Functional Specification

Maldive
Malware Analysis Tool

Shane Doherty C00249279

Supervisor: Joseph Kehoe

South East Technological University

1 Table of Contents

Contents

2 Application Description

This application will be a tool that can be used to dynamically analyse malware. The program will provide utilities that allow for a thorough investigation of how the program operates, and what information is being accessed by it.

3 Application Functionality

3.1 Core Features

3.1.1 Process hooking

The malware analysis tool must be able to be attached to a process being run on the operating system, then collect information as it executes. This is required to dynamically analyse the operation of the malware.

Once the malware analysis tool has been launched, it needs to be attached to the process that is suspected of being infected with malware. This can be done in a number of ways, such as through a hooking mechanism that allows the tool to intercept and monitor actions made by the process.

Once the tool has been successfully attached to the process, it can begin to collect information about its behaviour as it executes. This information might include system calls made by the process, files opened or modified, network connections established, and other events that can provide insight into the behaviour of the malware. This type of information is typically logged in real-time, allowing analysts to monitor the behaviour of the malware as it unfolds. This information can then be analysed and used to identify patterns of behaviour that are indicative of malware activity, such as attempts to connect to an external server, modification of critical system files, or attempts to spread to other machines on the network.

3.1.2 Process Description

When the user selects a process to analyse, the malware analysis tool should display detailed information about the process and its running state. This information should be unique and useful, providing insights into the behaviour of the process and how the operating system views it.

One key piece of information that should be displayed is the Process ID (PID) of the selected process. This is a unique identifier assigned by the operating system to each process and can be used to track the process as it executes.

The tool should also display information about the process's memory usage, including how much memory it is currently using, how much memory it has allocated, and how much memory it has reserved. This can provide valuable insights into how the process is using system resources and whether it may be engaging in malicious behaviour, such as attempting to perform buffer overflows or other memory-based attacks.

3.1.3 Code extraction

Once a process has been hooked into, the code of the targeted program should be able to be extracted. This is typically done by using a disassembler, such as the Capstone disassembly tool, to translate the machine code of the program into a human-readable format that can be analysed.

The Capstone disassembly tool is a widely-used tool in malware analysis that supports a variety of architectures and instruction sets, most notably x86 instructions for use in this project. It can be used to disassemble both binary files and live processes, and it provides a rich API that can be used to programmatically analyse and modify the disassembled code.

Once the code has been extracted and disassembled, the user should be able to investigate and modify it to determine if there are malicious instructions being used. This might involve looking for specific patterns of behaviour, such as attempts to steal sensitive data or communicate with remote servers, or it might involve analysing the structure and flow of the code to identify potential vulnerabilities or exploits.

In some cases, the user may also need to modify or replace parts of the code in order to test the behaviour of the malware or to develop countermeasures. This might involve patching vulnerabilities or adding instrumentation to track the behaviour of the malware in real-time. Therefore, having functionality that allows for this should be supplied/

3.1.4 Virtual Memory scanning

A virtual memory scanner is a tool used in malware analysis to examine the memory space of a program while it is running. The tool operates by accessing the virtual memory of the targeted program and extracting useful data that the program is reading. This information can help identify malware that is attempting to evade detection or hide its presence by residing only in memory.

As the scanner identifies potential malware, it extracts relevant data from the memory space and displays it in an easy-to-read format. This might include information such as the addresses of memory locations being accessed by the program, the contents of registers or other processor state, or the data contained in specific data structures or objects.

The output of the scanner must be carefully organised and presented to the user in a way that does not confuse or overwhelm them. This might involve grouping related data together, using colour-coding or other visual cues to highlight important information.

3.2 Non-Core Features

Process View A list of currently running processes should be displayed to the user, allowing them to select it to be analysed by the tool. This should include a form of

identification that can effectively outline which process they are selecting, along with additional information that is specific to the process.

4 Users

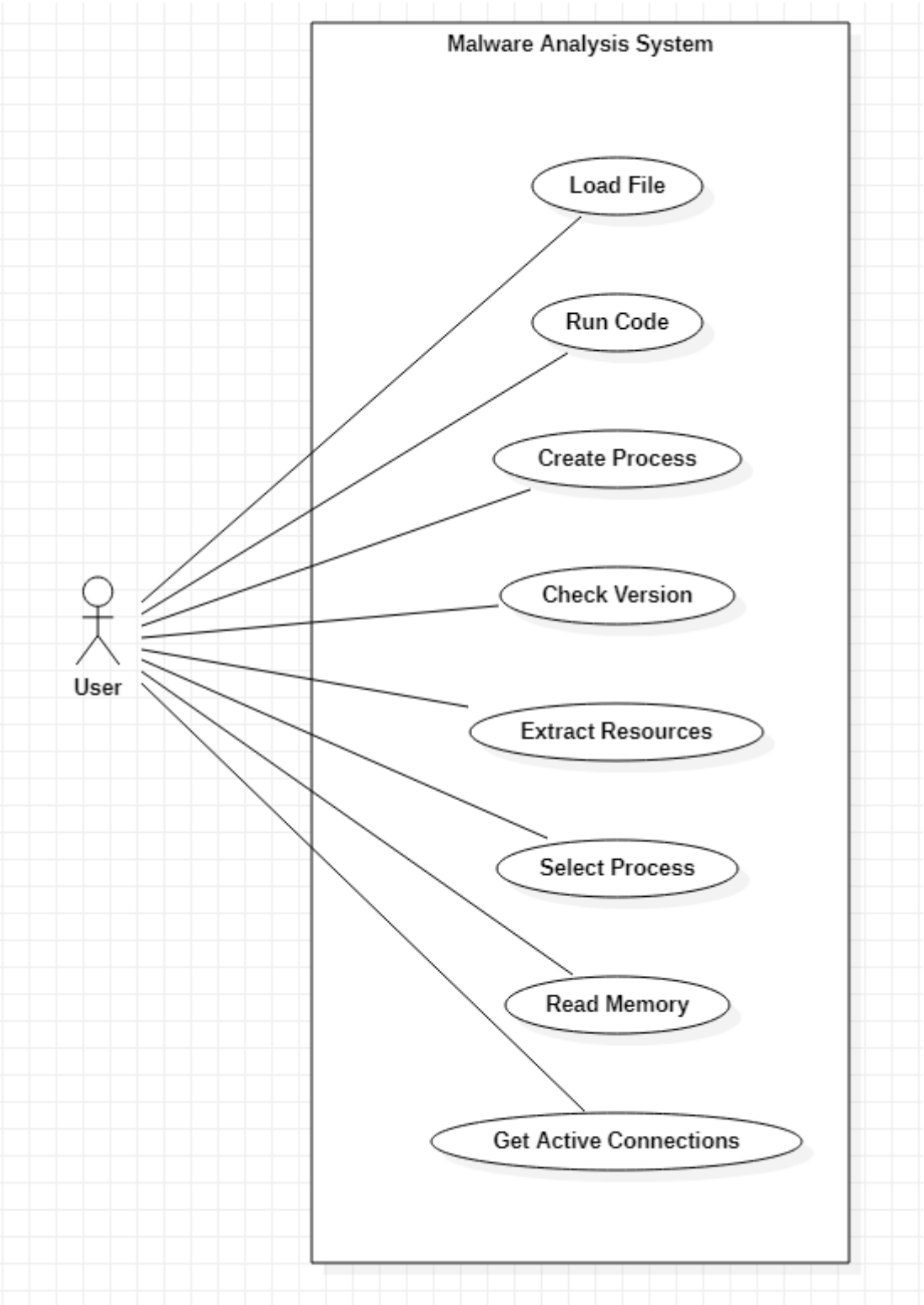
The users of this analysis tool can be broadly categorised into two groups: hobbyists and general users. The tool aims to serve both as an educational tool and as an analysis tool that can help users investigate if a program is malicious.

The first group of users includes hobbyists who are interested in understanding how malware operates. These users may be students, researchers, or enthusiasts who are interested in learning about the technical aspects of malware and how it can be analysed. The tool can help them explore and understand the inner workings of malware, as well as identify potential malware threats in their own systems.

The second group of users includes general users who may not have a background in coding or malware analysis, but still want to investigate if a program is malicious. These users may be concerned about the security of their systems or suspicious of a particular program, and want to conduct a basic analysis to determine if it poses a threat. The tool can provide them with a simple and accessible way to examine a program's memory space and identify any suspicious activity.

While the tool is designed as a technical tool, it does not require a professional-grade understanding of coding to be used effectively. Users with a basic knowledge of coding concepts and experience with command-line tools should be able to utilise the program to conduct basic analysis. The tool also includes documentation and educational resources to help users understand how it works and how to interpret its output.

5 Use Case


Malware Analysis System
Load File
Run Code
Create Process
Check Version
Extract Resources
Select Process
Read Memory
Get Active Connections
User

5.0.1 Load File Use Case

Name	Load File
Description	This use case begins when a user wishes to load a file containing executable code. The user is shown a file explorer window, where they choose a file to be loaded into the program.
Preconditions	The user wants to load a new file into the program
Primary Path	1. The user selects an option to load a new file 2. A file explorer window is shown 3. The user chooses which file they wish to enter 4. The file is loaded into the application, replacing the previous file if it exists
Postconditions	The file is shown in the application
Alternate Paths	4a. The file is invalid or does not contain executable code 1. An error message is shown, detailing that it is an invalid file 2. The file explorer window is shown again 3. The user selects a valid file from the explorer, and it is loaded into the application

5.0.2 Run Code Use Case

Name	Run Code
Description	This use case begins when a user wishes to run the code within a file. A debugger window is opened, and the user is given control over the file's executable code.
Preconditions	A file has been chosen and the user wishes to run the code it contains
Primary Path	1. The user chooses an option to run the code 2. The relevant fields are populated with information from the file 3. The user steps through the code, and the fields update accordingly 4. The user stops the execution when they are finished
Postconditions	The file has completed execution
Alternate Paths	N/A

5.0.3 Generate Report Use Case

Name	Generate Report
Description	This use case begins when a chosen file is executed by the program, and the program analyses its actions. A report is generated, detailing the detected attack methods it utilised.
Preconditions	A file has been chosen and the user wishes to generate a report
Primary Path	1. The user chooses an option to generate a report 2. The application runs through the executable code of the program 3. The application documents any malicious actions of the program 4. The report is shown to the user
Postconditions	A report has been generated
Alternate Paths	N/A

5.0.4 Create Process Use Case

Name	Create Process
Description	This use case begins when the user selects an option to run the program, creating a process that can be tracked by the analysing tool.
Preconditions	A file has been selected and the user selects the run option
Primary Path	1. The user chooses an option to run the file 2. The application launches the program, creating a process which identifies it 3. Identifying values are shown to the user
Postconditions	A process has been launched and values are updated
Alternate Paths	N/A

5.0.5 Check Version Use Case

Name	Check Version
Description	This use case begins when the program determines the version of the file to assist with reading sections of the file data.
Preconditions	A file has been selected and the user selects the run option
Primary Path	1. The PE file is loaded with its contents ready to be scanned 2. The byte value representing the version is read by the program 3. The version is stored for later use and displayed to the user
Postconditions	The version is displayed to the user
Alternate Paths	N/A

5.0.6 Extract Resources Use Case

Name	Extract Resources
Description	This use case begins when the file is read by the program, extracting the x86 instructions from the binary data
Preconditions	A file has been selected by the user
Primary Path	1. The file is loaded from the directory chosen by the user 2. The binary instructions are loaded into the Capstone disassembler 3. Capstone translates the instructions into readable x86 instructions 4. The instructions are loaded into a data structure
Postconditions	The instructions are displayed to the user
Alternate Paths	N/A

Name	Select Process
Description	This use case begins when the user chooses to enter a process that is currently running on the system.
Preconditions	The user wants to select a process to be loaded into the program
Primary Path	1. The user chooses to open a program 2. The user switches to “Process ID Mode" 3. The user enters a process ID, or chooses a process from a list 4. The program loads the process from the process ID.
Postconditions	The program displays information about the chosen process on the screen to the user
Alternate Paths	4a. The process ID is invalid or the process has been closed 1. An error message is shown, detailing that it is an invalid process ID 2. The select process screen is shown again, allowing the user to choose again 3. The user selects a valid process ID, and it is loaded into the application

Name	Read Memory
Description	This use case begins when the user chooses to read the virtual memory of a program.
Preconditions	A process is loaded in the program and is ready to be read from
Primary Path	1. The user chooses to display the virtual memory of the loaded program 2. The process ID is used to scan through the process' virtual memory 3. Details about the virtual memory are shown to the user
Postconditions	The program displays a screen containing the virtual memory of the chosen process
Alternate Paths	N/A

Name	Get Active Connections
Description	This use case begins when the user chooses to get the currently running network devices from the system, to then display the packet information.
Preconditions	A process is loaded in the program and is ready to be read from
Primary Path	1. The user selects the network viewing window 2. The window is displayed to the user, and updates with current network information 3. The user selects a network device that is in use 4. The addresses of the network device is shown to the user 5. The user chooses a packet from the chosen address 6. The packet information is displayed to the user
Postconditions	The program displays the network devices, addresses, and packets that have been sent
Alternate Paths	N/A

6 FURPS

FURPS is an industry standard used to describe the functional and non-functional requirements of an application. This gives the expectations that the program should be able to adhere to when it is fully developed. The acronym FURPS gives a detailed layout of these requirements under the following headings.

- Functionality
- Usability
- Reliability
- Performance
- Supportability

6.1 Functionality

This section refers to the core operation of the application, with the main functions that are required to analyse malware This is an important aspect to a technical program such as the dynamic malware analysis tool, as the functionality is th. Having these functions are essential to having a working program. For this project, the main functions are as follows:

- Ability to read and load a file.
- Disassemble the binary code of the file, producing code that can be executed.
- Ability to step through the code and execute accordingly.

6.2 Usability

This section refers to how accessible and streamlined the application is. Using conventional design patterns that the user can quickly understand has a major impact with how usable a program is. Utilising this, the program should be able to deliver on the following:

- A new user should be able to install the application within 2 minutes after downloading it.
- A new user should be able to load and begin stepping through a program within a minute after installing the application.

6.3 Reliability

This section refers to the stability of the program, and its ability to respond to any action that the user inputs. Any unintended action should fail gracefully, giving the user as much information as possible on what went wrong, and run through an alternative path if possible. The following points demonstrate the expected reliability of the program.

- The application should be able to run 95% of files as expected.
- Malware should be correctly and completely identified 90% of the time.
- The application should not crash 99% of uses.

6.4 Performance

This section refers to the speed at which each of the functions are executed by the application, including any delays caused by prolonged processing of actions. To be deemed acceptable, the program should adhere to the following points:

- The program must not take longer than one second between stepping through lines of code.
- It should not take longer than 2 seconds to load a file.

6.5 Supportability

This section refers to the extent at which the program can be modified, expanded upon, and maintained. Conventional programming standards, well documented code, and clearly defined comments and variables are essential for having a long-lasting program that can be developed well into the future.

6.6 Metrics

This section describes the outline of the features and behaviour of the program that are required to be deemed as a successful project. The following points describe the expected outline of the fully-developed application:

- The user should be able to load and run the code of files within the application
- The application should be able, to a reliable degree, declare if a program is malicious.
- The application should accurately trace through the lines of execution, and provide a detailed and accurate report at the end.
- The application process should be as user-friendly as possible by showing the information in an easy to follow manner.