****

**Software Requirements**

**Specification**

**for**

**SEng Malware (SEM)**

**Version 0.1**

**Prepared by:**

1. John Dornheim
2. Chris Durand
3. John Schlatter
4. Shelby Mitchell
5. Alex Lopez
6. Jon Harrity

**CSC 190 Term Project**

**Advisor:** Dr. Scott Jeffreys

Spring 2019

*Made with love in Axinn 806*

**Table of Contents**

1. Introduction

1.1 Preface

1.2 Overview

1.3 Glossary

2. User Requirements

3. System Architecture

3.1 User Interface & Program Entry

3.2 Virus Database

3.3 File Scanner

3.4 Threat Elimination

4. System Requirements

5. System Models

6. System Evolution

7. Appendix

1. **Introduction**

**1.1 Preface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reasons For Changes** | **Version** |
| C. Durand  J. Dornheim  J. Schlatter  S. Mitchell  A. Lopez J. Harrity | 2/10/19 | Began initial planning + designing of SEng Malware | 0.1 |
| C. Durand  J. Dornheim  J. Schlatter  S. Mitchell  A. Lopez J. Harrity | 2/10/19 -- 4/20/19 | Creation and implementation of SEng Malware | 0.2 |
| C. Durand  J. Dornheim  J. Schlatter  S. Mitchell  A. Lopez J. Harrity | 4/20/19  \*subject to change\* | \*subject to change\* | TBD |

**1.2 Overview**

In an increasingly connected world, security has become a necessary tool for every-day end-users. According to Norton Security, the U.S. will account for half of all breached data by 2023. In addition, total malware infections have been increasing over the last 10 years. In an effort to combat the globally growing malware problem, scanning for viruses will help mitigate this risk. SEng Malware is a malware scanner that will help the clients of Enterprise Software protect their devices from malicious files. With its constantly-updated malware database, SEng Malware allows users to scan parts or all of a file system for malware, and keeps records of any malicious files it finds. Users can schedule monthly, weekly, or daily scans of their files for peace of mind, and have the option to quarantine malware that has been found. Our primary objective is to allow end-users to effectively quarantine and delete files on their Windows devices that pose threats with an easy-to-use program that even the least savvy of computer users can utilize to its full potential.

**1.3 Glossary**

Throughout this document, many technical terms will be used. Their working definitions as applicable in this document are defined below.

**Back-end**: Or data access layer, is the part of the computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.

**Front-end:** Or presentation layer, is the part of the computer system or application in which the user interacts directly.

**Functional Requirement:** A statement of some function of feature that should be implemented in a system.

**Git:** A distributed version management and system building tool where developers take complete copies of the project repository to allow concurrent working.

**Github:** A server that maintains a large number of Git repositories. Repositories may be private or public. The repositories for many open-source projects are maintained on GitHub.

**GUI**: Graphical user interface. A GUI is the interface through which user interact with a program. Specifically, the GUI will be used here to communicate information to the end-user, and accept user input.

**Interface:** A specification of the attributions and operations associated with a software component. The interface is used as the means of accessing the component’s functionality.

**Malware**: Software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system.

**Nonfunctional Requirement:** A statement of a constraint or expected behavior that applies to a system. This constraint may refer to the emergent properties of the software that is being developed or to the development process.

**Open Source:** An approach to software development where the source code for a system is made public and external users are encouraged to participate in the development of the system.

**Python:** A programming language with dynamic types, which is particularly well-suited to the development of web-based systems.

**Risk:** An undesirable outcome that poses a threat to the achievement of some objective. A safety risk is a measure of the probability that a hazard will lead to an accident.

**Risk Management:** The process of identifying risks, assessing their severity, planning measures to put in place if the risks arise and monitoring the software and the software process for risks.

**Server**: A computer program which manages access to a centralized resource or service in a network.

**Software Architecture:** A model of the fundamental structure and organization of a software system.

**User Interface Design:** The process of designing the way in which system users can access the system functionality, and the way that information produced by the system is displayed.

**Version Control (VC) Systems:** Software tools that have been developed to support the process of version control. These may be based on either centralized or distributed repositories.

1. **User Requirements**

**2.1** The user will be able to scan files on their system for malicious programs.

**2.2** The system updates its virus definitions database with the most recent ones available.

**2.3** The system keeps a log of malicious files quarantined and deleted.

**2.4** The user can schedule scans by date and time.

1. **System Architecture**

SEng Malware will have four system modules. There will be a user interface and program entry module, a virus definition module, a file scanner module, and a threat elimination module.

**3.1** User Interface and Program Entry

**3.1.1** Display scanning results: Users will be shown the results of previous scans. For

each previous scan, the directories scanned will be listed. Any malware detected will

be listed, including the path to the program, the matching virus definition, and the

action taken to handle the issue.

**3.1.2** User Configurations

1. Scheduling scans by date and time of day.
2. History of files quarantined and deleted via logs.
3. File exceptions for false positives.

**3.1.3** Settings

**3.1.4** Main

**3.2** Virus Database

**3.2.1** Provide button to update antivirus definitions.

**3.2.2** Accesses “signature byte code and patterns” from database.

**3.3** File Scanner

**3.3.1** Scans specified directories, disks, or individual files.

**3.3.2** Flags files that are a “risk”.

**3.4** Threat Elimination

**3.4.1** Logs quarantine/deletion history.

**3.4.2** Removal/Quarantine of malicious files.

1. **System Requirements**

**4.1** Users can schedule scans for specific dates and times.

**4.2** Users will be able to specify which directories and folders gets scanned.

**4.3** Users have the ability to perform a “Surface Scan” which will scan the device’s most popular malware hiding spots.

**4.4** The scanner utilizes signature byte code detection and pattern matching.

**4.5** Upon detection of new malware, the new virus definition(s) will be saved upon deletion of the file(s).

**4.6** The system will update the virus definitions on a regular basis via the server database.

**4.7** Malware definitions may be derived from Avast, or another openly available antivirus program database. (Avast, Comodo, our own...??)

**4.8** Upon completion of a scan, the software application will quarantine the malicious file(s) and give the user the option of deletion or recovery of the file(s) if it finds any.

**4.9** The system shall be developed to run on Windows.

**4.10** The system will be written in Python.

**4.11** Electron may be used to develop the GUI.

**4.12** The software will have the option to run at startup.

1. **System Models**

**5.1 \*\*Diagram Showing the relationships between the system components and its environment\*\***

****

1. **System Evolution**

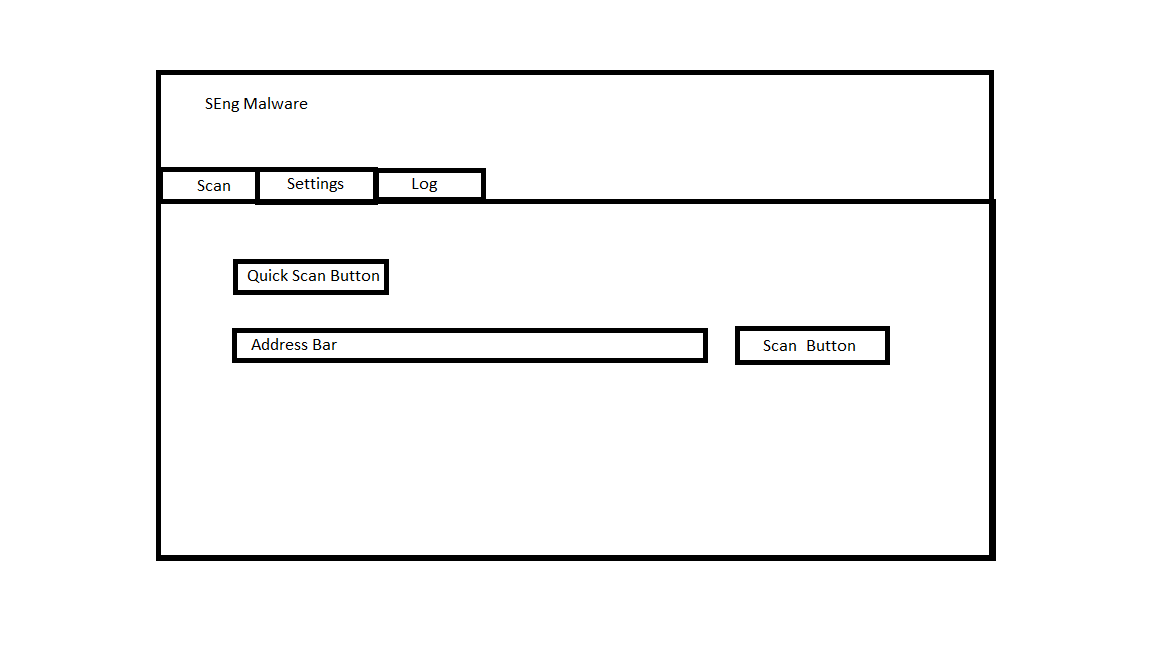
**6.1** Heuristic Detection Algorithm Integration - The system will analyze previously deleted malware’s bytecode in order to predict new bytecodes that future malware may use.

1. **Appendix**

**7.1 Functional Requirements**

**7.1.1** Graphical User Interface (GUI)

**7.1.2** Variable Scanner

1. Users will be able to choose whether they would like to conduct a Deep scan or an Surface scan to check their computer for malware. The Surface scan will search for the most popular directories where malware is commonly found, whereas the Deep scan will search the entire specified drive.

**Example of homescreen (above)**

**7.1.3** Exclusions

1. List of files and their paths to be excluded from deletion or quarantine in case of a false positive.

**7.1.4** Auto Run

1. When the computer boots successfully, the program can either startup alongside the desktop or startup manually by the user via the executable.

**7.1.5** Scheduled Scans

1. Users may schedule how many days and what time of the day they would like to scan their computer for malware.

**7.1.6** Quarantine/Deletion

1. Malware detected on a user’s computer will be quarantined, examined, and deleted as the user sees fit.

**7.1.6** Update Definitions of Viruses

1. Virus definitions taken from an open source database (exact database TBD) will be routinely updated so as to always protect the user.

**7.1.7** Record Quarantine/Deletion History

1. The names and information surrounding a deleted strain of malware will be recorded in a “log” tab. The user may always look back on their log to see the history of malware.

**7.2 Non-Functional Requirements**

**7.2.1** Version Control

1. Version control handled by Github, https://github.com/jonharrity/Seng-Malware

**7.2.2** Operational

1. SEng Malware aims to be up and running 99% of the time to make sure that users always feel secure.

**7.2.3** Technology Choice

1. SEM focuses on a Windows system because the majority of laptop and desktop users run on a Windows OS.

|  |  |
| --- | --- |
| **Functional** | **Nonfunctional** |
| GUI (electron) | VC (Git Hub) Version Control |
| Variable Scanner (Surface/Deep)(Single Directories) | Operational (Uptime for Server)(9% uptime) |
| Exclusions (of files) | Technology Choice (Windows) (Python Backend, Flask Backend) |
| Auto Run |  |
| Timer (Scheduled Scans) |  |
| Quarantine/Deletion |  |
| Update definitions of viruses |  |
| Record Quarantine/Deletion History |  |

**(Contributions on final page)**

**Contributions**

John Dornheim - Functional/Non-functional Requirement, Formatting, Editing

Chris Durand - User Requirements/System Requirements, Overview

John Schlatter - Overview, Logos, Formatting

Shelby Mitchell - Glossary, Scheduling, Editing, Appendix

Alex Lopez - Systems Models, Functional/Non-functional Requirement

Jon Harrity - Systems Architecture, User Requirements/System Requirements