www.cobaltstrike.com/product/features
[2] Mythic Developers, “Mythic Documentation.” Accessed: Oct. 16, 2023. [Online]. Available: https://docs.mythic-c2.net/
[3] Marcello, “byt3bl33d3r/gcat.” Feb. 01, 2024. Accessed: Feb. 01, 2024. [Online]. Available: https://github.com/byt3bl33d3r/gcat
[4] maldevel, “maldevel/gdog.” Jan. 07, 2024. Accessed: Feb. 01, 2024. [Online]. Available: https://github.com/maldevel/gdog