CYBER1-CodeVersion

Requirements and Specifications

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 work 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 a malware unknown to the detection software and allowed 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 its 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 looks 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 original source code undergoes an algorithmic process which 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 which 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 are able to run nondeterministic algorithms which will manipulate the original source code structure from the graph, to obtain a new semantic clone graph. A reversed process all the way to converting 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 original 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 original source code. Iterative processes are performed to extract the “best fit” semantic clones.

While creating top tier Graphical User Interface isn’t at the top of the priority scale, 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 original 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 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 Bitdefnder and Norton provide security at 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 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 clearly written documentation that allows cyber experts to utilize the tool for studying how malware changes over time. The outlined preferences delivered from our client are that the use of Large Language Models should not be integrated into the tool to solve the problem. LLM's do not provide any insightful work towards solving the solution. In fact, the use of nondeterministic algorithms or genetic algorithms were 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

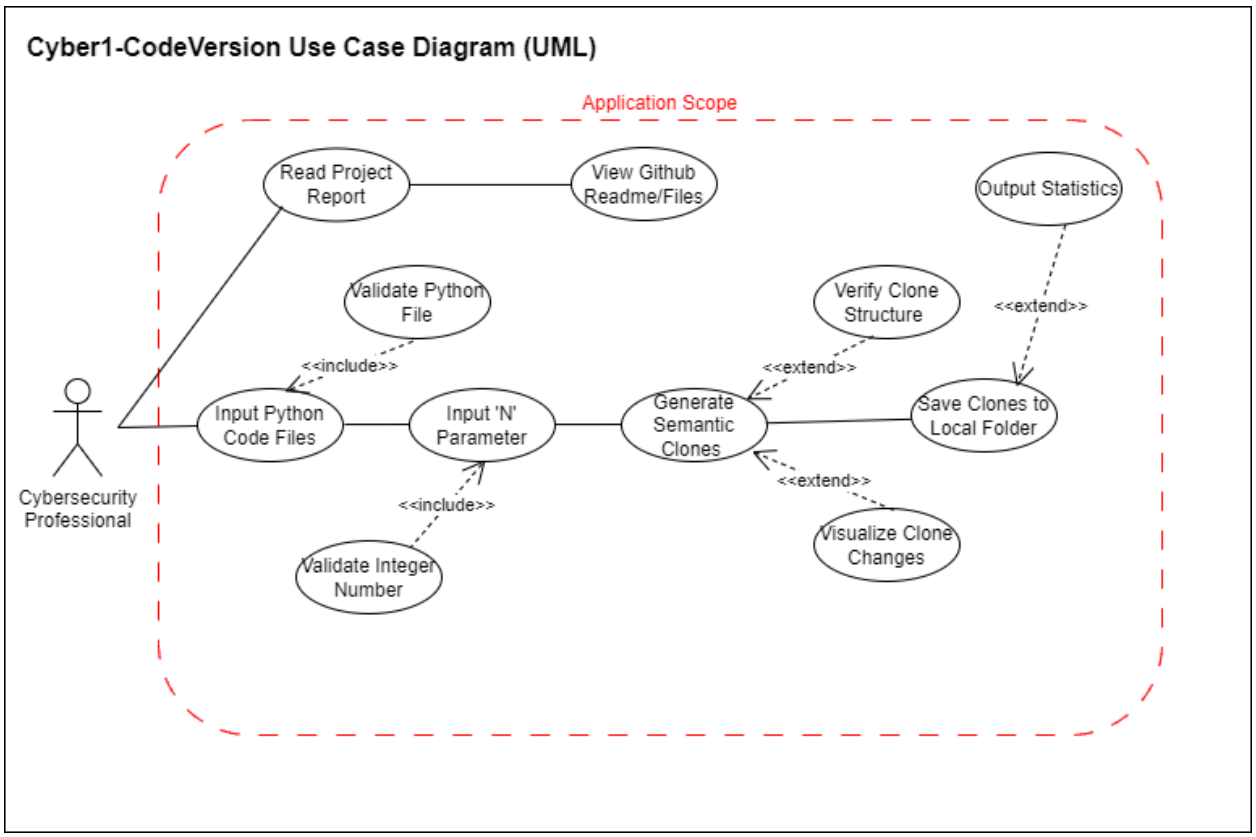
As a cybersecurity expert, designing malware detection software is a crucial responsibility in ensuring the integrity of user hardware, software, and other high value systems. While complex, these malware analysis software applications use a variety of methods to identify malware on user systems such 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 around the field of cybersecurity relates to the fact that malware and technology is 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 is stopping them from changing the original source code to produce a different digital signature.

The principle behind digital signatures, while taking the form of a bit-length string, is that a small change to a file can generate an entirely unique digital signature. Our designed tool aims to solve this issue by allowing cybersecurity experts to visually analyze how source code changes over time and develop countermeasures to detect these changes.

The objectives we aim to accomplish by designing the Cyber1-CodeVersion tool is 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 that 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



II.2.1 Use Cases

Read Project Report	
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	
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	
Pre-Condition	User has 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 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 correctly formatted Python file and is accepted into the file drop box.
Related Requirements	Source Code File Box

Input ‘N’ Parameter	
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	
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	
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

II.2.2 Functional Requirements

1. User Input and File Handling

Source Code File Box	
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 towards this requirement. As a cybersecurity professional, learning and analyzing the source code files is a necessary requirement to understanding 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	
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 towards this requirement. As a cybersecurity professional, the more branches we produce of an original source code file, allows for 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	
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 towards this requirement. As a cybersecurity professional, if the user wants to do 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	
Description	The application needs to contain a “Generate Clones” button that effectively creates ‘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 towards 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

Description	The application needs to contain an additional file box that is populated with semantic clone files once they are done generating from the original 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 towards 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	
Description	The application needs to contain a button which 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 behavior of how source code changes.
Source	Bryan Frederickson and Samantha Brewer originated and discussed methods towards 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.2.3 Non-Functional Requirements

Python Exclusivity:

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:

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

English-Only Support:

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:

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:

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

Multi-File Support:

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.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 to change and grow 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 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. 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 an original source code file that visually looks diverse, but behaviorally produces the same output.

IV. 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.