

# DeadDrop

*a clever command and control framework for penetration testers*

## Acceptance Criteria and Testing Strategy and Plan (P3)

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  
March 4, 2024

## Table of Contents

<b>I. Abstract</b>	<b>3</b>
<b>II. Project Updates and Changes</b>	<b>3</b>
<b>III. User Stories and Acceptance Criteria</b>	<b>5</b>
<b>IV. Testing Workflow</b>	<b>7</b>
<b>V. Testing Strategy</b>	<b>11</b>
<b>VI. Unit Testing</b>	<b>17</b>
Han	17
Keaton	19
Brian	20
Lloyd	21
<b>VII. Time Worked</b>	<b>25</b>

## 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, platform-agnostic payload generation, and video-based covert communication protocols used to communicate with agents. This document outlines our efforts to verify and test that the project is suitable for industry, as well as recent project developments.*

## II. Project Updates and Changes

At the start of this semester, we had successfully developed a proof of concept in which we successfully sent arbitrary data over YouTube and used it to execute commands remotely on an “infected” device. Our challenge this semester was to convert this prototype into something actually usable from an industry perspective, providing the foundation for other users to develop an ecosystem built around covert communication.

Since P2, we have made extensive progress toward building this foundation – although many of these changes are invisible and have yet to manifest as part of the web interface, they form the basis for implementing tightly-scoped features leading up to Spring Break. A brief summary of the progress includes:

- The implementation of a package manager, allowing users to build, distribute, and install their own agents and protocols in individual DeadDrop installations;
- The development of standardized metadata files that allow agents to expose important interfaces and information in a language-agnostic manner;
- The publishing of [a standard meta-library for DeadDrop](#), providing common interfaces and expectations for what agents and protocols must expose;
- Continued research and development toward adapting the YouTube communication protocol to operate over live Zoom calls and instead, in addition to a new architecture and a mock YouTube service;
- New research towards operating a communication protocol over Craigslist;
- Continued progress towards the authentication and user system, allowing proper logging and accountability for actions taken through the framework; and
- The use of Docker Compose on all major components (the server, frontend, and Pygin), reducing setup time on individual machines and increasing overall reliability.

There have not been any major changes to the overall design or specification of the framework since P2. However, since we felt that we were sufficiently far ahead in the project, we chose to develop with the idea of a future DeadDrop ecosystem – a change from our original approach to continue development with only the end of the semester in mind. This has expanded the scope of the project

slightly, bringing us to a total of 82 backlog items (of which 67 should be done by the end of Spring Break, with around 50 completed as of writing).

The largest design change since the fall prototype has been the implementation of a package manager, in which all agents and protocols expose standard build scripts and interfaces that allow them to be installed into a single instance of the Django backend. Although we originally planned on simply using Git submodules for agent and protocol distribution, we identified a need for more flexible installations, allowing multiple versions of the same agent to exist on the server at once. This has been implemented and will be demonstrated during the Week 6 progress demos.

### III. User Stories and Acceptance Criteria

1. Task Listing
  - a. User Story: As a user, I want to be able to see all the tasks currently running.
  - b. Acceptance Criteria:
    - i. The user must be on the task page.
    - ii. All currently running tasks must be displayed on the tasks table.
    - iii. The user must be able to display a certain number of tasks per page of the table.
2. Log Reporting
  - a. User Story: As a user, I want to make logs that I can export.
  - b. Acceptance Criteria:
    - i. The user must be on the logs page.
    - ii. All currently existing files must be displayed on the files table.
    - iii. The user must be able to display a certain amount of files, per page of the table.
3. Command Form
  - a. User Story: As a user, I want to be able to send commands to an agent.
  - b. Acceptance Criteria:
    - i. The user must be on the agent detail view.
    - ii. The user must be able to select 'issue a command' from the quick links on the detail view.
    - iii. The user must be able to select between the available commands listed in the agent detail view.
    - iv. The agent must be able to receive and execute the command from the server on the host.
4. Payload Generation Form
  - a. User Story: As a user, I want to be able to generate a payload to send to the agent.
  - b. Acceptance Criteria:
    - i. The user must be on the protocol detail view.
    - ii. The user must be able to select 'create a new payload with protocol' from the protocol detail view.
    - iii. The user must be able to select between the compatible communication protocols for the agent, on the detail view.
    - iv. The agent must be able to receive and download the payload and communicate with the server through that protocol.
5. Dashboard
  - a. User Story: As a user, I want to see info about my tasks and logs quickly.
  - b. Acceptance Criteria:
    - i. The user must be on the dashboard page.
    - ii. The user must be able to view the number of communications from server to agent in the last 24 hours on the communications graph.
    - iii. The user must be able to view the distribution of those communications based on agent type on the 'endpoint communication share' pie chart.

- iv. The user must be able to see the current running tasks on the bottom table.
  - v. The user must be able to see the number of registered endpoints, messages sent, messages fetched, ongoing tasks, and completed tasks in the statistics table.
- 6. Protocol Handler
  - a. User Story: As a user, I want to be able to see which endpoints are using a protocol
  - b. Acceptance Criteria:
    - i. The user must be on the protocol page.
    - ii. There must be an endpoint connected to a protocol.
    - iii. The listing must show all protocols and their endpoints.
- 7. Agent Creator
  - a. User Story: As a user, I want to register an agent within DeadDrop.
  - b. Acceptance Criteria:
    - i. The user must be on the agent page.
    - ii. There must be a valid agent with the correct metadata configuration files.
    - iii. The user must upload the agent file to DeadDrop.
- 8. Agent Uninstallation
  - a. User Story: As a user, I want to remove all traces of an agent from a device.
  - b. Acceptance Criteria:
    - i. The user must be able to click a “delete” button on the agent’s detail page.
    - ii. Deleting an agent must issue a command request to the agent.
    - iii. The agent must remove all of its files using a built-in script that allows it to delete its own files while residing in memory.
- 9. Sign Up
  - a. User Story: As a new user, I want to sign up to use DeadDrop.
  - b. Acceptance Criteria:
    - i. The user must be on the sign up page.
    - ii. The user must enter a username that is not already registered within the DeadDrop database.
    - iii. The user must enter matching passwords.
    - iv. The user must click sign up after filling out the fields.
- 10. Login
  - a. User Story: As a user, I want to be able to log in and interact with DeadDrop.
  - b. Acceptance Criteria:
    - i. The user must be on the login page.
    - ii. The user must have a valid username and password in the database
    - iii. The user must enter the correct credentials to the login form.
    - iv. The user must receive a cookie that verifies their authentication.

## IV. Testing Workflow

HP1 An agent can be created and deployed			
Success:	All steps result in their expected output	When:	After endpoints can be created and before each pull request
		How:	Python script
Step	Action	Expected Output	
1	The method to create an agent package is invoked	An agent package is received in the expected format	
2	A docker container is launched	There is a running docker container on the testing machine	
3	Deploy the package to the container	An executable is present on the container	

HP2 Register endpoint			
Success:	All steps result in their expected output	When:	After endpoints can be registered and after HP1 was ran and before each pull request
		How:	Python Script
Step	Action	Expected Output	
1	Register endpoint from HP1 by invoking the register endpoint method	Success message is received from the endpoint over obfuscated channel	

HP3 Login			
Success:	All steps result in their expected output	When:	After authentication is implemented and before each pull request
		How:	Selenium
Step	Action	Expected Output	
1	Non authenticated user	The user is redirected to login pages	

	navigates to frontend	
2	User inputs correct username, correct password and hits enter	The user is redirected to the home page

HP4 Connect to endpoint			
Success:	All steps result in their expected output	When:	After connecting to endpoint is implemented and before each pull request
		How:	Selenium
Step	Action	Expected Output	
1	Authenticated user navigates to connect to endpoint page	Home page is shown	
2	User selects endpoint from dropdown and hits enter	User is redirected to console page that is connected to endpoint	
3	User types 'ls<CR>'	Console displays contents of the current working directory	

UHP1 Non-authorized access of internal page			
Success:	All steps result in their expected output	When:	After User authentication is implemented and before each pull is merged
		How:	Selenium web scraping script
Step	Action	Expected Output	
1	User accesses frontend without being authenticated	User is redirected to login page	
2	User uses url input to attempt to navigate to internal page	User is redirected to 401 not authenticated page	

UHP2
------



Youtube or other protocol bounce is inaccessible			
Success:	All steps result in their expected output	When:	After HP2 is successful and before each pull request
		How:	Python script
Step	Action	Expected Output	
1	At emulated endpoint start firewall that blocks protocol bounce point	Youtube.com or whichever site or fakesite is used is no longer accessible	
2	Send message over protocol	Message send fails	
3	Refer to configuration to determine next step	Possible options are: Use different protocol Try again at a different time Delete self and all trace	

UHP3 Endpoint is non responsive			
Success:	All steps result in their expected output	When:	After HP2 is successful and before each pull request
		How:	Python script
Step	Action	Expected Output	
1	Forcefully kill or remove agent	Agent is no longer reachable	
2	Attempt to connect to agent	No response is received Provide logs of recent events Keep channel open waiting for response	

UHP4 Server backend is not responsive			
Success:	All steps result in their expected output	When:	Before each pull request
		How:	Python script
Step	Action	Expected Output	
1	Launch application	Application frontend is displayed	
2	Forcefully kill backend	Frontend attempts to connect to backend	

		After timeout, redirect to http 503 page
--	--	------------------------------------------

## V. Testing Strategy

In general, DeadDrop can be split up into three components, each of which has its own testing demands:

- **The web interface.** This is subject to various accessibility tests, both fully automated and semi-automated.
- **The Django backend.** This is subject to automated acceptance and unit tests, as well as manual QA tests with “supported” agents and protocols.
- **Standalone components**, such as agents (Pygin) and protocols (dddb, our YouTube-based protocol). In practice, these are subject to their own tests, as they are not inherently part of the core DeadDrop architecture. However, for modules developed by our team, “local” functionality is subject to static code analysis (such as static type checkers and code linters) and automated unit tests.

(Note that when automated tests are mentioned below, it is assumed that the developer will run these automated tests before pushing changes to the repo. The same is true of the reviewer in the pull request, who is the second team member identified in each paragraph below - this reduces the likelihood that passing tests are the result of developer-specific environments.)

The web interface used to directly interact with the rest of the framework is primarily developed by Brian, with its primary tests encompassing accessibility and specific workflows. The majority of these tests are either automated or semi-automated. For example, Lighthouse CI can be used to evaluate the accessibility and general usability of pages over time, allowing us to keep track of regressions and potential improvements across the codebase. During development, both developers and testers can use the Lighthouse and SiteImprove browser extensions to identify errors on pages in real time. Certain complex workflows that are difficult to automate, such as those that require multiple components to be running and require manual setup, will generally be conducted by Jann due to his involvement in the backend; this allows identified defects to be scoped more quickly. For example, a test to assert that non-admin users cannot uninstall an agent currently in use involves not only the frontend and backend, but also packages generated externally with their own build process; this cannot be reliably automated and therefore must be done manually.

The Django backend used to manage the server (which in turn manages much of the C2 framework) is primarily developed by Jann. Django provides its own unit testing framework, which generally allows users to build test databases, send data to endpoints, and assert that models function as expected. Most of this testing can be automated, as most models have clearly defined functionality and interfaces. Because the frontend is entirely dependent on the structure of the JSON outputs produced by the frontend and the reliability of any exposed endpoints, Brian is responsible for conducting the tests developed for the backend. This not only ensures that the test results are reasonable from an implementation perspective (i.e. they work as expected), but that the test results are acceptable from a frontend design perspective (i.e. they provide the information needed for current and future frontend features). Any manual tests that are determined to be the scope of the Django frontend/backend interface will be conducted by Brian.

Finally, standalone components are developed by Lloyd (in the case of agents) and Keaton (in the case of protocols). These components are generally subject to both system tests and unit tests; system tests ensure that agents and protocols continue to adhere to the standard, platform-agnostic interfaces, while unit tests ensure that core services function as expected. For example, Pygin contains unit tests for asserting the correctness of how individual commands should execute. Simultaneously, there are also larger-scoped tests that use standard DeadDrop messages passed into individual modules to assert that the architecture as a whole works as expected. In general, because the agents rely so much on the protocols and vice versa, Lloyd and Keaton are responsible for testing each others' implementation; this generally entails manual testing for edge cases, in addition to the automated tests above.

The general process for handling defects is to open an issue in the offending repository with a minimum reproducible example of how to cause the bug to appear. Details of the user's testing environment and precise commit should be included, allowing both the reporter and the assignee to determine whether the defect is an inherent flaw in the implementation or a user-specific edge case (which must be resolved either way). When the bug can be reliably reproduced, it is fixed in a feature branch. When required, this patch is then cherry-picked to functional (i.e. not bugfix) feature branches to ensure that the patch has no cascading effects.

The project is considered complete when acceptance tests (automated, semi-automated, or manual) corresponding to each of our requirements are fulfilled without issue. Many of the functional requirements are inherently tied to existing automated tests, and therefore do not *strictly* need their own acceptance tests; more complex requirements, such as those that call for a human to determine if the requirement has been met, involve a combination of semi-automated and manual efforts. For example, asserting that a properly configured agent communicates solely over YouTube requires that a human configure tcpdump and/or Wireshark, after which the network dump can be automatically analyzed to determine if the acceptance test (and the requirement) has been met. The acceptance criteria are defined on a requirement-by-requirement basis, similar to those defined in [User Stories and Acceptance Testing](#). The same holds for nonfunctional requirements; for example, asserting that the "codebase adheres to PEP8" may involve the use of multiple Python linting tools and observing that no major issues are identified.

The generic test procedure asserting that all workflow items and acceptance criteria mentioned in Section 3 are fulfilled is as follows (which includes the development of the tests themselves):

- Map each requirement to a coherent acceptance test composed of one or more acceptance criteria. A combination of one or more existing automated tests may be sufficient to cover an acceptance test that adequately asserts a requirement has been fulfilled, with documentation.
- For requirements that are not covered by existing automated tests, automated tests should be written with the goal of upholding a typical user's expectations. This should be independent of any ongoing design decisions, forcing the interfaces to remain consistent despite changes in the underlying architecture (unless absolutely necessary).

- Where automated tests are not possible, acceptance tests and criteria should be expressed as a detailed sequence of manual steps.
- Other team members must now review these tests to resolve any ambiguity or baseless assumptions written within the tests, as these can have long-term impacts on the design of the framework.
- The feature covered by each acceptance test should be developed (if not already completed).
- Automated tests must pass as the framework is developed incrementally.
- Once the project is considered to be in a stable state, all automated and manual acceptance tests should be exercised in a clean environment by their respective authors *and* assignees (those who did not develop the feature but are assigned for testing, as mentioned above). Should all tests pass, the project can be considered complete.

The formal “test plan”, as described in the slides, can be seen on the following page. The outputs and results are reflective of the current state of the project.

## Acceptance Criteria and Testing Strategy and Plan (P3)

No.	Type	File	Name	Purpose	Environment	Expected Result	Actual Result	Outcome and Actions Required
1	Frontend	tasks/+page.svelte tasks/+page.js backend/views.py	Task display	Test that running Celery tasks are visible in the table	Python 3.11, pegged requirements to requirements.txt.  Svelte (latest), pegged requirements to package-lock.json.  Both should be run in a Docker Compose environment.	All AsyncResults known to the server that are currently running are visible.  Executing five long-running and five short-running tasks should only show the five long-running tasks in the table.	All as expected.	No actions required.
2	Frontend	commands/+page.svelte commands/+page.js backend/views.py backend/messages.py backend/commands.py	Command execution	Test that commands can be sent to the agent through the frontend.	The backend and frontend Docker Compose environment should be running with default committed configs.  The agent Docker Compose environment should be running with default committed configs.	Commands sent to the frontend should be encoded as a JSON message over a protocol.  The agent should execute the shell command.  The result of the command should raise a notification.	1 as expected. 2 as expected.  3 is not currently implemented and therefore is not supported.	The live push notification of a completed Celery task must be implemented, requiring work on both the frontend and backend.
3	Backend	commands/+page.svelte commands/+page.js backend/views.py backend/packages.py backend/payloads.py	Payload generation	Test that payloads can be generated using the package manager.	The backend Docker Compose environment mentioned above should be running.	The specified payload should be used as a template to generate an instance of Pygin in the build output folder.  The specified configuration should be included as part of the build result.	1 as expected.  2 is not currently implemented and therefore is not supported.	The process for accepting a build input and using it as part of the parameterized Docker build is well-defined but has yet to be implemented. This is required for this test to pass.
4	Frontend	dashboard/+page.svelte dashboard/+page.js backend/models.py backend/views.py	Dashboard	Test that the dashboard contains summary information for the last 24 hours.	The backend and frontend Docker Compose environments should be running.	The dashboard should contain aggregate information for the Endpoint, Task, and Log models.	1 as expected.  2 is not implemented, and so only placeholders are present.	The Django REST API does not currently expose options for filtering or pagination. This is currently in progress, and is

## Acceptance Criteria and Testing Strategy and Plan (P3)

						When task and log information associated with agents is outside of the last 24 hours, it should not appear.		required for Svelte-side filtering to work. Django already has filtering support, but does not expose it to the frontend.
5	Frontend	dashboard/+page.svelte dashboard/+page.js backend/models.py backend/views.py	Protocol identification	Test that users can see which endpoints are using a specific protocol from the protocols page.	The backend and frontend Docker Compose environments should be running.	When viewing a protocol detail page, all endpoints that use that protocol should be visible in the table.  A share of the agent types using the protocol should be visible.	1 and 2 are not implemented; only a frontend mockup is available, though the required information is passed to the frontend.	The frontend must render the relevant elements.
6	Backend	backend/views.py backend/packages.py backend/serializers.py	Agent package installation	Assert that users can install new bundles and that existing bundles are not overwritten.	The backend Docker Compose environment should be running.	When installing a new agent through the installAgent endpoint, a copy of the bundle should be created.  The bundle should be decompressed and copied to the backend's package listing.  An Agent model should be created on successful package installation.  If a bundle already exists, it should be rejected.	All as expected.	No actions required.
7	Frontend	backend/views.py backend/serializers.py credentials/+page.js credentials/+page.svelte	User sign up	Assert that the user can sign up, creating a User model with no permissions by default.	The backend and frontend Docker Compose environments should be running.	When signing up through the frontend, a new User model should be created.  The credentials used should be valid.	All as expected.	No actions required.
8	Frontend	backend/views.py backend/serializers.py credentials/+page.js credentials/+page.svelte	User login	Assert that the user can log in, passing Django session	The backend and frontend Docker Compose environments	When logging in, a Django session token should be	Works as expected.	No actions required.

## Acceptance Criteria and Testing Strategy and Plan (P3)

				information through Svelte.	should be running.	passed up through Svelte to the user, bypassing Svelte authentication.		
9	Backend	backend/views.py backend/serializers.py	Authentication test	Assert that users must be authenticated and have the required roles to access resources.	The backend Docker Compose environment should be running.	When attempting to access the dashboard (non-privileged), the user should be redirected to the login page.  When attempting to access the payload construction page, only administrative users should be allowed to access it.	1 as expected.  Role-based authentication from Svelte is not currently validated, though it is validated in Django.	Implement a "not authorized" error message in Svelte provided information from Django.
10	Backend	backend/views.py backend/models.py backend/packages.py backend/serializers.py	Package uninstallation	Assert that when agents or protocol models are deleted, their corresponding packages are deleted.	The backend Docker Compose environment should be running.	When deleting an agent through the RESTful API, the packages should be synchronously uninstalled.  When deleting an agent through the admin interface, the packages should be uninstalled.	All as expected.	No actions required.



## VI. Unit Testing

### Han

This unit test checks for the correct implementation of the user model. It checks to make sure that the creation of a user to the user database is working and the new user is being registered. This is important as when a new user registers, we would like for them to be able to log in without any problems.

This unit test uses Django's built-in testing framework which uses Python's [unittest](#) module. To run the test, you can execute `python3 manage.py test` or `make test`. The unit test shown below can be found in the [tests.py](#) file on the Django server repo.

```
# make test
python3 manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified some issues:

WARNINGS:
backend.Endpoint.connections: (fields.W340) null has no effect on ManyToManyField.

System check identified 1 issue (0 silenced).
.
-----
Ran 1 test in 0.272s

OK
Destroying test database for alias 'default'...
#
```

Full screenshot of the test output. All tests pass, so the entire run passes.

```
1  from django.test import TestCase, Client
2  from django.contrib.auth.models import User
3
4  # Create your tests here.
5  class TestModels(TestCase):
6      def test_model_User(self):
7          username = 'user'
8          pwd = 'pass'
9          user = User.objects.create_user(username=username, password=pwd)
10         self.assertEqual(str(user), username)
11         self.assertTrue(isinstance(user, User))
```

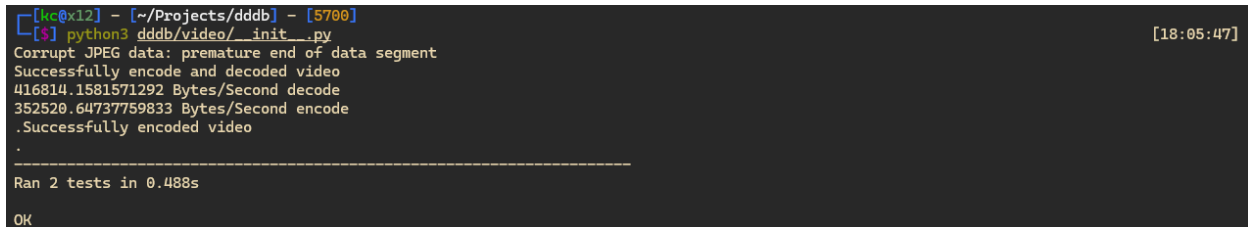
Screenshot of the unit test itself.



## Keaton

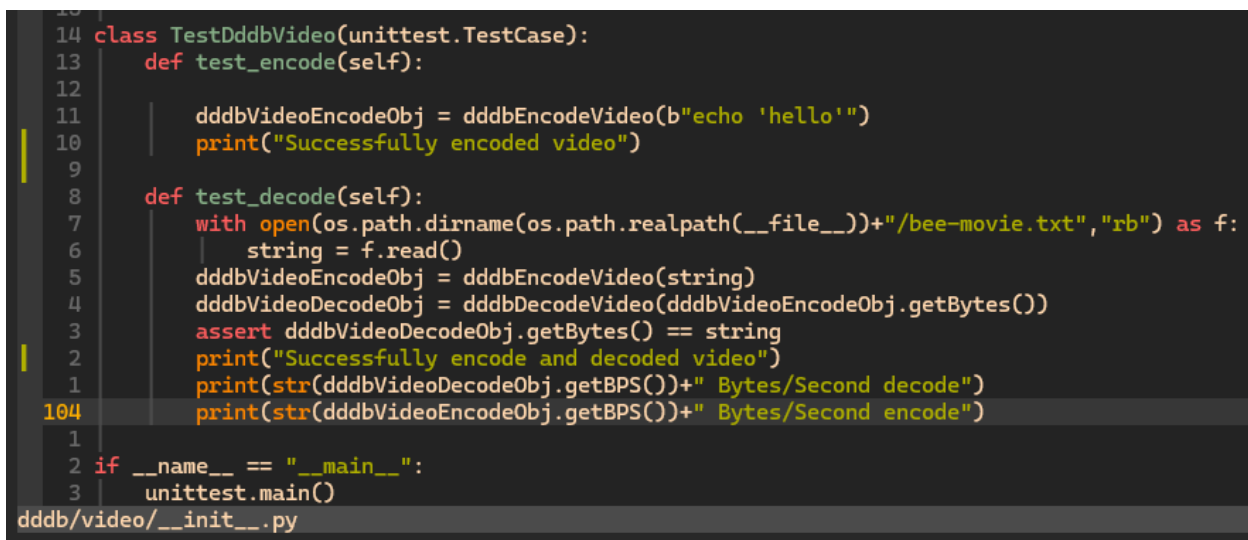
This unit test checks that DeadDrop Dropbox (dddb) is correctly encoding and subsequently decoding video. This unit test uses the default Python [unittest](#) module and so has no additional requirements that the library does not. To run the test simply execute the library module as you would any other Python script.

To test the video encoding it passes text into the encoder object and passes the resulting video as bytes into the decoder object. It then ensures that the resulting binary blob is equivalent to the text passed into the encoder.



```
[~@x12] - [~/Projects/dddb] - [5700]
[$] python3 dddb/video/__init__.py
Corrupt JPEG data: premature end of data segment
Successfully encode and decoded video
416814.1581571292 Bytes/Second decode
352520.64737759833 Bytes/Second encode
Successfully encoded video
.
-----
Ran 2 tests in 0.488s
OK
```

Screenshot of the unit test successfully running (also include Bytes/Second encode and decode)



```
14 class TestDddbVideo(unittest.TestCase):
13     def test_encode(self):
12
11         dddbVideoEncodeObj = dddbEncodeVideo(b"echo 'hello'")
10         print("Successfully encoded video")
9
8     def test_decode(self):
7         with open(os.path.dirname(os.path.realpath(__file__))+"/bee-movie.txt", "rb") as f:
6             string = f.read()
5             dddbVideoEncodeObj = dddbEncodeVideo(string)
4             dddbVideoDecodeObj = dddbDecodeVideo(dddbVideoEncodeObj.getBytes())
3             assert dddbVideoDecodeObj.getBytes() == string
2             print("Successfully encode and decoded video")
1             print(str(dddbVideoDecodeObj.getBPS())+" Bytes/Second decode")
104            print(str(dddbVideoEncodeObj.getBPS())+" Bytes/Second encode")
1
2 if __name__ == "__main__":
3     unittest.main()
dddb/video/__init__.py
```

Screenshot of the unit test itself (also includes just testing encoding)

## Brian

This unit test checks for the correctness of creating superusers, which ensures that only privileged users are allowed to access sensitive models (when configured). This forms the basis for the frontend's role-based authentication system. It checks that the creation of a superuser within Django's database works as expected, registering the administrator and marking their account as privileged. It additionally asserts that Django regular users are not marked as superusers internally, ensuring that unprivileged users are unable to access the admin interface or make direct edits to the underlying models.

This unit test uses Django's built-in testing framework, which utilizes python's unittest module. To run the test, you can execute `python3 manage.py test` or `make test`. The unit test shown below can be found in the `test.py` file of the `prototype_backend` repository.

```
class TestModels(TestCase):
    def test_is_superuser(self):
        suser_name = 'suser'
        spwd = 'spwd'
        username = 'uuser'
        upwd = 'upwd'
        super_user = User.objects.create_superuser(username=suser_name, password= spwd)
        user = User.objects.create_user(username=username, password=upwd)
        self.assertTrue(super_user.is_superuser)
        self.assertFalse(user.is_superuser)
```

Screenshot of the create superuser unit test.

```
bbuslon@bbuslonlenovo:/mnt/c/Users/buslo/OneDrive/Documents/DockerD/prototype_backend$ make test
python3 manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified some issues:

WARNINGS:
backend.Endpoint.connections: (fields.W340) null has no effect on ManyToManyField.

System check identified 1 issue (0 silenced).
.
-----
Ran 1 test in 0.233s

OK
Destroying test database for alias 'default'...
```

Screenshot of the unit test output.

## Lloyd

This unit test checks for the correct implementation of Pygin's command execution system. At its core, it tests the correct operation of the "ping" command, a low-complexity command provided by Pygin for testing connectivity between the server and the agent. However, it also indirectly tests that internal changes in Pygin's command execution system do not cause previously valid command request messages to suddenly fail.

This unit test leverages the [pytest](#) framework for testing, leveraging [tox](#) to set up automated testing environments on multiple Python versions. Running the unit test is as simple as installing tox with pip, and then executing `tox` at the root of Pygin's directory. The unit test shown below can be found in [Pygin's test directory](#).

The screenshot displays a development environment with three main panels. The left panel shows a Python file named `test_commands.py` with a `TestClass` containing a `test_ping` method. The code includes comments describing the requirements for the ping command and asserts that the response contains specific fields like `ping_timestamp` and `pong_timestamp`. The middle panel shows the `tox.ini` configuration file, which defines the test environment with `env_list = format, lint, type, py311` and `skipdist = True`. The right panel shows the output of running `tox`, which includes the pytest output for the `test_ping` test, showing it passed successfully.

```

pygin > tests > test_commands.py 1 M x
11
12 from src.agent_code.command_dispatch import execute_command
13
14 class TestClass:
15     def test_ping(self):
16         """
17         You, 35 seconds ago | 2 authors (Iqbal and others)
18         """
19         Check that the ping command functions as expected.
20
21         This tests the following requirements for the ping command:
22         - The ping command shall preserve and return the timestamp of the
23         command request.
24         - The ping command shall include the time of execution as part of the
25         result.
26         - The ping command shall echo the message specified by the user if one
27         was provided.
28         """
29         # Load the raw command request message
30         DOCUMENT = Path("../tests/resources/test_ping.json")
31         with open(DOCUMENT, "rt") as fp:
32             msg = DeadDropMessage.model_validate_json(fp.read())
33
34         # Execute the message as parsed by the library
35         result = execute_command(msg.payload['cmd_name'], msg.payload['cmd_args'])
36
37         # Assert that the response contains all of the following:
38         # - a ping_timestamp, equal to "ping"
39         # - a pong_timestamp that is greater than ping_timestamp and is within
40         #   10 seconds of the current time of the test (if this takes more than
41         #   a quarter of a second to execute, we have bigger problems)
42         # - the expected echo message
43
44         # Check that the ping_timestamp is equal to the original timestamp
45         assert result['ping_timestamp'] == msg.payload['cmd_args']['ping_timestamp']
46
47         # Check that the pong_timestamp is greater than that of the "base" message.
48         # This is valid check provided that the system time is correct, since
49         # all timestamps are assumed to be UTC
50         assert result['pong_timestamp'] > float(msg.payload['cmd_args']['ping_timestamp'])
51
52         # Assert that the listed pong timestamp is within 10 seconds of true time.
53         assert (datetime.utcnow().timestamp() - result['pong_timestamp']) <= 10
54
55         # Assert that the user's message, if any, was echoed back.
56         assert result['message'] == msg.payload['cmd_args']['message']
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
```

```

/mnt/c/Users/hbd/Desktop/Projects/unr-deadddrop/pygin 27-implement-tests >1 !3 system node 22:19:34
> tox
format: commands[0]> black ./src
All done! 🌟🍰🌟
29 files left unchanged.
format: OK ✓ in 1.81 seconds
lint: commands[0]> flake8 ./src
lint: OK ✓ in 2.5 seconds
type: commands[0]> mypy ./src
Success: no issues found in 29 source files
type: OK ✓ in 7.26 seconds
py311: commands[0]> pytest
===== test session starts =====
platform linux -- Python 3.11.6, pytest-7.4.0, pluggy-1.4.0
cachedir: .tox/py311/.pytest_cache
rootdir: /mnt/c/Users/hbd/Desktop/Projects/unr-deadddrop/pygin
collected 1 item

tests/test_commands.py . [100%]

===== 1 passed in 2.07s =====
format: OK (1.81=setup[0.10]+cmd[1.71] seconds)
lint: OK (2.50=setup[0.07]+cmd[2.43] seconds)
type: OK (7.26=setup[0.08]+cmd[7.18] seconds)
py311: OK (5.04=setup[0.10]+cmd[4.94] seconds)
congratulations :) (16.68 seconds)

```

The result of running tox, which runs several linters and static code checkers (black, flake8, mypy) in addition to pytest. All linters and tests pass, so the entire tox run passes for the Python 3.11 environment.

```

11
12 from src.agent_code.command_dispatch import execute_command
13
14 You, 1 minute ago | 2 authors (lgactna and others)
15 class TestClass:
16     def test_ping(self):
17         """
18         codeauthor:: Lloyd Gonzales <lgonzalesna@gmail.com>
19
20         Check that the ping command functions as expected.
21
22         This tests the following requirements for the ping command:
23         - The ping command shall preserve and return the timestamp of the
24         | command request.
25         - The ping command shall include the time of execution as part of the
26         | result.
27         - The ping command shall echo the message specified by the user if one
28         | was provided.
29         """
30         # Load the raw command_request message
31         DOCUMENT = Path("../tests/resources/test_ping.json")
32         with open(DOCUMENT, "rt") as fp:
33             msg = DeadDropMessage.model_validate_json(fp.read())
34
35         # Execute the message as parsed by the library
36         result = execute_command(msg.payload['cmd_name'], msg.payload['cmd_args'])
37
38         # Assert that the response contains all of the following:
39         # - a ping_timestamp, equal to "ping"
40         # - a pong_timestamp that is greater than ping_timestamp and is within
41         #   10 seconds of the current time of the test (if this takes more than
42         #   a quarter of a second to execute, we have bigger problems)
43         # - the expected echo message
44
45         # Check that the ping timestamp is equal to the original timestamp
46         assert result['ping_timestamp'] == msg.payload['cmd_args']['ping_timestamp']
47
48         # Check that the pong timestamp is greater than that of the "base" message.
49         # This is valid check provided that the system time is correct, since
50         # all timestamps are assumed to be UTC.
51         assert result['pong_timestamp'] > float(msg.payload['cmd_args']['ping_timestamp'])
52
53         # Assert that the listed pong timestamp is within 10 seconds of true time.
54         assert (datetime.utcnow().timestamp() - result['pong_timestamp']) <= 10
55
56         # Assert that the user's message, if any, was echoed back.
57         assert result['message'] == msg.payload['cmd_args']['message']

```

Screenshot of the unit test itself.

```

wsl x
cachedir: .tox/py311/.pytest_cache
rootdir: /mnt/c/Users/hbd/Desktop/Projects/unr-deaddrop/pygin
collected 1 item

tests/test_commands.py F [100%]

===== FAILURES =====
TestClass.test_ping

self = <tests.test_commands.TestClass object at 0x7f9b21496c50>

def test_ping(self):
    """
    codeauthor:: Lloyd Gonzales <lgonzalesna@gmail.com>

    Check that the ping command functions as expected.

    This tests the following requirements for the ping command:
    - The ping command shall preserve and return the timestamp of the
      command request.
    - The ping command shall include the time of execution as part of the
      result.
    - The ping command shall echo the message specified by the user if one
      was provided.
    """
    # Load the raw command_request message
    DOCUMENT = Path("./tests/resources/test_ping.json")
    with open(DOCUMENT, "rt") as fp:
        msg = DeadDropMessage.model_validate_json(fp.read())

    # Execute the message as parsed by the library
    result = execute_command(msg.payload['cmd_name'], msg.payload['cmd_args'])

    # Assert that the response contains all of the following:
    # - a ping_timestamp, equal to "ping"
    # - a pong_timestamp that is greater than ping_timestamp and is within
    #   10 seconds of the current time of the test (if this takes more than
    #   a quarter of a second to execute, we have bigger problems)
    # - the expected echo message

    # Check that the ping timestamp is equal to the original timestamp
    assert result['ping_timestamp'] == msg.payload['cmd_args']['ping_timestamp']

    # Check that the pong timestamp is greater than that of the "base" message.
    # This is valid check provided that the system time is correct, since
    # all timestamps are assumed to be UTC.
    > assert result['pong_timestamp'] < float(msg.payload['cmd_args']['ping_timestamp'])
E   assert 1709475909.333437 < 1709442984.879301
E   + where 1709442984.879301 = float(1709442984.879301)

tests/test_commands.py:50: AssertionError
===== short test summary info =====
FAILED tests/test_commands.py::TestClass::test_ping - assert 1709475909.333437 < 1709442984.879301
===== 1 failed in 4.10s =====
py311: exit 1 (8.90 seconds) /mnt/c/Users/hbd/Desktop/Projects/unr-deaddrop/pygin> pytest pid=2481

```

Demonstration of a failing unit test and its traceback (which was achieved by inverting one of the conditions in the unit test assertions).



## VII. Time Worked

Note that the recorded time includes time spent performing research, writing code, and implementing the modules described in this document (since P2). This also includes meetings to discuss the project as a team as relevant to P3.

Team Member	Hours	Sections Contributed
Jann Arellano	20	<ul style="list-style-type: none"> <li>- User Stories and Acceptance</li> <li>- Unit Testing</li> </ul>
Brian Buslon	15	<ul style="list-style-type: none"> <li>- User Stories and Acceptance</li> <li>- Unit Testing</li> </ul>
Keaton Clark	23	<ul style="list-style-type: none"> <li>- Testing Workflow</li> <li>- Unit Testing</li> </ul>
Lloyd Gonzales	19	<ul style="list-style-type: none"> <li>- Abstract</li> <li>- Project Updates and Changes</li> <li>- Testing Strategy</li> <li>- Unit Testing</li> </ul>