CYBER1-CodeVersion

Prototype Report

Tashi Stirewalt, Scalable Algorithms for Data Science Lab

Code Crafters

Samantha Brewer, Bryan Frederickson

Table of Contents

I. Description and Clarification

I.1 Introduction

Malware detection works through various different methods, with one of the most popular being a signature-based system. This signature-recognition system compares a given file to a database of known malware to find matches, so that once malicious software is identified it is not allowed access to the system. However, the drawback is that simple obfuscation methods can render malware unknown to the detection software and allow it to pass unhindered.

As such, the goal of this project is to create an algorithm to generate permutations of a given source code file that accomplish the same function, so it can be integrated into databases to better identify evolving malware. Multiple obfuscation techniques will be tested, such as simply changing the variable names and moving code blocks around, to changing the flow of the program to something unique from the original.

I.2 Background and Related Work

As said, multiple network intrusion detection systems exist, including anomaly-based, which looks for deviations from normal behavior in the network, and knowledge-based, which compares known malicious attacks to new incoming data [1].

Of knowledge-based detection systems, our project focuses on signature-based detection. Recent work for signature-based has been for developing ways to preemptively detect malicious attacks. One method was to create a way to extrapolate from the often vague descriptions of attacks and vulnerabilities such as with CVEs, so that said threat can be identified before a signature has become available [2]. Another method is pattern matching specifically for high-speed network connections, examining packets for malicious communications [1].

Specific skills and knowledge required for this project revolve around how the permutations of code will be generated. There are multiple possible implementations, including:

- Non-Deterministic Graph Algorithms
- Evolutionary Algorithms
- Machine Learning Algorithms
- Automating Code Obfuscation Techniques

The team will also need to be able to parse the source code, with Abstract Syntax Trees being a likely choice for how it will be represented internally. As such, Python will be the coding language of choice for its accessibility, popularity for data mining analysis, and a preexisting library for Abstract Syntax Trees.

I.3 Project Overview

Digital signatures are the primary key to identifying malware within source code. While digital signatures are helpful for Intrusion Detection and Prevention Systems, they will not always be able to stop every piece of malware that makes its way into crucial systems. As technology advances and more devices make their way into the cloud, securing our devices has never

been more of a challenge. One of the biggest threats to these systems is malware. Malware can present itself in numerous forms which are programmed by malicious actors. With the evolution of malware detection software, digital signatures have been recorded to help cyber security experts recognize known pieces of malware. Here the problem arises, where malicious actors can rework malware code that is known to be recognized by security systems. The significance of our project is to resolve and analyze these problems.

The premise of our project can be summarized as utilizing algorithms to determine the nature of how malware changes over time. The process begins with inputting a Python source code file into our tool. This Python file is not necessarily a piece of working malware, but a simple source code file that may contain ordinary functionality. Our tool also accepts an ‘N’ parameter which represents the number of semantic clones it will produce. To generate the semantic clones, the source code undergoes an algorithmic process that can be implemented in a variety of different ways.

The base process consists of converting a Python source code file to a string. From a string, we will utilize public Python libraries to build a graph from the string that contains nodes. These nodes define attributes about the code that was read such as functions, variables, loops, etc. After obtaining our graph data structure such as an Abstract Syntax Tree, we can run nondeterministic algorithms which will manipulate the source code structure from the graph, to obtain a new semantic clone graph. A reversed process to convert the new graph to a string is performed. From the new string, we can produce a new source code file which is now a successful semantic copy of the source code. This process will take place ‘N’ times to produce ‘N’ semantic clones. For the genetic algorithm approach, the semantic clones undergo fitness testing to see how well they compare to the source code. Iterative processes are performed to extract the “best fit” semantic clones.

While creating a top-tier Graphical User Interface isn’t at the top of the priority scale, the Cyber1-Code version will consist of a UI that accepts an ‘N’ parameter and a way to upload a Python source code file to be parsed. If we were to implement some feature at the end of the tool prototype, it would be to implement a more complex GUI that visually displays the changes to program graphs so that the end user can better understand the shifts and changes from the source code.

The desired outcome for the Cyber1-CodeVersion is to produce a baseline working tool that serves as a benchmark for understanding how malware may change over time. The use of this tool will be extremely significant to cyber security experts because performing malware analysis with the human eye is a very time-consuming process. The goal of this tool is to provide an efficient way to analyze malware files and recognize diverse patterns of changes to the original code.

As malware is constantly evolving, one of the most important objectives of this project is to carry on the process to other students beyond the work of Code Crafters. The software will not be a viable source for too much longer, for it will have to be developed and more efficient as malware becomes more complex. While malware detection software such as Bitdefender and Norton provide security at a high scale, they do not provide such a feature such as *learning* about the malware.

I.4 Client and Stakeholder Identification and Preferences

The primary clients for our project are Tashi Stirewalt and Assefaw Gebremedhin, who represent our main mentors and point of contact for the project. The other primary clients are cybersecurity experts who will utilize the tool for understanding malware and the Scalable Algorithms for Data Science Laboratory through Washington State University. Some of the significant stakeholders for our project include future student researchers who will continue to develop the efficiency of the project, as well as end users for the product to rework specifications and improve the quality of the tool.

The necessities and deliverables expected from our main clients include a detailed plan for how to approach building the tool, a defined set of algorithms to analyze code structure, a working software application that accepts source code files and outputs semantic clone source files, and written documentation that allows cyber experts to utilize the tool for studying how malware changes over time. The outlined preferences delivered by our client are that the use of Large Language Models should not be integrated into the tool to solve the problem. LLMs do not provide any insightful work towards solving the solution. The use of non-deterministic algorithms or genetic algorithms was encouraged and can be effectively used to produce an output for our software. For our stakeholders, the deliverables provided will be a base working software that can be built upon and developed with more features as malware becomes more complex to analyze.

For future tool use, we imagine that the software can be run on multiple types of operating systems, as well as the ability to handle different source code languages other than Python. While we are more focused on producing a baseline tool to work specifically with Python language, it is important to emphasize the usability of this tool across multiple systems.

II. Requirements and Specifications

II.1 Introduction

As a cybersecurity expert, designing malware detection software is a crucial responsibility in ensuring the integrity of user hardware, software, and other high-value systems. While complex, these malware analysis software applications use a variety of methods to identify malware on user systems such as digital signatures. In a perfect world, malicious actors would stop creating new forms of malware if their previous attempts were identified and blocked by these malware detection systems. However, one of the main challenges in the field of cybersecurity relates to the fact that malware and technology are constantly evolving, and the fight to ensure security is everlasting. While digital signatures are an effective way to identify malware, this method can only work for so long. After attackers recognize patterns that their malicious code is recognized by malware detection systems, nothing stops them from changing the original source code to produce a different digital signature.

The objectives we aim to accomplish by designing the Cyber1-CodeVersion tool are to create an ‘N’ number of unique semantic clone variations sprouted from an original Python source code file, visualize changes made to semantic clone files to understand how the source code has mutated to other branches, and to enforce a non-deterministic output approach on AST’s to

generate semantic clones and understand how source code can be changed over time. The end goal we hope to achieve is to create a base tool for understanding how malware changes can be used by cybersecurity professionals, as well as cyber researchers that can integrate new findings into the software to make detecting malware more efficient and effective. Clear and concise documentation will be generated from the Cyber1-CodeVersion for tool usage and ideas stemming from the project could be integrated into existing malware detection systems. We are designing the tool with efficiency in mind, which takes more of the pressure off of the people analyzing malware with the human eye so that they can see the big picture.

II.2 System Requirements Specification

This section of the project report defines functional and non-functional requirements, use cases and stories, UML, as well as a traceability matrix from the Cyber1-CodeVersion tool.

II.2.1 Functional Requirements

The boxes below represent functional requirements from our client with information such as the function requirement title, the description of the functional requirement, the source of the functional requirement, and the priority level of the functional requirement.

1. User Input and File Handling

Source Code File Box [FR-1.1]	
Description	The application needs to contain a section/box labeled with a logo of a file, that allows the user to either drag and drop or select a specific file from their local machine to be analyzed by the tool. This file drop box should only accept Python source code files.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, learning and analyzing the source code files is a requirement to understand how malware changes. This interface allows the professional to select which file to analyze.
Priority	Priority Level 0: Essential and required functionality.

‘N’ Semantic Clones Parameter [FR-1.2]	
Description	The application needs to contain a text box that accepts user input of an integer. This integer represents the number of semantic clones that will be produced from the selected source file by the cybersecurity professional. This integer should not be super large in order to maintain the quick runtime of the application.

Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, the more branches we produce of a source code file, allow for a better understanding of the diverse ways code can change over time. This user input allows the professional to create different variations of the original Python file.
Priority	Priority Level 0: Essential and required functionality.

Saves Clones Button [FR-1.3]	
Description	The application needs to contain a “Save Clones” button that opens the local machine file explorer application, which allows the user to select which directory the clones should be saved under. Clones should be able to be saved individually, as well as all together.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, if the user wants to do a more intensive investigation on individual source code files, they should be able to download them on their local machine.
Priority	Priority Level 1: Desired functionality.

2. Semantic Clone Generation

Generate Clones Button [FR-2.1]	
Description	The application needs to contain a “Generate Clones” button that effectively creates an ‘N’ number of semantic clone files stemming from the original Python file. These clones all need to be diverse from one another to enforce the rule of non-determinism.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, obtaining the clones is a key requirement to see how source code changes over time. This button is a necessity to obtain these clones.
Priority	Priority Level 0: Essential and required functionality.

Semantic Clone Output Box [FR-2.2]	
Description	The application needs to contain an additional file box that is populated with semantic clone files once they are done generating from the source code. This box should be able to allocate up to ‘N’

	specified spots which are chosen by the user.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, it is ideal to be able to see the actual clones that have been generated. This output box allows you to see all the permutations that were created after the semantic clone generation process.
Priority	Priority Level 0: Essential and required functionality.

Summarize Clones Button [FR-2.3]	
Description	The application needs to contain a button that allows users to summarize statistics such as runtime, time complexity, and storage that each semantic clone takes up in memory. These statistics help cybersecurity experts analyze the behavior of how source code changes.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, instead of individually analyzing source code files, bringing these relevant statistics to the user allows for a more efficient malware analysis process.
Priority	Priority Level 1: Desired functionality.

II.2.2 Non-Functional Requirements

Python Exclusivity [NFR-1]:

For the first implementation of the program for our research project, Python will be used as it has a native and robust library for Abstract Syntax Trees, however, it only supports parsing Python files.

Multi-Platform Usability [NFR-2]:

As Python is not compiled into an executable, the code can be parsed regardless of the source platform. The user program will likely be a Jupyter Notebook file which can be run in a web browser.

English-Only Support [NFR-3]:

Only English will be supported for parsing, as the developers are only familiar with English, and finding source files in other human languages will not be feasible.

Robust Memory Size [NFR-4]:

As users can specify the number of permutations, the program would need a large amount of space for parsing and creating unique output as the number requested increases.

Quick Response Time [NFR-5]:

As the AST library is natively supported in Python, the parsing time should take only as long as compile time, while the creating permutations are dependent on the amount requested. Using Jupyter Notebook will allow the user to rely on much larger memory for processing.

Multi-File Support [NFR-6]:

As many Python files rely on external files or libraries, to fully understand the function of the code for semantic clone generation, processing all external dependencies will be necessary. An option to select a folder or a single file may be one implementation, running on the assumption that the user has those libraries included.

II.2.3 Use Cases

The use cases describe common scenarios of user interactions with the proposed system, showcasing how the previous functional requirements are used in given situations or control flows. The overall use case is presented in Figure 1 below, with each section being explained in its section further below.

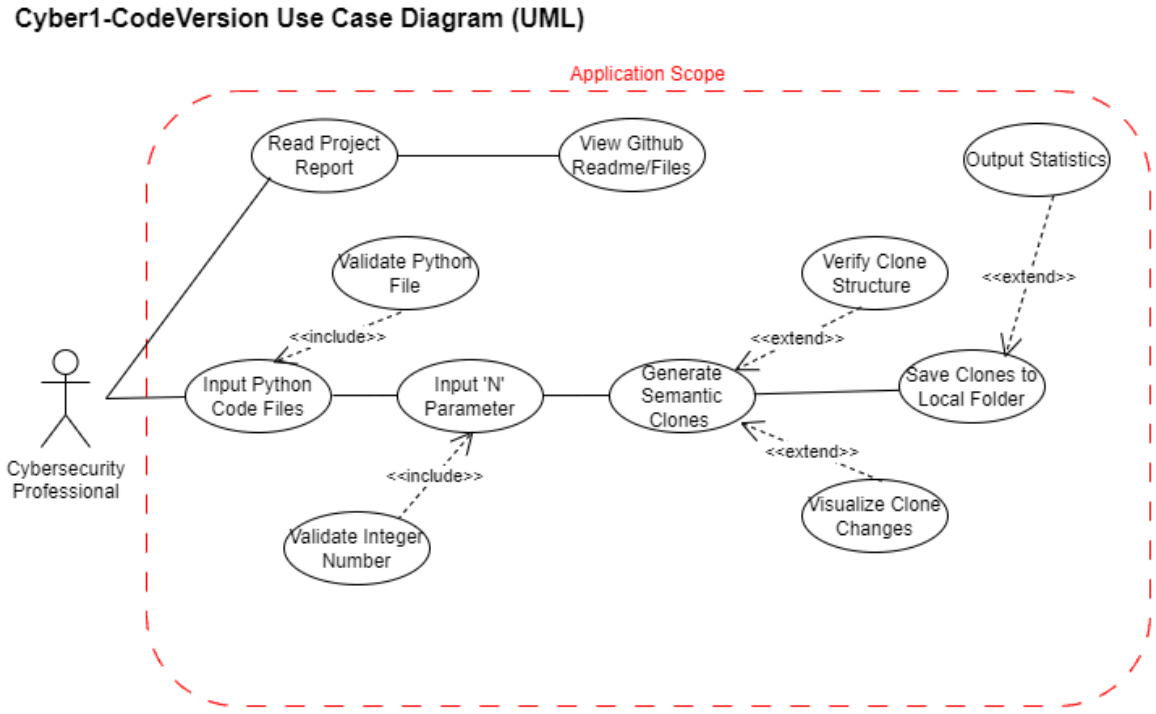


Figure 1: Use-Case Diagram

In *Figure 1*, the use cases that will be covered in depth are the basic path requirements. These basic path nodes are requirements that have a core function in the Cyber1-CodeVersion tool of a priority level of 1 or lower. For the other nodes in the UML, these describe extra features that will be integrated into the tool that represent a core functionality for the software to run. The actors of each use case will be the Cybersecurity expert because that is the essential target client of who the tool will be utilized by.

Read Project Report [UC-1]	
Pre-Condition	User opens the application interface/UI.
Post-Condition	Project report has been opened displaying all important documentation.
Basic Path	1. User opens file which contains project report information 2. Project report is displayed on the user's screen
Alternative Path	N/A
Related Requirements	N/A for functional but required for initial understanding.

View Github Readme/Files [UC-2]	
Pre-Condition	User has familiarized and reviewed this project report documentation.
Post-Condition	User is familiarized with how to use the software.
Basic Path	1. User navigates to the GitHub link. 2. Users click on Readme files or other source code files. 3. Users are equipped to use software and interact with tool features.
Alternative Path	N/A
Related Requirements	N/A for functional but required for initial understanding.

Input Python Code Files [UC-3]	
Pre-Condition	User has the application tool open and has Python source code files ready to upload
Post-Condition	Python files are successfully uploaded to the tool and ready to be branched into semantic clones.
Basic Path	1. User navigates to the file drop box and clicks the “Upload Files” button. 2. Users either drag and drop or click the Python file they want

	to be analyzed. 3. Python File is displayed with a checkmark if the file is correctly accepted into the file drop box.
Alternative Path	1. If Python files are not accepted because of structure issues, prompt users to resubmit valid code files. 2. User uploads a correctly formatted Python file and is accepted into the file drop box.
Related Requirements	Source Code File Box

Input ‘N’ Parameter [UC-4]	
Pre-Condition	User has uploaded valid Python source code files.
Post-Condition	The value of ‘N’ is evaluated and accepted if the integer is below a certain cap.
Basic Path	1. Locate the text box that accepts the ‘N’ parameter. 2. Enter an integer value that falls below the cap limit. 3. Click the “Confirm” button that allows the integer to be processed.
Alternative Path	1. User enters an integer that exceeds the cap size which will prompt the user to enter a smaller integer value. 2. The user enters an acceptable integer size which allows the program to run quickly.
Related Requirements	‘N’ Semantic Clones Parameter

Generate Semantic Clones [UC-5]	
Pre-Condition	User has a valid Python source code file in the file drop box and has specified an ‘N’ integer amount of semantic clones that will be generated.
Post-Condition	Semantic clone Python files are displayed in another file handler box that displays the generated clone files that can be downloaded.
Basic Path	1. User navigates to the “Generate Clones” button and clicks it. 2. Semantic clone files are created over time and output to the file handler box. 3. Users may select specific files to download or a “Download All” button option that downloads all clone files to the local machine directory.
Alternative Path	1. Semantic clone generation results in an error and notifies the user which file it had trouble processing.

	2. User makes changes to the source code file that handles the issue. 3. User resubmits the file to the text box and clicks the “Generate Clones” button which correctly creates the clone files.
Related Requirements	Generate Clones Button, Summarize Clones Button

Save Clones to Local Folder [UC-6]	
Pre-Condition	User has activated the “Generate Clones” button and the semantic Python files have been produced by the tool.
Post-Condition	The selected directory chosen by the user is populated with the semantic clone Python files.
Basic Path	1. User generates semantic clone files. 2. User clicks the “Save” button which opens the device explorer. 3. User navigates to the desired directory they want files saved to. 4. User clicks the “Save” button within the file explorer UI and saves files to the picked directory.
Alternative Path	1. User selects a ‘File’ to upload to instead of a ‘Directory’. 2. Users are notified that Python clones can only be updated into a Directory. 3. User selects a valid folder to upload clones files to and successfully downloads to the desired directory file.
Related Requirements	Semantic Clone Output Box, Save Clones Button

Below are some short explanations of the non-required features labeled in *Figure 1*.

Validate Python File [UC 3.1]: This feature determines whether the Python source code input file is syntactically correct or can be formed into an AST.

Validate Integer Number [UC 4.1]: This feature determines whether the integer input number exceeds a certain number in order to keep the runtime and space of the program to an efficient level.

Visualize Clone Structure [UC 5.1]: This feature allows the user to visualize the structure of the AST graphs to have a better pictorial grasp of how semantic clones differ from each other.

Visualize Clone Changes [UC 5.1]: This feature allows the user to see the highlighted changes in semantic clone code from the original source file.

II.2.4 User Stories

User Story US1: Select Python Source File

As a user, I want to be able to browse my storage when selecting source code input so that I can easily select the correct Python source file

Feature: Source Code File Box [FR-1.1]

Scenario: User clicks Source Code File Box button
Given the user has a valid Python file
When they navigate to where it is in their storage
Then the Python source file should show up and be selectable

User Story US2: Show Only Python Source Files

As a user, I want to only see valid Python files when selecting the source code input so that I have an easy time finding the correct file I want to submit.

Feature: Source Code File Box [FR-1.1]

Scenario: User clicks Source Code File Box button
Given the user has a valid Python file
When they navigate to where it is in their storage
Then only the Python files should be visible to be selected

User Story US3: Input Valid Number Parameter

As a user, I want to be able to be able to input a number for the N parameter of clones to generate so that I can specify how many I want for a given run

Feature: ‘N’ Semantic Clones Parameter [FR-1.2]

Scenario: User clicks Source Code File Box button
Given the user has a valid Python file
When they navigate to where it is in their storage
Then only the Python files should be visible to be selected

User Story US4: Save Each as Generated

As a user, I want to have the option to have each generated clone to be saved before continuing so that progress is not lost if the process ends prematurely.

Feature: Save Clones Button [FR-1.3]

Scenario: Clone Generation Begins
Given the save clones location given earlier is valid and the user activated autosaving
When a clone is generated

Then the clone is saved to the location before generating the next clone

User Story US5: Hands-Off Generation

As a user, I want the program to not need any more input after beginning generation until it is finished so that I can leave it running without having to constantly watch it.

Feature: Generate Clones Button [FR-2.1]

Scenario: Clone Generation Begins
Given the inputs are all valid
When the program begins clone generation
Then the program requires no oversight until it has finished or it crashes

User Story US6: View Outputs in Program

As a user, I want to be able to view generated clones in the program so that I do not need to download the outputs if they are not useful.

Feature: Semantic Clone Output Box [FR-2.2]

Scenario: Clone Generation Finishes
Given the generation was successful and the amount generated can fit in memory/storage
When the user clicks the Clone Output Box
Then they can see the generated clones

User Story US7: Statistics on Generation

As a user, I want to see the statistics of the previous run of clone generation so that I can see if there were any issues with the generation.

Feature: Summarize Clones Button [FR-2.3]

Scenario: User clicks Summarize Clones Button
Given the clone generation was successful
When the user clicks the Summarize Clones Button
Then the program displays statistics such as average time per clone, overall runtime, and size in memory

II.2.5 Traceability Matrix

The table below maps functional requirements to their respective use cases and user stories. This ensures that all requirements are accounted for and linked to user scenarios.

The priority rankings are as follows:
Priority Level 0: Essential and required functionality.

Priority Level 1: Desirable functionality.
Priority Level 2: Extra features or stretch goals.

Functional Requirement	**Use Case**	**User Story**	**Priority**
FR-1.1: Source Code File Box	UC-3: Input Python Code Files	US1: As a user, I want to be able to browse my storage when selecting source code input	Level 0
FR-1.1: Source Code File Box	UC-3: Input Python Code Files	US2: As a user, I want to only see valid Python files when selecting the source code input	Level 1
FR-1.2: ‘N’ Semantic Clones Parameter	UC-4: Input ‘N’ Parameter	US3: As a user, I want to be able to be able to input a number for the N parameter of clones to generate	Level 0
FR-1.3: Save Clones Button	UC-6: Save Clones to Local Folder	US4: As a user, I want to have the option to have each generated clone to be saved before continuing	Level 1
FR-2.1: Generate Clones Button	UC-5: Generate Semantic Clones	US5: As a user, I want the program to not need any more input after beginning generation until it is finished	Level 0
FR-2.2: Semantic Clone Output Box	UC-5: Generate Semantic Clones	US6: As a user, I want to be able to view generated clones in the program	Level 0
FR-2.3: Summarize Clones Button	UC-5: Generate Semantic Clones	US7: As a user, I want to see the statistics of the previous run of clone generation	Level 2

II.3 System Evolution

As this is a research project first and foremost, multiple different implementations of the program for generating semantic clones will be created. The focus currently is on Abstract Syntax Trees

and a comparatively simple and rudimentary permutation creator, but later down the line more complex methods such as Non-Deterministic Algorithms and Evolutionary Machine Learning may be implemented. This results in the requirements specified in this section changing and growing as the project develops, with separate sections for each implementation being one potential way to write this report.

Both developers only have access to Windows and Linux systems, so if a downloadable program that runs on the system is created, it may only be able to support one or both of those systems. Mac support is unlikely unless the developers have extra time down the line to address it as a stretch goal. However, for the first implementation, an in-browser Jupyter Notebook file will be used, which is platform-independent.

Requirements may change for the AST implementation, as the developer's unfamiliarity with Python's AST functionality could result in finding more appropriate libraries to use as development continues. Specifically, if implementing the code parsing and semantic clone generation ends up too rudimentary at first, as the developers become more comfortable working with ASTs, more ways to generate semantic clones could be worked on. Other libraries may be removed if they are too inefficient compared to another library discovered later down the line.

Lastly, due to the project being composed of two active members, some features may need to become stretch goals to ensure a deliverable prototype is available in time. Depending on how involved creating an algorithm to automate semantic clone generation will end up being, creating things such as a user interface compared to a simple Python Jupyter Notebook may become an extra goal to implement.

III. Project Solution Approach

III.1 Introduction

This document lays out the specifications for developing a program to generate semantic clones of a given source code via syntax trees. Its intended readers are mainly the clients of the developers working on this project.

This program aims to create permutations of a known code file so that signature-based intrusion detection systems can detect a known malicious code after it has been changed and obfuscated. Syntax trees, for this project, are a specific representation of a given program's behavioral structure through the lenses of individual chunks of code, such as functions leading to the various possible outputs or subsequent actions. This allows for traversing through the program or viewing a specific branch of it. Depending on the syntax tree used, the syntax tree can be edited and then converted back into functional code, which will be touched on below in the Data Design section.

III.2 System Overview

The main functionality of the Cyber1-CodeVersion tool aims to generate diverse types of visualizations between semantic clones for cybersecurity experts. The data visualizations include metrics such as similarity differences between clones, graph representations of

programs and how they are altered over time, and runtime statistics. These visualizations can be interpreted by the user and used to make informed decisions regarding code optimizations, security vulnerabilities, and maintenance strategies. This helps cybersecurity experts better understand how code evolves, identify potential risks, and improve the overall quality and security of the software. To ensure the Cyber1-CodeVersion tool produces this overall functionality, we have divided tasks of production into three major categories; Base User Interface, Semantic Clone Generation and Handling, as well as Visualization. In order of priority, our client requested that Visualization be the most significant and properly produced since that will be the source of what cybersecurity professionals utilize to interpret the results of the semantic clone process.

III.3 Architecture Design

This Architecture Design section aims to equip the user with a top-down design of how the Cyber1-CodeVersion functions with the support of subsystems and various architectural layers. The tool is built on a layered architecture pattern, which allows for modularity across different parts of the system. Key subsystems include the Base User Interface, Semantic Clone Generation, and Visualization, each playing a critical role in delivering the tool's core functionality.

III.3.1 Overview

To represent our tool from a top-down perspective, we decided to use a Component-Based Architecture. This architectural pattern suits our Cyber1-CodeVersion tool best because, from a conceptual perspective of how all the subsystems interact with each other, it made sense to group the tool into three layers. Identifying these groups allowed the team to identify connections between provided and required components. With visualization such as the component diagram provided below, the efficiency of designing user stories and prioritizing the development of subcomponents will be organized.

The three layers we defined are the UI components, Semantic Clone Processing, and Visualization. For the UI components, we of course support a main or base GUI that encapsulates the 'N' parameter, a file input user interface, as well as a semantic clone output UI. Overall, these different interfaces make up the main GUI and serve as provided interfaces for user interaction. In terms of the Semantic Clone Processing layer, this involves handling all the outputs from running the non-deterministic algorithms and utilizing the UI for cybersecurity professionals to take certain actions on those results. The last layer and most important layer, the Visualization layer utilizes external Python libraries to make data representation a lot cleaner and readable for the user and allows end users to make informed decisions about the visuals they are presented with such as graphs and numerics.

Our system isn't web-emphasized or database-oriented, though we are still able to identify three major layers that clearly define the main requirements for our tool. The major utilities used in all of our subcomponents are libraries provided by Python which include the tkinter library which is used for GUI features and file handling, the ast library for checking code syntax and converting programs to graphs, as well as the matplotlib library for graph and statistical plotting. Since most of the non-deterministic is pretty difficult to design in practice, Python provides easy API methods to simulate the "non-deterministic" aspect by modifying data structures. Most of our group's efforts will be put forth towards the data visualization aspect of the semantic clones. Under the hood, there are many forms of value and parameter validation that will serve as a

requirement for the tool to run correctly.

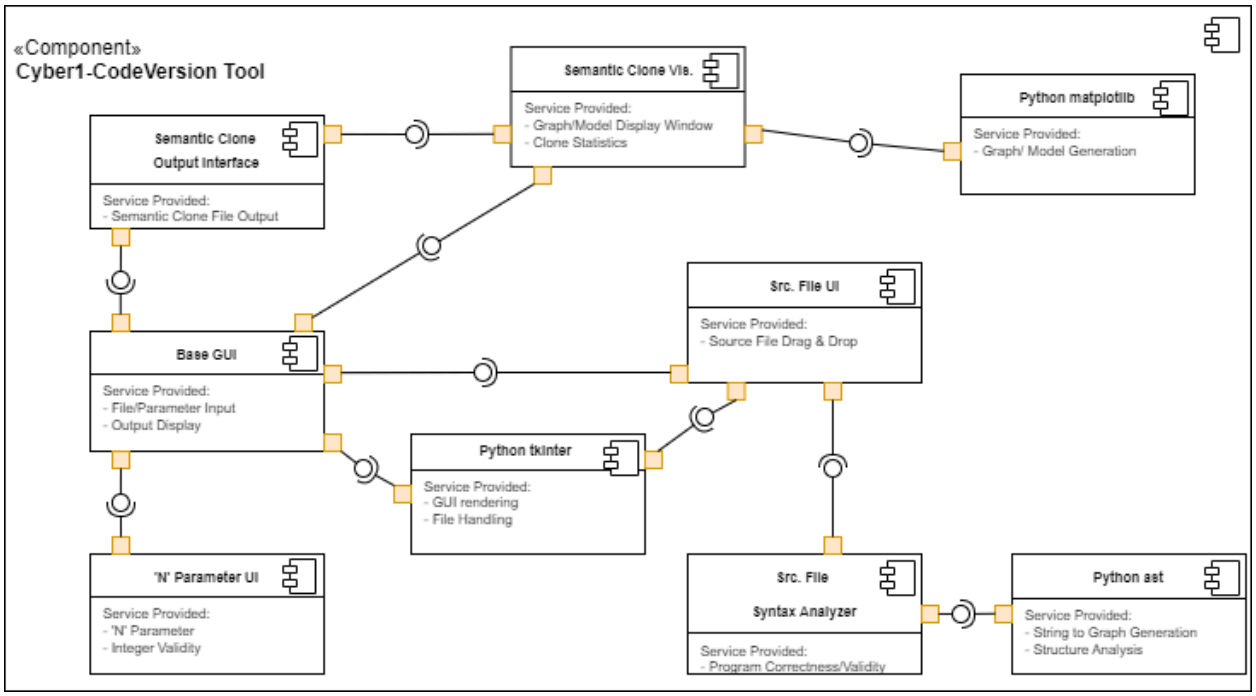


Figure III.3.1: Cyber1-CodeVersion Component Diagram

III.3.2 Subsystem Decomposition

The Subsystem Decomposition section provides a detailed report of all of the significant components and subsystems working under the hood of the Cyber1-CodeVersion tool. Information provided about each component consists of the name of the component, a description of what the component does in the system, concepts about how this component is integrated into the tool, and a description of the required and provided interfaces of the subsystem.

[SubSys-1] Base GUI

Description

The Base Graphical User Interface (GUI) provides users with the primary point of interaction, allowing for input file selection, parameters for code clone generation, and viewing the process results. It presents forms, buttons, and visualizations to help cybersecurity experts interpret data outputs.

Concepts and Algorithms Generated

The Base Graphical User Interface provides a feature that is responsible for the central page of input parameters such as the ‘N’ parameter and file input. Without the GUI, the app cannot function, and navigating software would be difficult. The focus is not particularly making a complex GUI, but a simple way to interact with buttons to produce complex visualizations.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
File/Parameter Input	Source Clone Output UI	Allows for input file selection and parameter input.
Output Display	Semantic Clone Vis.	Displays process results in the GUI.

Figure 1: Provided Services

Service Name	***Service Provided From***
Source File Drag and Drop	Source File User Interface
GUI Rendering	Python “tkinter” package
‘N’ Parameter	‘N’ Parameter User Interface

Figure 2: Required Services

[SubSys-2] Semantic Clone Output Interface

Description

This subsystem is responsible for handling the output of the semantic clone detection process. It takes the analysis results and makes them available for further processing or visualization in the GUI.

Concepts and Algorithms Generated

The Semantic Clone Output Interface provides a feature which is responsible for processing the output generated by the non-deterministic algorithm and preparing it for further use by other subsystems, such as the Semantic Clone Visualization. After users complete a full process of

generating new semantic clones, the output should be displayed in an organized format such as a flow of files branching from a singular source file.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Semantic Clone File Output	Base GUI, Semantic Clone Visualization	Outputs the semantic clone output for display.

Figure 1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Figure 2: Required Services

[SubSys-3] 'N' Parameter User Interface

Description

The Base Graphical User Interface (GUI) provides users with the ability to enter an integer into a text box to be evaluated later for Semantic Clone Generation. It validates the input and ensures it is applied correctly to the analysis process.

Concepts and Algorithms Generated

The ‘N’ parameter UI provides a feature that is responsible for generating the ‘N’ number of semantic clones to the Semantic Clone Output UI. Once a user enters an integer number, that number is checked if it is above a certain threshold and prompts the user if the number is too big or if it is not valid.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***

‘N’ parameter	Base GUI	Allows integer input for 'N' parameter selection.
Integer Validity	Base GUI	Validates the integer input for 'N' parameter.

Figure 1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Figure 2: Required Services

[SubSys-4] Python “tkinter” package

Description

The Python "tkinter" package provides users with a graphical interface for interacting with the tool. It handles the rendering of visual components like buttons, text boxes, and file drop boxes.

Concepts and Algorithms Generated

The Python "tkinter" package provides a feature which is used to manage the graphical user interface components for the Cyber1-CodeVersion tool. This subsystem generates various interactive elements, such as buttons, text fields, and file drop boxes. Tkinter handles the integration of the Base GUI and other components, managing events such as file drag-and-drop or button clicks. The system ensures that user interactions flow smoothly, from file uploads to viewing clone results.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
GUI Rendering	Base GUI	Renders the GUI components.
File Handling	Source File UI	Handles file interactions within the GUI.

Figure 1: Provided Services

Service Name	***Service Provided From***

N/A	N/A

Figure 2: Required Services

[SubSys-5] Python Source File User Interface

Description

The Python Source File User Interface provides users with drag-and-drop or local file selection for easy file input. It validates the uploaded files and ensures they are ready for analysis.

Concepts and Algorithms Generated

The Python Source File User Interface subsystem provides a feature that is an intuitive drag-and-drop or local file selection functionality for loading Python source files into the system for clone detection. The system uses algorithms to monitor file input events and validate file formats and types before they are passed along for further processing. Additional error-handling algorithms are included to ensure that only valid files are submitted and that users are notified of any issues with the files during upload such as the Syntax Analyzer component,

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Source File Drag and Drop	Base GUI	Enables drag-and-drop functionality for file input.

Figure 1: Provided Services

Service Name	***Service Provided From***
File Handling	Python “tkinter” package

Figure 2: Required Services

[SubSys-6] Source File Syntax Analyzer

Description

The Source File Syntax Analyzer provides users with syntax validation for their source code before clone detection. It ensures only syntactically correct code is passed on for analysis.

Concepts and Algorithms Generated

The Source File Syntax Analyzer provides a feature that ensures that source code files are syntactically correct before they are analyzed for semantic clones. This subsystem utilizes the Python “ast” to validate its syntax against predefined programming language rules. If any syntax errors are found, the system generates error messages to prompt the user to correct the issues. The subsystem guarantees that only valid code is processed by the file handler, which prevents unnecessary errors during the semantic clone generation.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Program Correctness/Validity	Python "ast" package, Source File UI	Validates the syntax of the input files before analysis.

Figure 1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Figure 2: Required Services

[SubSys-7] Python “ast” package

Description

The Python "ast" package provides users with a way to convert their Python code into an abstract syntax tree (AST) for structural analysis. It enables semantic comparisons by breaking down the code into its logical components.

Concepts and Algorithms Generated

The Python "ast" package is used to translate Python code into a tree structure that represents the logical components of the code. The primary algorithms focus on parsing the source code and converting it into an abstract syntax tree, which simplifies the comparison process used in clone detection. This subsystem is essential for transforming the code into a format that enables semantic clone comparison to see if clones are structurally different but functionally similar.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description

of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
String to Graph Generation	Source File Syntax Analyzer	Converts source code into graphs for analysis.
Structure Analysis	Source File Syntax Analyzer	Checks the program structure to see if any syntax errors exist.

Figure 1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Figure 2: Required Services

[SubSys-8] Semantic Clone Visualization

Description
The Semantic Clone Visualization provides users with graphical representations of semantic clone files. It helps users visualize the relationships between similar code fragments using models and metrics.

Concepts and Algorithms Generated
The Semantic Clone Visualization provides a feature that is responsible for displaying the relationships between code fragments identified as clones. Using graph generation algorithms, this subsystem produces visual models that represent the structural similarities between different sections of code. The visual output is designed to help cybersecurity professionals quickly interpret complex clone relationships, making tasks such as identifying potential vulnerabilities or refactoring code much more efficient.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Graph/Model Display	Base GUI, Python	Displays clone

Window	"matplotlib" package	relationships graphically.
Clone Statistics	Semantic Clone Output Interface	Displays clone numerics such as runtime, similarity %, etc.

Figure 1: Provided Services

Service Name	***Service Provided From***
Semantic Clone Output Interface	Semantic Clone Output Interface

Figure 2: Required Services

[SubSys-9] Python “matplotlib” package

Description

The Python "matplotlib" package provides users with graph and model generation capabilities to display the results of the Python semantic clone files. It allows end users to efficiently interpret files without looking over them with the human eye.

Concepts and Algorithms Generated

The Python "matplotlib" package is used to generate graphs and charts that visually represent the results of the semantic clone files. This subsystem focuses on algorithms for plotting data and generating clear and informative visualizations to show code clone statistics. The visual output generated by Matplotlib is one of the priorities for our tool to help cybersecurity professionals.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Figure 1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Figure 2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Graph/Model Generation	Semantic Clone Visualization	Generates visual graphs for clone representation.

Figure 1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Figure 2: Required Services

III.4 Data Design

The program will generate semantic clones by manipulating an abstracted syntax tree representation of the source code. For each permutation, they will have a corresponding syntax tree permutation in memory before being converted back to source code and saved. Tracking each permutation can be done simply, such as via a linked list or array.

The syntax tree data structure itself is the most complex that will be used in the program, although as development continues other data structures may be introduced to optimize clone generation. For our program, we have settled on the auxiliary Python library LibCST. LibCST creates a unique syntax tree that saves the comments and other formatting present in the source code, allowing for it to be converted back into functioning Python code [4]. This functionality is not present in Python's natively supported Abstract Syntax Trees and is thus called Concrete Syntax Trees. LibCST was created with the refactoring of Python code in mind, so the source code is preserved and the syntax tree is both readable and one-to-one.

The program runs on the user's system and memory is only constrained by the size of the source code file, the number of permutations to create, and the system's memory for clone generation. No online or cloud functionality will be implemented for the deliverable product.

III.5 User Interface Design

The user interface will go through several iterations during development until the final prototype is ready, as the program will be coded in Python. The GUI will be limited to the terminal for most of the development, with the focus being on enough functionality for the developers to test the generation algorithm.

The first style will be a simple importing file button on the Google Colab file the developers will be working on. The Colab file will be downloadable and able to run in the terminal, but the visual functionality will still be limited since it will be in early development.

When the prototype is available as a user-friendly executable, the accompanying GUI will allow the user to input files, view generation progress, and download the output without needing the terminal. Installation will involve downloading the executable program and running it as described in the Requirements and Specifications section earlier (section II). The Windows operating system will be prioritized for the functional executable, with support for common Linux distributions like Ubuntu as a stretch goal. For users unable to download and run the executable, the original Python file will be available for download and running in the terminal, with the Python file being functional regardless of platform. The breakdown of the primary parts of the GUI will be explained below, with rough designs subject to change.

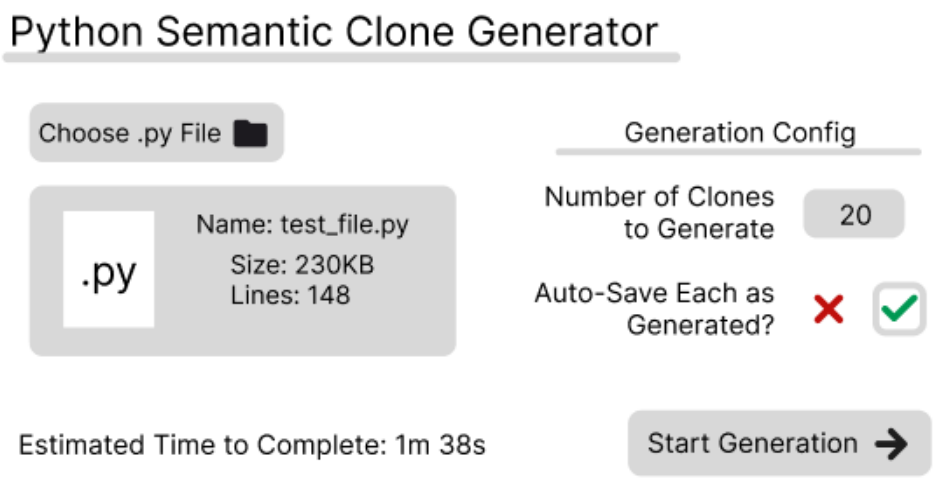


Figure III.5.1: Rough Input Stage of Program GUI

Above is a rough mockup of the configuration and input stage, corresponding to use cases Input Python Code Files [UC-3] and Input 'N' Parameter [UC-4]. The user can choose to input a python file, which opens a folder browser that then determines if the chosen file is a valid python file. The details of the file, to make sure it is the correct one the user wants to generate clones from, will be displayed on the screen. The user will also be able to specify the amount 'N' of clones to generate as well as whether they want each clone to be saved to storage instead of just in temporary memory. The setting for where clones are saved automatically may become an extra option that can be configured differently for each time the user wants to run the program, or may be a universal default setting. The estimated "Time to Complete" display may not be included depending on how intensive or inaccurate the implementation is, as it will be factoring in the program's generation speed on that system, how many clones to generate, and the complexity of the source file. Once the user decides to begin generating clones, the program will proceed to the screen below.

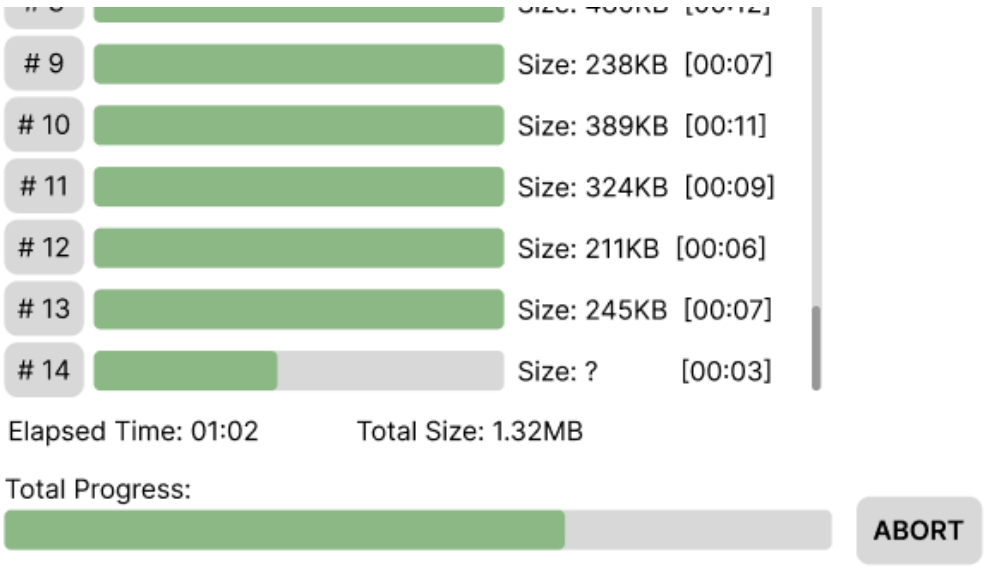


Figure III.5.2: Rough Generation Stage of Program GUI

Above is a rough mockup of the generation stage, corresponding to Generate Semantic Clones [UC-5] in the Requirements section. Depending on the program's implementation, there is a lot of potential overlap between the generation stage's GUI and the output statistics. The primary focus will be on generating the clones one by one, with the autosave feature, if activated previously, backing up each clone to storage on top of the default temporary memory. Since the statistics for each clone generated are saved during that round, the data is available as soon as the generation is completed, so a separate statistics page may not be necessary. An "abort" or "cancel" button on the generation will be important, so that the user has control in case generation is taking too long, the current amount is satisfactory, or the wrong file was used.

IV. Testing and Acceptance Plans

IV.1 Introduction

The Cyber1-CodeVersion tool is designed for cybersecurity experts to efficiently analyze how pieces of code and malware change over time. As resources and time for adversaries grow significantly every year, new forms of malicious code may seem obscure to the human eye which delays the process of counteracting attempts to gain access to critical systems. In order to produce a reliable tool for the cybersecurity professionals to utilize, software testing is a key component to ensuring success. In cybersecurity, time is critical and lackluster testing of software can cause a lot more damage than just money. This section aims to equip the user of the functional unit tests that are performed in the Cyber1-CodeVersion tool.

IV.1.1 Project Overview

The Cyber1-CodeVersion tool requires many dependencies in order to function. From a base level, the tool requires input from the user such as an 'N' semantic clone parameter to define how many clones will be produced from a source 'seed' file, as well as a Python file populated with any type of source code. Given these parameters, the core functionality or output of this tool is to provide the user with visualizations using the "matplotlib" Python library. These visualizations provide the cybersecurity professional with insightful statistics and graphs that represent the similarity between clone samples and a source file. With these visualizations, a more efficient process of analyzing multiple types of source code is created. The tool is almost completely Python based which utilizes libraries such as "tkinter" for file handling, "matplotlib" for graphing and visualization, "libcst", and "ast" for syntax tree building.

Normally, users would go line by line and check the functionality of a program, but this takes time and is not efficient. Overall, the tool is a software solution designed for cybersecurity experts, enabling them to analyze Python source code by creating semantic clones, which are modified versions of a seed source code file with equivalent functionality. This functionality helps in identifying variations in code while maintaining the underlying bit logic. This tool serves a great purpose for the realm of cybersecurity because it automates processes that are very time consuming, while also incorporating the human aspect through user input and code analysis. Ultimately, we are not letting the tool make the decisions. The decisions are made completely by the user, the tool just presents the statistical facts to the user for better insights and increased

reliability. To summarize, a one-line objective for this tool is to enable faster insights on code analysis while preserving user control over decisions.

The major functionalities of the Cyber1-CodeVersion tool that require testing include:

- User Input Validation (Python files, ‘N’ Parameter)
- Python Library Dependency Check
- Python File Code Syntax Analysis (Abstract Syntax Tree)
- GUI/Visualization Component Rendering
- Concrete Syntax Tree Validation
- Clone Similarity Analysis
- File Saving and Management

IV.1.2 Test Objectives and Schedule

Testing will be done during development of the larger program, due to the strict deliverable deadline at the end of the semester. This ensures that whatever phase of the program we are able to reach will be just as functional and robust as earlier parts, instead of a broad application that is very fragile during use.

The program focuses on manipulating Python files so that they read differently but have identical functionality, so bug-testing will focus on being able to support as many inputs as possible. Due to Python’s flexibility in typing, definitions, and operations, several edge-cases need to be considered. One example are function names being defined during an ‘import as’ statement, instead of during the more common function ’def.’ The deliverables for testing-related features, along with the default application deliverable, will be crafted Python files that serve as specific edge cases, as well as automated tests using these files to help speed up testing.

Major milestones will focus on developing an algorithm to change a specific feature or node type in the CST, with testing being the final part before that algorithm is declared functional and finished.

IV.1.3 Scope

The purpose of this section of the report is to equip the user with how testing will be conducted for the functional requirements of the Cyber1-CodeVersion. While the team will not cover one-to-one input and output expectations, we aim to provide a clear framework of the flow of the development process. More details for the testing plan of different components of the tool are provided below.

IV.2 Testing Strategy

Continuous Integration will be the main methodology used, as that allows for more manual oversight in development. Since our program itself is simplistic in design, the automated Github

tests will focus more on the user-access end, such as testing whether it can still operate on the desired platforms after changes have been made to the GUI. The testing approach will be done as follows:

1. For the current feature or requirement being developed, identify any possible vulnerabilities or edge-cases that can be encountered through user interaction or the given Python input file.
2. Develop test cases for these vulnerabilities.
3. Review the test results and for any failures, either decide if the given issue is too complex to fix and may be left or how to fix the vulnerability. If other vulnerabilities are discovered during this phase, integrate into testing.
4. Repeatedly test until all necessary tests are successful.
5. Once the feature or requirement has been verified via testing, it is integrated into the larger application.
6. Move on to the next feature.

IV.3 Test Plans

The Cyber1-CodeVersion tool will undergo a variety of different testing types from a range of specific tool functional requirements as well as general app usage. This section aims to equip the user with the unique categorizations of testing techniques to ensure the efficiency and reliability of smooth tool usage and execution. Under each type of specific testing such as unit, integration, functional, performance and user acceptance testing, is a detailed description of how the team will implement the flow of tests for the required functionalities.

IV.3.1 Unit Testing

While the Cyber1-CodeVersion tool isn't built off of a pre-existing framework and almost entirely Python based, the team will conduct unit tests by separating the functional requirements into categories and testing the individual features. For the functionalities included in *IV.1.1 Project Overview*, we will conduct tests per feature. An example of one test can be as follows; A user decides to import a .csv file to the Python file drop box. An expected output of this transaction would result in an error, notifying the user that only .py files must be accepted. Tests such as the file example will be embedded into our program anyways to serve as a "user input validation" feature. The idea behind this form of unit testing ensures that each individual component works as intended. When the team transitions to the integration testing, any errors presenting themselves will be an easy fix because the components are not tightly coupled. Some Python testing libraries that will be utilized are "Tkintertest" and "unittest".

IV.3.2 Integration Testing

By clustering certain components of our Cyber1-CodeVersion tool such as the Python source code file input to the CST creation, the team is able to ensure smooth interactions among the

tool's core functionalities. By combining multiple components to test their respective interactions with each other, the team is able to identify new bugs and develop solutions to the errors. This process will continue to cycle until all components are accounted for and interacting with each other to form the system testing stage. Since designing the GUI components is placed with the least priority in the development stage, it will be a challenge for the team to test GUI to core functional requirements such as non-deterministic algorithm generation. The importance for this stage is to make sure that the core functionalities are working, so we can efficiently make it visually appealing for cybersecurity professionals to use. Overall, GUI components won't interrupt the algorithm aspect of the tool which will allow for better prioritization of significant requirements.

IV.3.3 System Testing

The system testing phase consists of three different areas of testing. The three areas are functional, performance, and user acceptance testing. The overarching goal of the system testing is to allow developers to utilize the Cyber1-CodeVersion tool as a normal consumer such as a cybersecurity professional. If the developer encounters any flaws or abnormal functionality in tool execution, it is the team's duty to fix those issues and ensure the program works as intended. The three categories of testing below contain high level explanations of the process behind integrating each test into the fully functioning environment of the Cyber1-CodeVersion tool.

IV.3.3.1 Functional Testing

For functional testing, as stated in the Testing Strategy, tests will be manually created for each requirement as they are being developed. For example, the automatic function renaming scheme has a crafted Python file that uses several different ways for a function name to be defined, to make sure that no function names are missed that can compromise program functionality. Due to this program being developed with generating clones from malicious software in mind, we have to be prepared for obfuscation already coming in the program that we have to track.

IV.3.3.2 Performance Testing

The primary part of the application to be performance intensive will be the semantic clone generation itself, as the time to process depends on both the length or complexity of the Python file as well as how many clones the user specifies. To test the performance, both of these variables will be changed as well as the system itself. Due to the developing team's limited hardware, the more intensive code generation will be tested on higher-end personal pc's or on google Colab. Virtual machines may be used to emulate lower end systems and manually limit processing power.

IV.3.3.3 User Acceptance Testing

User acceptance testing will focus on the ease of use for the GUI and setup of the program on different systems. For this, the developers will test the program, such as

seeing if the required libraries can be easily installed to run the Python script in the command line, before moving on to the GUI itself. Once the GUI or program is in a usable state, the developers plan to share the repository with voluntary test users who will attempt to set up the program themself. Any issues encountered will be documented and worked on, such as a user finding the program cannot run on their setup or installing it presents errors. The developers will also use Python files not created for this project, such as old homework assignments, to test the functionality. The volunteers will be encouraged to do the same.

IV.4 Environment Requirements

The **necessary** properties of the test environment are as follows:

- **Facilities:** Isolated workstation/lab that does not have any important external files. Operates with the minimum requirements to run the Cyber1-CodeVersion tool.
- **Hardware:**
  - **CPU**: Dual-core processor
  - **RAM**: 8 GB
  - **DISK**: 100 GB
  - **DISPLAY**: Two 21''-27'' monitors for clearer representation of semantic clone generation and graphs/statistics.
- **System Software:**
  - Windows/Ubuntu 22.04 Operating Systems
  - pip package installer
  - Python 3.8 or higher, with all required dependencies pre-installed (e.g., libcst, tkinter, matplotlib)
- **Mode of Usage:** To access the Cyber1-CodeVersion tool, initial network connection to download Google Colab file in cloud environment. After the file is downloaded and all dependencies installed, no network access is needed and should run securely on a local workstation.
- **Testing Tools:**
  - **Testing Frameworks**: "unittest" for automated unit and integration tests.
  - **GUI Testing**: "Tkintertest" for testing GUI components.
- **Other:**
  - **GitHub**: Branch control, pull requests, documentation, etc.
  - **Markdown**: Language for documenting and code reports.

The **desired** properties of the test environment are as follows:

- **Facilities:** Isolated workstation/lab that does not have any important external files. Some secure source of access to cloud environments with higher security.
- **Hardware:**
    - **CPU**: Quad-core processor
    - **RAM**: 32 GB
    - **DISK**: 200 GB
    - **DISPLAY**: Dual 21"–27" monitors, or potentially a higher-resolution monitor for more detailed visualization and analysis.
- **System Software:**
    - Windows/Ubuntu 22.04 Operating Systems
    - pip package installer
    - Python 3.10 with all required dependencies pre-installed (e.g., libcst, tkinter, matplotlib)
- **Mode of Usage:** To access the Cyber1-CodeVersion tool, initial network connection to download Google Colab file in cloud environment. If in secure space, access to GPT API for variable/function renaming in source code.
- **Testing Tools:**
    - **Testing Frameworks**: “unittest” or “pytest” for automated unit and integration tests.
    - **GUI Testing**: “Tkintertest” for testing GUI components.
- **Other:**
    - **GitHub**: Branch control, pull requests, documentation, etc.
    - **Markdown**: Language for documenting and code reports.

V. Alpha Prototype

V.1 Introduction

Malware detection works through various different methods, with one of the most popular being a signature-based system. This signature-recognition system compares a given file to a database of known malware to find matches, so that once malicious software is identified, it is not allowed access to the system. However, the drawback is that simple obfuscation methods can render malware unknown to the detection software and allow it to pass unhindered.

As such, the goal of this project over the academic year is to create an algorithm to generate permutations of a given source code file that accomplish the same function, so it can be integrated into databases to better identify evolving malware. Multiple obfuscation techniques will be tested, such as simply changing the variable names and moving code blocks around, to changing the flow of the program to something unique from the original.

For the purposes of this document, only the first semester's implementation will be examined, where Concrete Syntax Trees are used. Syntax trees are a specific representation of a given program's behavioral structure through the lenses of individual chunks of code, such as functions leading to the various possible outputs or subsequent actions. This allows for traversing through the program or viewing a specific branch of it. The Concrete Syntax Tree library, LibCST, contains the ability to also convert these trees back into source code while preserving white space and comments [4].

The purpose of the Alpha Prototype section is to detail the progress made throughout the first semester on this implementation, what was left uncompleted and why, and then what future plans for next semester are.

V.2 Team Members - Bios and Project Roles

Bryan Frederickson is a Computer Science student at Washington State University. Some of his interests include Networking, Cybersecurity, and C programming. For the Cyber1- CodeVersion, he was responsible for implementing an AST Python analyzer for syntax validity and the variable renamer functionality utilizing a CST transformer/visitor function. Some of his previous projects include using an ML model for radio frequency classification and networking analysis between users and biomanufacturing iot devices living on a network.

Figure V.2.1: Photo of Bryan Frederickson.

Samantha Brewer is a Computer Science student at Washington State University. Some of her technical interests include Networking, Cybersecurity, and modding video games. For the Cyber1- CodeVersion, she was responsible for implementing a command line parser for UI, function renamer functionality utilizing a CST transformer/visitor function, and updating documentation for prerequisites required to run the tool. A previous project of hers is C/C++ to LLVM Intermediate representation via Bash shell scripts.

Figure V.2.2: Picture of Samantha Brewer

V.3 Project Requirements

The project requirements section includes the main features planned for the CST-based prototype implementation, with a breakdown of how they were created and their limitations. These requirements were created with the exception of a full GUI being implemented, however throughout development, our client expressed that a GUI is not expressly necessary. However, the general use-case is planned for our current command line-based prototype application.

V.3.1 Functional Requirements

The boxes below represent functional requirements from our client with information such as the function requirement title, the description of the functional requirement, the source of the functional requirement, and the priority level of the functional requirement.

1. User Input and File Handling

Source Code File Box [FR-1.1]	
Description	The application needs to contain a section/box labeled with a logo of a file, that allows the user to either drag and drop or select a specific file from their local machine to be analyzed by the tool. This file drop box should only accept Python source code files.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, learning and analyzing the source code files is a requirement to understand how malware changes. This interface allows the professional to select which file to analyze.
Priority	Priority Level 0: Essential and required functionality.

‘N’ Semantic Clones Parameter [FR-1.2]	
Description	The application needs to contain a text box that accepts user input of an integer. This integer represents the number of semantic clones

	that will be produced from the selected source file by the cybersecurity professional. This integer should not be super large in order to maintain the quick runtime of the application.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, the more branches we produce of a source code file, allow for a better understanding of the diverse ways code can change over time. This user input allows the professional to create different variations of the original Python file.
Priority	Priority Level 0: Essential and required functionality.

Saves Clones Button [FR-1.3]	
Description	The application needs to contain a “Save Clones” button that opens the local machine file explorer application, which allows the user to select which directory the clones should be saved under. Clones should be able to be saved individually, as well as all together.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, if the user wants to do a more intensive investigation on individual source code files, they should be able to download them on their local machine.
Priority	Priority Level 1: Desired functionality.

2. Semantic Clone Generation

Generate Clones Button [FR-2.1]	
Description	The application needs to contain a “Generate Clones” button that effectively creates an ‘N’ number of semantic clone files stemming from the original Python file. These clones all need to be diverse from one another to enforce the rule of non-determinism.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, obtaining the clones is a key requirement to see how source code changes over time. This button is a necessity to obtain these clones.
Priority	Priority Level 0: Essential and required functionality.

Semantic Clone Output Box [FR-2.2]

Description	The application needs to contain an additional file box that is populated with semantic clone files once they are done generating from the source code. This box should be able to allocate up to ‘N’ specified spots which are chosen by the user.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, it is ideal to be able to see the actual clones that have been generated. This output box allows you to see all the permutations that were created after the semantic clone generation process.
Priority	Priority Level 0: Essential and required functionality.

Summarize Clones Button [FR-2.3]	
Description	The application needs to contain a button that allows users to summarize statistics such as runtime, time complexity, and storage that each semantic clone takes up in memory. These statistics help cybersecurity experts analyze the behavior of how source code changes.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods for this requirement. As a cybersecurity professional, instead of individually analyzing source code files, bringing these relevant statistics to the user allows for a more efficient malware analysis process.
Priority	Priority Level 1: Desired functionality.

V.3.2 Non-Functional Requirements

Python Exclusivity [NFR-1]:

For the first implementation of the program for our research project, Python will be used as it has a native and robust library for Abstract Syntax Trees, however, it only supports parsing Python files.

Multi-Platform Usability [NFR-2]:

As Python is not compiled into an executable, the code can be parsed regardless of the source platform. The user program will likely be a Jupyter Notebook file which can be run in a web browser.

English-Only Support [NFR-3]:

Only English will be supported for parsing, as the developers are only familiar with English, and finding source files in other human languages will not be feasible.

Robust Memory Size [NFR-4]:

As users can specify the number of permutations, the program would need a large amount of space for parsing and creating unique output as the number requested increases.

Quick Response Time [NFR-5]:

As the AST library is natively supported in Python, the parsing time should take only as long as compile time, while the creating permutations are dependent on the amount requested. Using Jupyter Notebook will allow the user to rely on much larger memory for processing.

Multi-File Support [NFR-6]:
As many Python files rely on external files or libraries, to fully understand the function of the code for semantic clone generation, processing all external dependencies will be necessary. An option to select a folder or a single file may be one implementation, running on the assumption that the user has those libraries included.

V.4 Solution Approach

The solution approach section includes the system decomposition and functional requirements of the Cyber1- CodeVersion tool. These base requirements were used to form the overall architecture and help the team define dependencies between the functional requirements. As stated in the functional requirements section, the GUI was put in the backlog for development for the next development cycle, but will still be incorporated as part of the final tool product. Below, the information provided is a description of what a functional component does in the system, concepts about how this component is integrated into the tool, and a description of the required and provided interfaces of the subsystem.

[SubSys-1] Base GUI

Description

The Base Graphical User Interface (GUI) provides users with the primary point of interaction, allowing for input file selection, parameters for code clone generation, and viewing the process results. It presents forms, buttons, and visualizations to help cybersecurity experts interpret data outputs.

Concepts and Algorithms Generated

The Base Graphical User Interface provides a feature that is responsible for the central page of input parameters such as the ‘N’ parameter and file input. Without the GUI, the app cannot function, and navigating software would be difficult. The focus is not particularly making a complex GUI, but a simple way to interact with buttons to produce complex visualizations.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 1.1* shows what services the subsystem provides, who the service is provided to, as well as the description

of the subsystem. *Table 1.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
File/Parameter Input	Source Clone Output UI	Allows for input file selection and parameter input.
Output Display	Semantic Clone Vis.	Displays process results in the GUI.

Table 1.1: Provided Services

Service Name	***Service Provided From***
Source File Drag and Drop	Source File User Interface
GUI Rendering	Python “tkinter” package
‘N’ Parameter	‘N’ Parameter User Interface

Table 1.2: Required Services

[SubSys-2] Semantic Clone Output Interface

Description

This subsystem is responsible for handling the output of the semantic clone detection process. It takes the analysis results and makes them available for further processing or visualization in the GUI.

Concepts and Algorithms Generated

The Semantic Clone Output Interface provides a feature which is responsible for processing the output generated by the non-deterministic algorithm and preparing it for further use by other subsystems, such as the Semantic Clone Visualization. After users complete a full process of generating new semantic clones, the output should be displayed in an organized format such as a flow of files branching from a singular source file.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 2.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 2.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	Service Provided To	Description
Semantic Clone File Output	Base GUI, Semantic Clone Visualization	Outputs the semantic clone output for display.

Table 2.1: Provided Services

Service Name	Service Provided From
N/A	N/A

Table 2.2: Required Services

[SubSys-3] 'N' Parameter User Interface

Description

The Base Graphical User Interface (GUI) provides users with the ability to enter an integer into a text box to be evaluated later for Semantic Clone Generation. It validates the input and ensures it is applied correctly to the analysis process.

Concepts and Algorithms Generated

The ‘N’ parameter UI provides a feature that is responsible for generating the ‘N’ number of semantic clones to the Semantic Clone Output UI. Once a user enters an integer number, that number is checked if it is above a certain threshold and prompts the user if the number is too big or if it is not valid.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 3.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 3.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	Service Provided To	Description
‘N’ parameter	Base GUI	Allows integer input for 'N' parameter selection.
Integer Validity	Base GUI	Validates the integer input for 'N' parameter.

Table 3.1: Provided Services

Service Name	Service Provided From
N/A	N/A

Table 3.2: Required Services

[SubSys-4] Python “tkinter” package

Description

The Python "tkinter" package provides users with a graphical interface for interacting with the tool. It handles the rendering of visual components like buttons, text boxes, and file drop boxes.

Concepts and Algorithms Generated

The Python "tkinter" package provides a feature which is used to manage the graphical user interface components for the Cyber1-CodeVersion tool. This subsystem generates various interactive elements, such as buttons, text fields, and file drop boxes. Tkinter handles the integration of the Base GUI and other components, managing events such as file drag-and-drop or button clicks. The system ensures that user interactions flow smoothly, from file uploads to viewing clone results.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 4.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 4.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
GUI Rendering	Base GUI	Renders the GUI components.
File Handling	Source File UI	Handles file interactions within the GUI.

Table 4.1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Table 4.2: Required Services

[SubSys-5] Python Source File User Interface

Description

The Python Source File User Interface provides users with drag-and-drop or local file selection for easy file input. It validates the uploaded files and ensures they are ready for analysis.

Concepts and Algorithms Generated

The Python Source File User Interface subsystem provides a feature that is an intuitive drag-and-drop or local file selection functionality for loading Python source files into the system for clone detection. The system uses algorithms to monitor file input events and validate file formats and types before they are passed along for further processing. Additional error-handling algorithms are included to ensure that only valid files are submitted and that users are notified of any issues with the files during upload such as the Syntax Analyzer component,

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 5.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 5.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Source File Drag and Drop	Base GUI	Enables drag-and-drop functionality for file input.

Table 5.1: Provided Services

Service Name	***Service Provided From***
File Handling	Python “tkinter” package

Table 5.2: Required Services

[SubSys-6] Source File Syntax Analyzer

Description

The Source File Syntax Analyzer provides users with syntax validation for their source code before clone detection. It ensures only syntactically correct code is passed on for analysis.

Concepts and Algorithms Generated

The Source File Syntax Analyzer provides a feature that ensures that source code files are syntactically correct before they are analyzed for semantic clones. This subsystem utilizes the Python “ast” to validate its syntax against predefined programming language rules. If any syntax errors are found, the system generates error messages to prompt the user to correct the issues. The subsystem guarantees that only valid code is processed by the file handler, which prevents unnecessary errors during the semantic clone generation.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 6.1* shows what services the subsystem provides, who the service is provided to, as well as the description

of the subsystem. *Table 6.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Program Correctness/Validity	Python "ast" package, Source File UI	Validates the syntax of the input files before analysis.

Table 6.1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Table 6.2: Required Services

[SubSys-7] Python “ast” package

Description

The Python "ast" package provides users with a way to convert their Python code into an abstract syntax tree (AST) for structural analysis. It enables semantic comparisons by breaking down the code into its logical components.

Concepts and Algorithms Generated

The Python "ast" package is used to translate Python code into a tree structure that represents the logical components of the code. The primary algorithms focus on parsing the source code and converting it into an abstract syntax tree, which simplifies the comparison process used in clone detection. This subsystem is essential for transforming the code into a format that enables semantic clone comparison to see if clones are structurally different but functionally similar.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 7.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 7.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
String to Graph Generation	Source File Syntax Analyzer	Converts source code into graphs for analysis.
Structure Analysis	Source File Syntax Analyzer	Checks the program structure to see if any syntax errors exist.

Table 7.1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Table 7.2: Required Services

[SubSys-8] Semantic Clone Visualization

Description

The Semantic Clone Visualization provides users with graphical representations of semantic clone files. It helps users visualize the relationships between similar code fragments using models and metrics.

Concepts and Algorithms Generated

The Semantic Clone Visualization provides a feature that is responsible for displaying the relationships between code fragments identified as clones. Using graph generation algorithms, this subsystem produces visual models that represent the structural similarities between different sections of code. The visual output is designed to help cybersecurity professionals quickly interpret complex clone relationships, making tasks such as identifying potential vulnerabilities or refactoring code much more efficient.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 8.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 8.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Graph/Model Display Window	Base GUI, Python "matplotlib" package	Displays clone relationships graphically.
Clone Statistics	Semantic Clone Output Interface	Displays clone numerics such as runtime, similarity %, etc.

Table 8.1: Provided Services

Service Name	***Service Provided From***
Semantic Clone Output Interface	Semantic Clone Output Interface

Table 8.2: Required Services

[SubSys-9] Python “matplotlib” package

Description

The Python "matplotlib" package provides users with graph and model generation capabilities to display the results of the Python semantic clone files. It allows end users to efficiently interpret files without looking over them with the human eye.

Concepts and Algorithms Generated

The Python "matplotlib" package is used to generate graphs and charts that visually represent the results of the semantic clone files. This subsystem focuses on algorithms for plotting data and generating clear and informative visualizations to show code clone statistics. The visual output generated by Matplotlib is one of the priorities for our tool to help cybersecurity professionals.

Interface Description

The two tables below showcase a detailed description of the tool subsystem. *Table 9.1* shows what services the subsystem provides, who the service is provided to, as well as the description of the subsystem. *Table 9.2* shows what services the subsystem requires as well as from whom the service is provided.

Service Name	***Service Provided To***	***Description***
Graph/Model Generation	Semantic Clone Visualization	Generates visual graphs for clone representation.

Table 9.1: Provided Services

Service Name	***Service Provided From***
N/A	N/A

Table 9.2: Required Services

V.5 Test Plan

The test plan section goes over the different testing approaches the development team planned to go through for testing the CST application’s implementation, especially in regards to edge cases and user accessibility.

V.5.1 Unit Testing

While the Cyber1-CodeVersion tool isn't built off of a pre-existing framework and almost entirely Python based, the team will conduct unit tests by separating the functional requirements into categories and testing the individual features. For the functionalities included in *IV.1.1 Project Overview*, we will conduct tests per feature. An example of one test can be as follows; A user decides to import a .csv file to the Python file drop box. An expected output of this transaction would result in an error, notifying the user that only .py files must be accepted. Tests such as the file example will be embedded into our program anyways to serve as a "user input validation" feature. The idea behind this form of unit testing ensures that each individual component works as intended. When the team transitions to the integration testing, any errors presenting themselves will be an easy fix because the components are not tightly coupled. Some Python testing libraries that will be utilized are "Tkintertest" and "unittest".

V.5.2 Integration Testing

By clustering certain components of our Cyber1-CodeVersion tool such as the Python source code file input to the CST creation, the team is able to ensure smooth interactions among the tool's core functionalities. By combining multiple components to test their respective interactions with each other, the team is able to identify new bugs and develop solutions to the errors. This process will continue to cycle until all components are accounted for and interacting with each other to form the system testing stage. Since designing the GUI components is placed with the least priority in the development stage, it will be a challenge for the team to test GUI to core functional requirements such as non-deterministic algorithm generation. The importance for this stage is to make sure that the core functionalities are working, so we can efficiently make it visually appealing for cybersecurity professionals to use. Overall, GUI components won't interrupt the algorithm aspect of the tool which will allow for better prioritization of significant requirements.

V.5.3 System Testing

The system testing phase consists of three different areas of testing. The three areas are functional, performance, and user acceptance testing. The overarching goal of the system testing is to allow developers to utilize the Cyber1-CodeVersion tool as a normal consumer such as a cybersecurity professional. If the developer encounters any flaws or abnormal functionality in tool execution, it is the team's duty to fix those issues and ensure the program works as intended. The three categories of testing below contain high level explanations of the process behind integrating each test into the fully functioning environment of the Cyber1-CodeVersion tool.

V.5.3.1 Functional Testing

For functional testing, as stated in the Testing Strategy, tests will be manually created for each requirement as they are being developed. For example, the automatic function renaming scheme has a crafted Python file that uses several different ways for a function name to be defined, to make sure that no function names are missed that can

compromise program functionality. Due to this program being developed with generating clones from malicious software in mind, we have to be prepared for obfuscation already coming in the program that we have to track.

V.5.3.2 Performance Testing

The primary part of the application to be performance intensive will be the semantic clone generation itself, as the time to process depends on both the length or complexity of the Python file as well as how many clones the user specifies. To test the performance, both of these variables will be changed as well as the system itself. Due to the developing team's limited hardware, the more intensive code generation will be tested on higher-end personal pc's or on google Colab. Virtual machines may be used to emulate lower end systems and manually limit processing power.

V.5.3.3 User Acceptance Testing

User acceptance testing will focus on the ease of use for the GUI and setup of the program on different systems. For this, the developers will test the program, such as seeing if the required libraries can be easily installed to run the Python script in the command line, before moving on to the GUI itself. Once the GUI or program is in a usable state, the developers plan to share the repository with voluntary test users who will attempt to set up the program themself. Any issues encountered will be documented and worked on, such as a user finding the program cannot run on their setup or installing it presents errors. The developers will also use Python files not created for this project, such as old homework assignments, to test the functionality. The volunteers will be encouraged to do the same.

V.6 Alpha Prototype Description

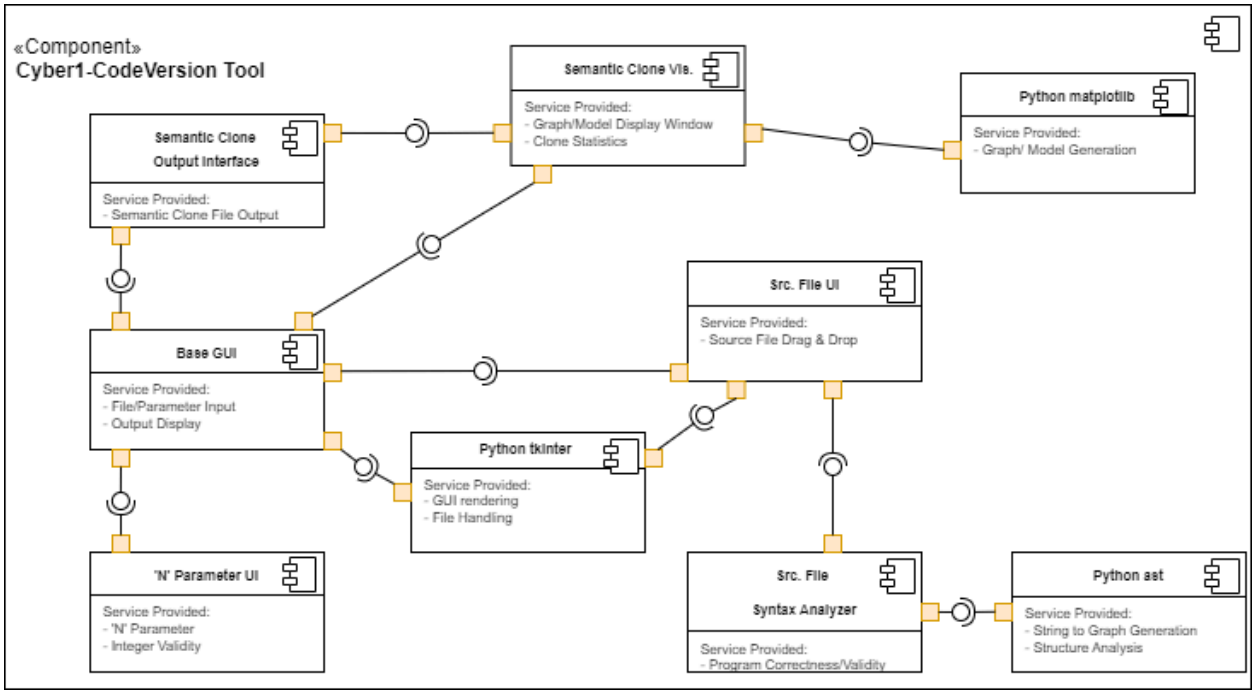


Figure V.6.1: Cyber1-CodeVersion Component Diagram

As a reference to the component diagram addressed earlier in this project report, this architecture was one of the key aspects that helped steer the functional requirements that needed to be developed by the team. For all of the UI related elements in the architecture, the proof of concept has been developed for how we want to display our analytical results for the semantic lone files. As far as physical results, our client requested that the functionality of the code be precedence over the GUI components. For the sprints in this section of the project, only a command line tool was implemented in terms of UI. For elements such as the Syntax analyzer, ast, and the algorithms produced, were able to be close to fully developed. These algorithms include aspects such as function and variable renaming to help contribute to the semantic clone generation component. This component diagram is subject to change and become more elaborate over the course of the development to highlight specifics within designing the semantic clone files.

V.6.1 UI & File Handler

V.6.1.1 Functions and Interfaces Implemented

```
PS C:\Users\bryan\Downloads> python .\prototype.py
Input the filepath for input Python file. In Windows file explorer, select the file then shift right click, then select
'Copy as path'. Paste into terminal using right click.
Filepath:
```

Figure V.6.1.1: Screenshot of Windows Command Prompt, with a Python file prompting for the filepath to input Python file.

The UI & File Handler's significance to the Cyber1- CodeVersion tool allows for user input which is one of the components with the height priority. Without a Python source file, no semantic clones can be generated. Figure V.6.1.1 shows the interface provided to the Cybersecurity professional for specifying a file path to be further analyzed by the AST.

V.6.1.2 Preliminary Tests

To test the UI & File Handler component, the team provided correct and invalid file paths on the local host machine to see if the specified user input paths could be recognized. If the path was valid the program could continue with execution, versus prompting the user for another path if the path was invalid.

V.6.2 Function Renamer

V.6.2.1 Functions and Interfaces Implemented

The Function Renamer significance to the Cyber1- CodeVersion tool allows for Python file obfuscation effectiveness. For Python files to appear different at a glance, but functionally produce the same result, names of variables must be changed in order to reduce suspicion of similar code entities. The function renamer works in conjunction with the variable renamer as one process but must be run after the variable renamer. The result of the process produces a distinct Python source file with changed function names covering a variety of edge cases.

V.6.2.2 Preliminary Tests

To test the Function renamer component, test files tailored towards function heavy code were utilized and created by the team. More edge cases are yet to be discovered but a majority and base line of basic usage has been covered and is fully functional. If edge cases were discovered, enhanced methods were implemented within the subclass transformer to cover specific cases.

V.6.3 Variable Renamer

V.6.2.1 Functions and Interfaces Implemented

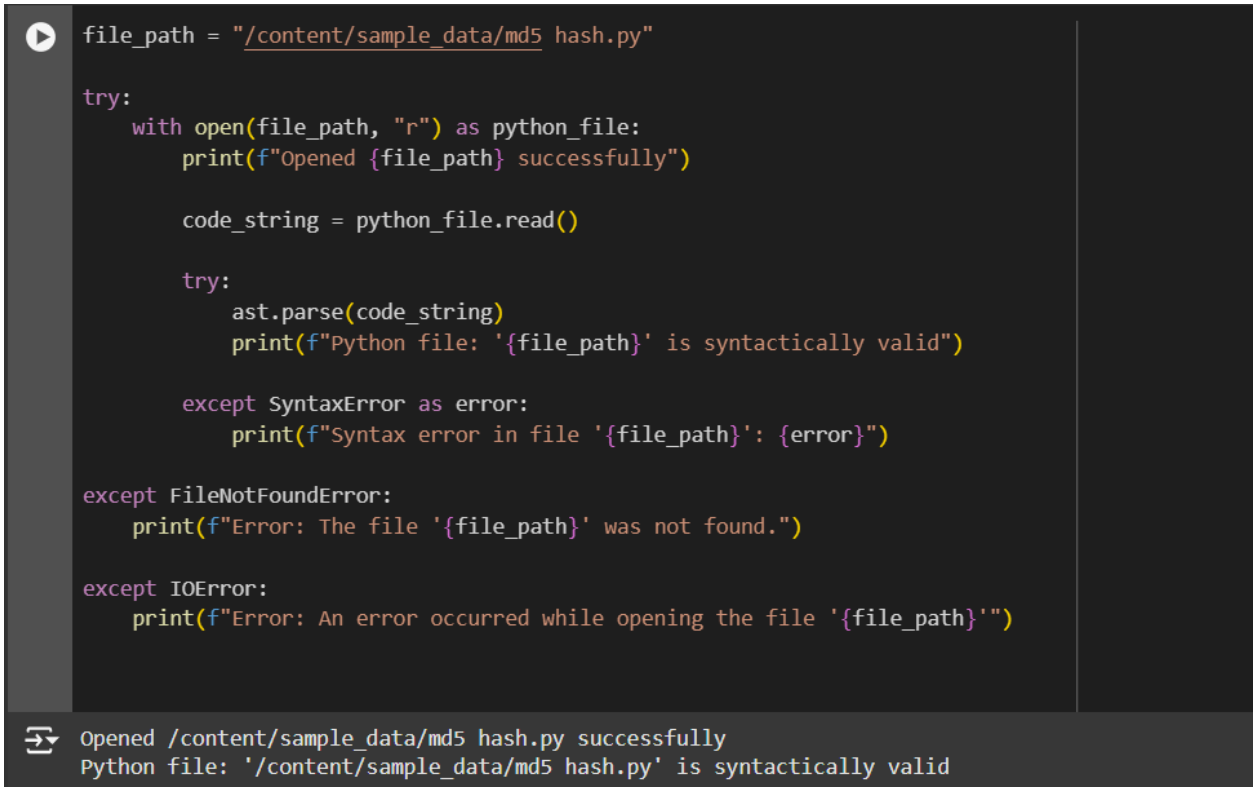
The Variable Renamer significance to the Cyber1- CodeVersion tool allows for Python file obfuscation effectiveness. For Python files to appear different at a glance, but functionally produce the same result, names of functions must be changed in order to reduce suspicion of similar code entities. The variable renamer works in conjunction with the function renamer as one process but must be run before the function renamer. The results of the process produces a distinct Python source file with changed variable names covering a variety of edge cases.

V.6.2.2 Preliminary Tests

To test the Variable renamer component, test files tailored towards variable heavy code were utilized and created by the team. More edge cases are yet to be discovered but a majority and base line of basic usage has been covered and is fully functional. If edge cases were discovered, enhanced methods were implemented within the subclass transformer to cover specific cases.

V.6.4 Python Syntax Analyzer

V.6.2.1 Functions and Interfaces Implemented



```
file_path = "/content/sample_data/md5 hash.py"

try:
    with open(file_path, "r") as python_file:
        print(f"Opened {file_path} successfully")

        code_string = python_file.read()

        try:
            ast.parse(code_string)
            print(f"Python file: '{file_path}' is syntactically valid")

        except SyntaxError as error:
            print(f"Syntax error in file '{file_path}': {error}")

except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")

except IOError:
    print(f"Error: An error occurred while opening the file '{file_path}'")
```

```
Opened /content/sample_data/md5 hash.py successfully
Python file: '/content/sample_data/md5 hash.py' is syntactically valid
```

Figure V.6.2.1: A screenshot of Google Colab, showing the Python file syntax parser code and the output for a syntactically valid Python input file.

Figure V.6.2.1 displays the Google Colab file written in C code which utilizes the “ast” library in Python. The Python Syntax Analyzers significance to the Cyber1- CodeVersion tool allows for preprocessing of files to conserve time. If Python files were to be analyzed with syntax issues, a number of semantic clones will be generated also with syntax issues. The AST libraries allows a Python source code file to be represented as a string, parsed into nodes, and represented as a module to test for correctness of code structure and logic.

V.6.2.2 Preliminary Tests

To test the UI & File Handler component, the team provided syntactically correct and incorrect file Python files on the local host machine to see if the specified files could be flagged for incorrect syntax. If the Python file was structurally correct according to the AST, it would be accepted. Otherwise, the user is prompted to enter another file that passes the requirements of a valid AST structure.

V.7 Alpha Prototype Demonstration

Our frequent meetings with our mentor throughout the semester has helped guide our development, with us showing the current prototype and progress each week. For our most recent meeting, we showed our mentor the current prototype and example outputs after using

the python command line version of the application. We showed how there were still some issues in overlap between variable and function renaming, as it is difficult to tell if an import alias is renaming a library, sub-function, or sub-variable. However, we expressed that the application works, with a robust file-search system that processes the file and outputs a clone for it. We also showed our Google Colab notebook file which broke down the individual functions mentioned in the Prototype Description above.

Our mentor expressed satisfaction with our prototype as well as our larger progress this semester. Specifically, he felt that an in-depth GUI was not necessary for the CST implementation so long as the python notebook file breaks down the steps and serves as documentation of what the source code is processed through. For the final deliverable, our mentor wants the python code to produce fully functional clones, even if they are not changed much, since syntax trees are difficult to fully manipulate. It also needs to be able to generate a specified amount of outputs, which can be done via randomization and potentially processing previous outputs again. The python notebook is expected to be well documented, so that he can see the individual functions and understand how they are manipulating the code.

For specific concerns addressed in the demo, our mentor explained that a specific style of more complex implementation for next semester is not set in stone and may be changed, which will be addressed further in the below Future Work section. With the randomized outputs, so that the outputs are extremely likely to be unique, he agreed with the potential method of cycling outputs in as new inputs to process to have more varied code. For questions on our end, we explained that our code largely functions already and that we are just struggling with the extreme amount of edge cases our code would have to cover, or at least not partially cover and break functionality. He also asked about how involved we expect the python file to be for the user, which we explained as being mainly hands-off once generation begins, as the user simply inputs the location of the source file and number of clones when prompted. During generation, we plan to save each output to the designated output folder as they are created so that the user may stop generation early without losing everything. To further mitigate loss and make analyzing clones easier, we are considering saving a code output after every function e.g. the function or variable renamer.

V.8 Future Work

Reflecting back to the first section of prototype development, every priority functional requirement has been implemented in terms of the obfuscation techniques for variable names and function names. If Python source code files were put side to side for comparison, they would appear to be unique, but functionally produce the same result. For the next portion of our tool development, our team will aim to create a more organized and visually appealing GUI, utilize genetic algorithms and assign probabilities for Python source code file changes, and enhance the structural changes of code logic.

The team’s plan to incorporate all of the new functional requirements is to first begin by implementing the probabilities and structural code changes. The emphasis of production of the Cyber1- CodeVersion tool is towards baseline functionality. The GUI components will be the last thing the team implements because it is lowly coupled and will not influence the effectiveness of the functional requirements. To prepare for these functional requirements, our team will continue to meet with our client to discuss the process of genetic algorithms, as well as non-deterministic algorithms that exist to help incorporate probability to our code.

VI. Glossary

Abstract Syntax Trees (AST)	A simple representation of a code structure, showing the flow of the program.
Common Vulnerabilities and Exposures	Publicly accessible database that displays common/known security vulnerabilities in various software, hardware, etc.
Non-Deterministic	The same input will not always have the same output.
Obfuscation	Making code difficult to parse for humans or computers.
Semantic Clone	Output of source code generated by a source code file that visually looks diverse, but behaviorally produces the same output.
Concrete Syntax Trees (CST)	Similar to an AST but provides more insight on program structure represented as nodes. Increased functionality for refactoring edges between nodes in the graph.
Graphical User Interface (GUI)	A clean and interactive window for end users to interact with software efficiently.

VII. References

[1] F. Erlacher and F. Dressler, 'On High-Speed Flow-Based Intrusion Detection Using Snort-Compatible Signatures', *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 495–506, 2022.

[2] S. More, M. Matthews, A. Joshi, and T. Finin, 'A Knowledge-Based Approach to Intrusion Detection Modeling', *2012 IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 75–81.

[3] V. Vinayakarao and R. Purandare, 'Structurally Heterogeneous Source Code Examples from Unstructured Knowledge Sources', *Indraprastha Institute of Information Technology, Delhi,* 2015, pp. 21-26.

[4] Instagram, "LibCST." github.com. https://github.com/Instagram/LibCST (Accessed Oct. 20, 2024)