Bilkent University
Department of Computer Engineering
Senior Design Project
T2322
CleaverWall

Analysis and Requirement Report

21901358, Ali Emre Aydoğmuş,
emre.aydogmus@ug.bilkent.edu.tr
21903472, Arda Barış Örtlek,
baris.ortlek@ug.bilkent.edu.tr
21802925, Onur Korkmaz,
onur.korkmaz@ug.bilkent.edu.tr
21802856, Selahattin Cem Öztürk,
selahattin.ozturk@ug.bilkent.edu.tr
21903227, Yekta Seçkin Satır,
seckin.satir@ug.bilkent.edu.tr

Özcan Öztürk
Erhan Dolak
Tağmaç Topal

13.11.2022
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

Analysis and Requirement Report

CleaverWall

1 Introduction

Malware detection is a trending topic since the 1980s and it is becoming more significant due to the immense increase in malware programs. Even though there is use of different approaches, most of the open-source anti-malware programs rely upon signature based methods. CleaverWall optimizes this method commonly used in the industry by developing an application that totally depends on machine learning to eliminate the disadvantages of signature based detection methods. The application has three stages, the extraction of features to put into the model, the classification using a machine learning model, and the operations after the detection of malware. We are planning to produce two versions of the products, the first will be a web service and the second will be a Windows application.

The details of CleaverWall, its functional and non-functional requirements, system models and other analysis elements will be discussed in the rest of this report.

2 Proposed System

2.1 Overview

CleaverWall is an anti-malware mechanism to detect whether a portable executable is malicious, if so, to classify the malware type. For the classification process, a set of malware classifiers trained with various machine learning and deep learning techniques will be used. Static and dynamic analysis will be conducted to create different feature vectors for the models. Those different classifiers will work collaboratively to decrease the false positive occurrences.

In static analysis, we have different approaches. The first approach consists of disassembling a portable executable and collecting information from that .asm file. Operation codes, registers, symbols, sections, miscellaneous and Windows API calls will be analyzed to create features [1], [2]. Capstone [3] and pefile [4] modules will be utilized to form disassemble scripts. Moreover, portable executable headers include valuable information such as SizeOfCode and AddressOfEntryPoint [5]. We aim to extend our feature vector by adding new features from PE headers. Feed Forward Neural Network structure will be used for this model.

The second approach is to represent an executable file as a grayscale image by interpreting every byte as one pixel in an image, ranging from 0 to 255. This approach reduces malware classification problem to image classification problem on which Convolutional Neural Networks are very efficient. Studies show that images of the same malware family are similar to each other while they are distinct from images of other families [6], [7], [8].

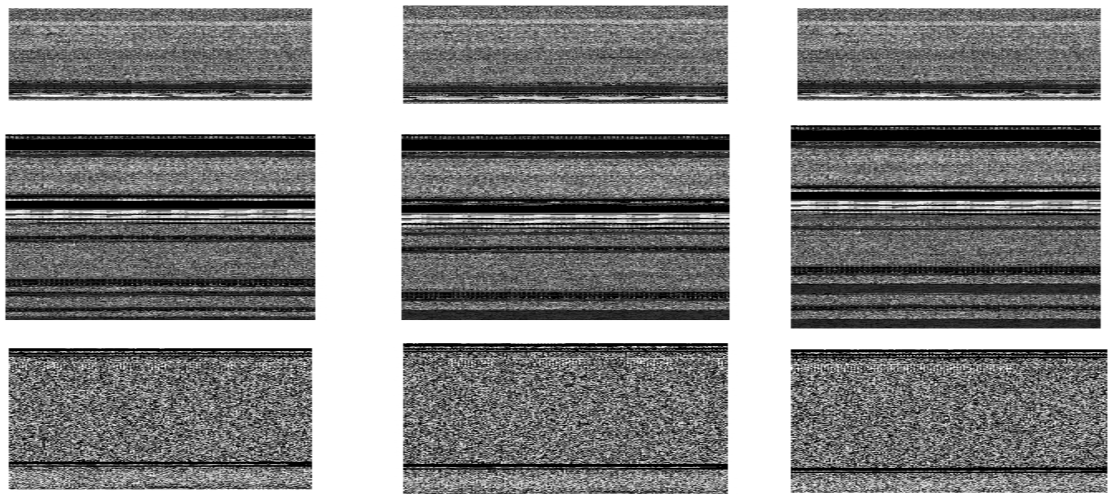


Figure 1: The first row represents Yuner.A samples, the second row represents VB.AT samples and the third row represents Skintrim.N samples [9].

The third approach is to combine models discussed in the first and the second approach and form a multimodal malware classifier. Output of 2 different layers fed with different inputs will be concatenated into a single layer. We observed similar approaches [10], [11] and decided to implement this specific approach to increase the evaluation score of the static classification.

In dynamic analysis, a portable executable's Windows API call sequence will be analyzed to create the feature vector [12]. For this purpose, an executable will be run on a sandbox for a small amount of time. After that, the sandbox will return information regarding the API call sequence. Cuckoo Sandbox [13] and a server where the sandbox can operate will be utilized. For the model, Feed Forward Neural Network or XGBoots will be selected depending on their performance.

Dataset for training the models will be taken from VirusShare [14]. Academic API of VirusTotal [15] will be utilized to label the dataset. Moreover, we aim to enrich our dataset from the executable files that users upload to the server when using Web Client.

Since CleaverWall is an open source project, we initially do not aim to compete with premium services. However, open source anti-malwares such as ClamAV and other applications using only signatures based detection methods perform poorly. Therefore, our goal is to surpass them on both evaluation and performance metrics due to our machine learning approach. The innovation type that we want to implement is product performance and it is incremental. As we improve our application, we want to change the case that reliability is expensive.

2.2 Functional Requirements

2.2.1 User Functionalities

Users can:

- Scan a portable executable file through the desktop or the web app,
- Scan a windows directory,
- Demand further analysis with other heavier machine learning models,
- View the detailed log of scan results,
- Schedule an auto-scan process for a specific file path,
- Determine options such as quarantine or deletion for a scanned portable executable file which is detected as malicious.
- View previous scan results.

2.2.2 System Functionalities

The server can:

- Accept portable executable files,
- Apply static and dynamic analysis on them,
- Return the results,
- Save the obtained data to a database.

Desktop application can,

- Use models of different weights to scan a portable executable file statically,
- Scan directories,
- When needed, send files to server for further analysis,
- Display the results,
- Quarantine detected malwares & delete them on demand,
- Present a UI.

Web client can:

- Display a UI that accepts portable executable files,
- Display the results,
- Display information about the system.

2.3 Non-functional Requirements

2.2.1 Usability

- Our graphical user interface will be clear and intuitive to increase the ease of usage.
- We will ensure that the graphical user interface styles of the desktop application and the web server are similar to each other so that our customers will not have a hard time when they switch the service that they are using.

2.2.2 Security and Privacy

- We will ensure that the uploaded files will be safe against cyber attacks.
- We will ensure that our classification model can not be disturbed by outside forces.
- Newly obtained and saved data to improve the machine learning model should not be disturbed by outside forces.

2.2.3 Reliability

- We will ensure that our classification model can classify mainstream malware families.
- By using the new saved data, our classification model should be able to increase its accuracy.

2.2.4 Scalability

- The server should be able to analyze multiple files that are uploaded by different users concurrently.

2.2.5 Maintainability

- The mean time to restore the system following a system failure on the server side must not be greater than 10 minutes. Mean time to restore the system includes all corrective maintenance time and delay time.

2.4 Pseudo Requirements

- Source code will be controlled through git, on a Github repository.
- The desktop and server-side applications will be written in Python. For the desktop application, C scripts may be written as needed.
- Cuckoo Sandbox will be utilized.
- Capstone will be used as the disassembler tool.
- Tensorflow [16] framework will be used for machine learning operations.
- The simple web client will be written in Javascript.
- Django framework will be utilized for the backend application.

2.5 System Models

2.5.1 Scenarios

2.5.1.1 Scenarios for Desktop

Scenario 1: Scan an executable

Actors: Desktop User

Entry Conditions: User should be in the scanning section.

Exit Conditions: User does not demand further analysis.

Flow of events:

1. User provides the path of the executable file.
2. User selects the type of analysis and one of the related classifiers.
3. User clicks the scan button.
4. After scanning operations are completed, user sees the result screen.
5. User optionally deletes or quarantines the executable file.
6. User optionally demands further analysis with different classifiers.

Scenario 2: Scan a directory

Actors: Desktop User

Entry Conditions: User should be in the scanning section.

Exit Conditions: User does not demand further analysis.

Flow of events:

1. User provides the path of the directory.
2. User selects the type of analysis and one of the related classifiers.
3. User clicks the scan button.
4. After scanning operations are completed, user sees the result screen.
5. User optionally deletes or quarantines executable files that are predicted as malicious.
6. User optionally demands further analysis with different classifiers.

Scenario 3: Schedule an auto-scan process

Actors: Desktop User

Entry Conditions: User should be in the scheduling section.

Exit Conditions: User navigates to a different application section.

Flow of events:

1. User provides the path of the directory or executable.
2. User selects the type of analysis and one of the related classifiers.
3. User determines the date of the scan operation.
4. User clicks the schedule button.

Scenario 4: Sign Up

Actors: Desktop User

Entry Conditions: User clicks Sign Up button.

Exit Conditions:

- User closes the app.

- Credentials are authorized.

Flow of events:

1. User provides an email address and password.
2. User gets an authentication email.
3. User verifies the account.

Scenario 5: Login

Actors: Desktop User

Entry Conditions: User clicks Login button.

Exit Conditions:

- User closes the app.
- Credentials are verified.

Flow of events:

1. User enters his/her email address and password.
2. Credentials are verified.

Scenario 6: View previous scan results

Actors: Desktop User

Entry Conditions: User is logged in and they click the Previous Scans button.

Exit Conditions:

- User closes the app.
- User navigates to a different application section.

Flow of events:

1. User waits for the server request to fetch previous scans.
2. User observes the previous scan results.
3. User may save the list of previous scans.

2.5.5.2 Scenarios for Web

Scenario 1: Scan an executable

Actors: Web User

Entry Conditions: User should be in the home page.

Exit Conditions: User does not demand further analysis.

Flow of events:

1. User uploads the executable file.
2. User selects the type of analysis and one of the related classifiers.
3. User clicks the scan button.
4. After scanning operations are completed, user sees the result screen.
5. User optionally demands further analysis with different classifiers.

Scenario 2: Sign Up

Actors: Web User

Entry Conditions: User clicks Sign Up button.

Exit Conditions:

- User closes the app.
- Credentials are authorized.

Flow of events:

1. User provides an email address and password.
2. User gets an authentication email.
3. User verifies the account.

Scenario 3: Login

Actors: Web User

Entry Conditions: User clicks Login button.

Exit Conditions:

- User closes the app.
- Credentials are verified.

Flow of events:

1. User enters his/her email address and password.
2. Credentials are verified.

Scenario 4: View previous scan results

Actors: Web User

Entry Conditions: User is logged in and they click the Previous Results button.

Exit Conditions:

- User closes the app.
- User navigates to a different application section.

Flow of events:

1. User waits for the server request to fetch previous scans.
2. User observes the previous scan results.
3. User may save the list of previous scans.

2.5.2 Use-Case Model

2.5.2.1 Use-Case Model for Desktop

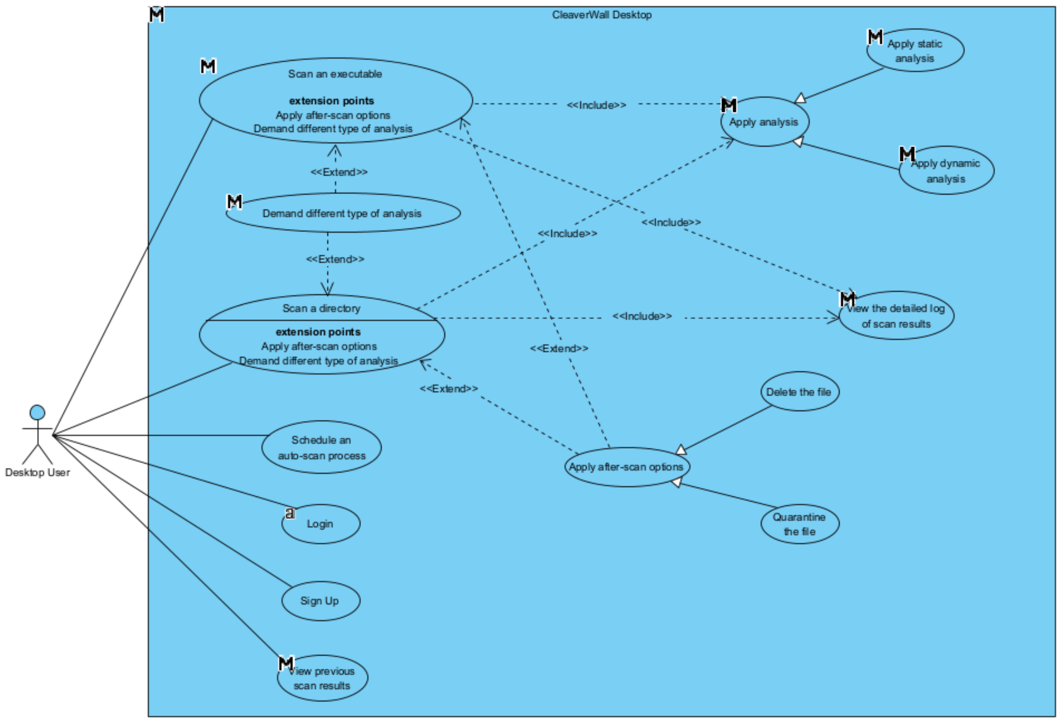


Figure 2: Use-Case Diagram for Desktop

2.5.2.2 Use-Case Model for Web

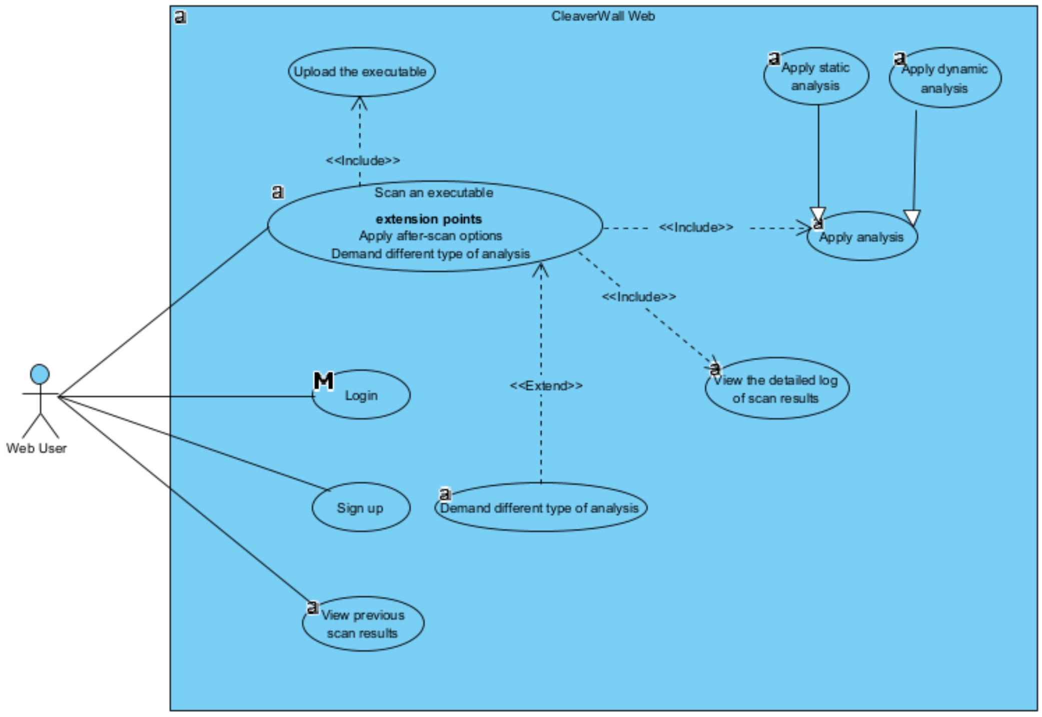


Figure 3: Use-Case Diagram for Web

2.5.3 Object and Class Models

2.5.3.1 Object Design of Server Side

Controller classes will present the API endpoints. Business logic will be implemented in the SubmissionService class. Submission is the only entity to be stored in the database, if the user authentication is excluded. VirtualMachine manager class is left empty. Supporting dynamic analysis is a later stage goal of the project and it is quite hard to predict the requirements for the VM setup as we are still currently working on it

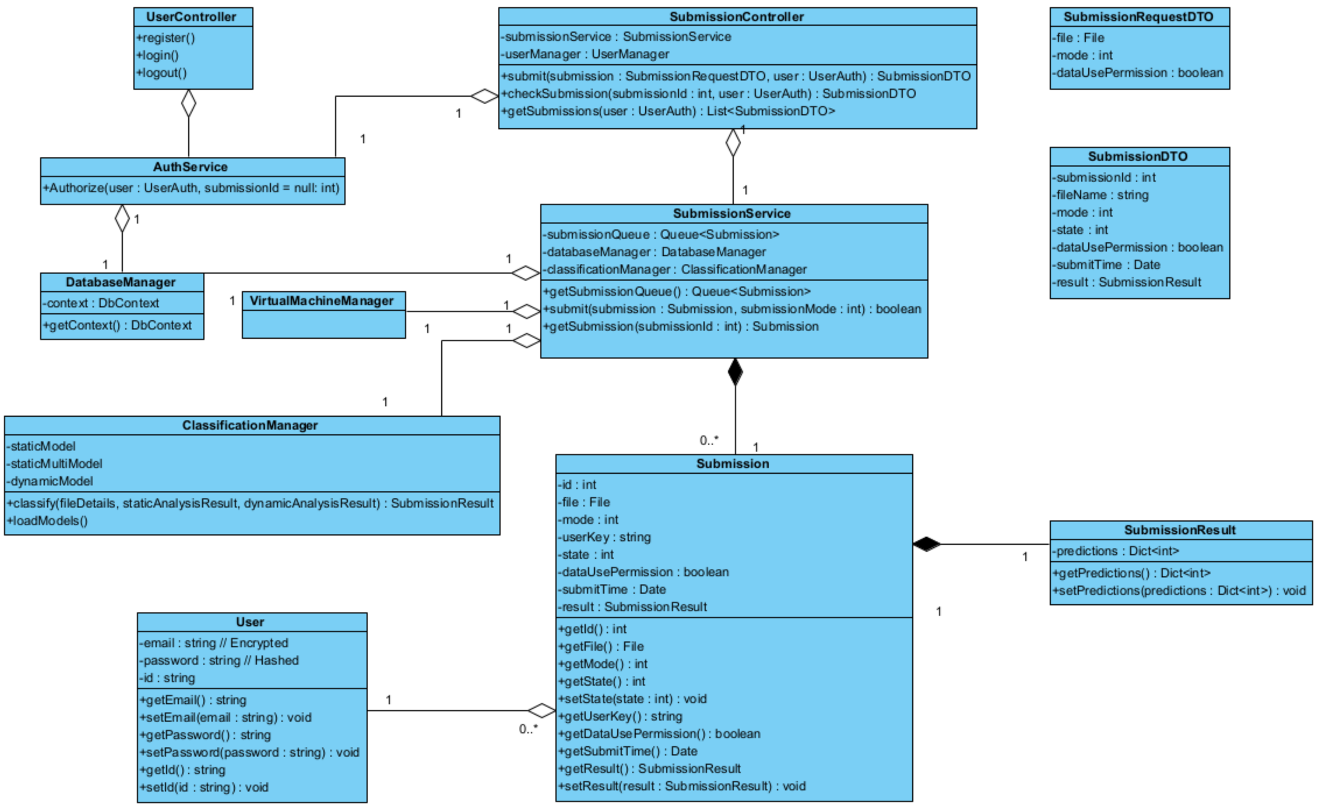

UserController
+register()
+login()
+logout()
SubmissionController
-submissionService : SubmissionService
-userManager : UserManager
+submit(submission : SubmissionRequestDTO, user : UserAuth) : SubmissionDTO
+checkSubmission(submissionId : int, user : UserAuth) : SubmissionDTO
+getSubmissions(user : UserAuth) : List<SubmissionDTO>
SubmissionRequestDTO
-file : File
-mode : int
-dataUsePermission : boolean
SubmissionDTO
-submissionId : int
-fileName : string
-mode : int
-state : int
-dataUsePermission : boolean
-submitTime : Date
-result : SubmissionResult
AuthService
+Authorize(user : UserAuth, submissionId = null: int)
SubmissionService
-submissionQueue : Queue<Submission>
-databaseManager : DatabaseManager
-classificationManager : ClassificationManager
+getSubmissionQueue() : Queue<Submission>
+submit(submission : Submission, submissionMode : int) : boolean
+getSubmission(submissionId : int) : Submission
DatabaseManager
-context : DbContext
+getContext() : DbContext
VirtualMachineManager
ClassificationManager
-staticModel
-staticMultiModel
-dynamicModel
+classify(fileDetails, staticAnalysisResult, dynamicAnalysisResult) : SubmissionResult
+loadModels()
Submission
-id : int
-file : File
-mode : int
-userKey : string
-state : int
-dataUsePermission : boolean
-submitTime : Date
-result : SubmissionResult
+getId() : int
+getFile() : File
+getMode() : int
+getState() : int
+setState(state : int) : void
+getUserKey() : string
+getDataUsePermission() : boolean
+getSubmitTime() : Date
+getResult() : SubmissionResult
+setResult(result : SubmissionResult) : void
SubmissionResult
-predictions : Dict<int>
+getPredictions() : Dict<int>
+setPredictions(predictions : Dict<int>) : void
User
-email : string // Encrypted
-password : string // Hashed
-id : string
+getEmail() : string
+setEmail(email : string) : void
+getPassword() : string
+setPassword(password : string) : void
+getId() : string
+setId(id : string) : void
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0..*
1
1
0..*

Figure 4: Class Diagram of Server Side

2.5.3.2 Object Design of Client Sides

The Web client application will not follow an object oriented approach. It will consist of a tree of functions (React Components), and will occasionally use objects from external libraries in its scripts

The desktop application will follow an Object-Oriented approach mainly for organizing its functionalities into the graphical user interface (GUI). The main functionality will be implemented by the scripts invoked by these classes. The diagram below does not provide further GUI related classes as it is outside the scope of this part of the report, that is given in section 3.5.5.

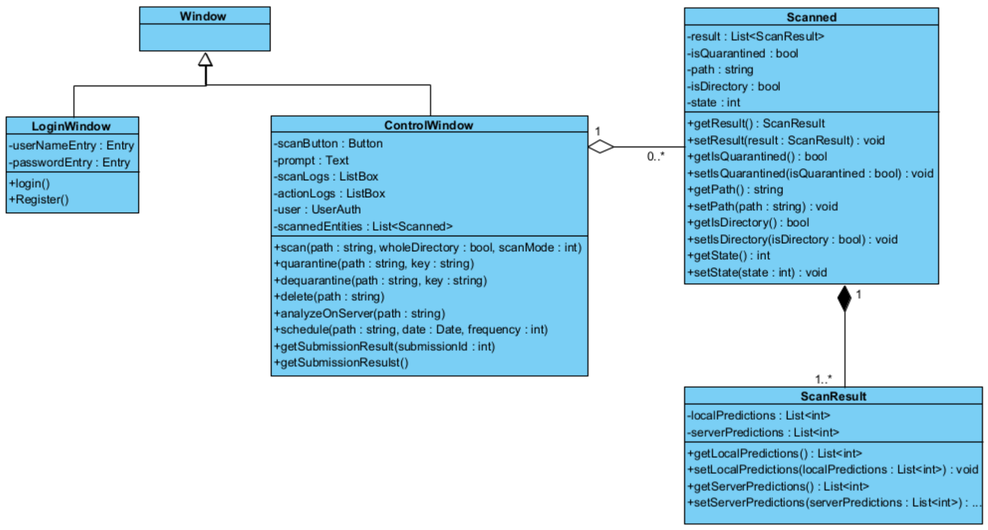

Window
LoginWindow
-userNameEntry : Entry
-passwordEntry : Entry
+login()
+Register()
ControlWindow
-scanButton : Button
-prompt : Text
-scanLogs : ListBox
-actionLogs : ListBox
-user : UserAuth
-scannedEntities : List<Scanned>
+scan(path : string, wholeDirectory : bool, scanMode : int)
+quarantine(path : string, key : string)
+dequarantine(path : string, key : string)
+delete(path : string)
+analyzeOnServer(path : string)
+schedule(path : string, date : Date, frequency : int)
+getSubmissionResult(submissionId : int)
+getSubmissionResulst()
1
0..*
Scanned
-result : List<ScanResult>
-isQuarantined : bool
-path : string
-isDirectory : bool
-state : int
+getResult() : ScanResult
+setResult(result : ScanResult) : void
+getIsQuarantined() : bool
+setIsQuarantined(isQuarantined : bool) : void
+getPath() : string
+setPath(path : string) : void
+getIsDirectory() : bool
+setIsDirectory(isDirectory : bool) : void
+getState() : int
+setState(state : int) : void
1
1..*
ScanResult
-localPredictions : List<int>
-serverPredictions : List<int>
+getLocalPredictions() : List<int>
+setLocalPredictions(localPredictions : List<int>) : void
+getServerPredictions() : List<int>
+setServerPredictions(serverPredictions : List<int>) : ...

Figure 5: Class Diagram for Desktop Application

2.5.4 Dynamic Models

2.5.4.1 Sequence Diagram for Web

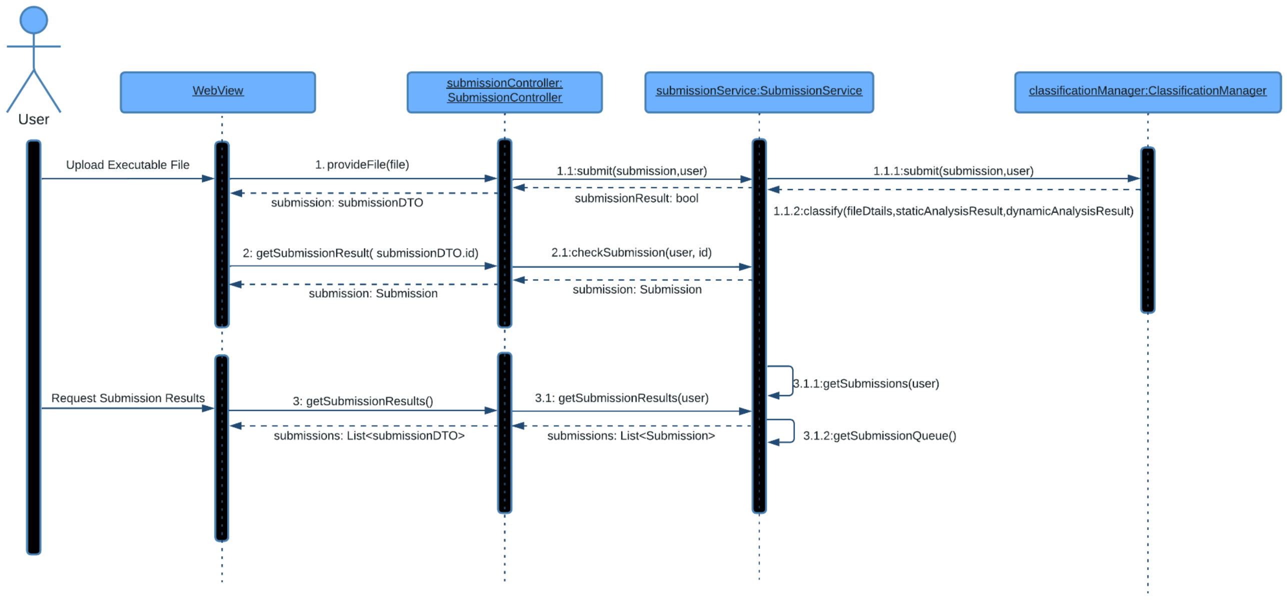

User
WebView
submissionController: SubmissionController
submissionService:SubmissionService
classificationManager:ClassificationManager
Upload Executable File
1. provideFile(file)
submission: submissionDTO
1.1:submit(submission,user)
submissionResult: bool
1.1.1:submit(submission,user)
1.1.2:classify(fileDtails,staticAnalysisResult,dynamicAnalysisResult)
2: getSubmissionResult(submissionDTO.id)
submission: Submission
2.1:checkSubmission(user, id)
submission: Submission
3.1.1:getSubmissions(user)
Request Submission Results
3: getSubmissionResults()
3.1: getSubmissionResults(user)
submissions: List<submissionDTO>
submissions: List<Submission>
3.1.2:getSubmissionQueue()

Figure 6: Sequence Diagram for Web

In this diagram the user uploads an executable file. That uploaded file is analyzed in the server and the result of the analysis will be displayed in the home page when it is ready. Also users can view the previous scans.

2.5.4.1 Sequence Diagram for Desktop

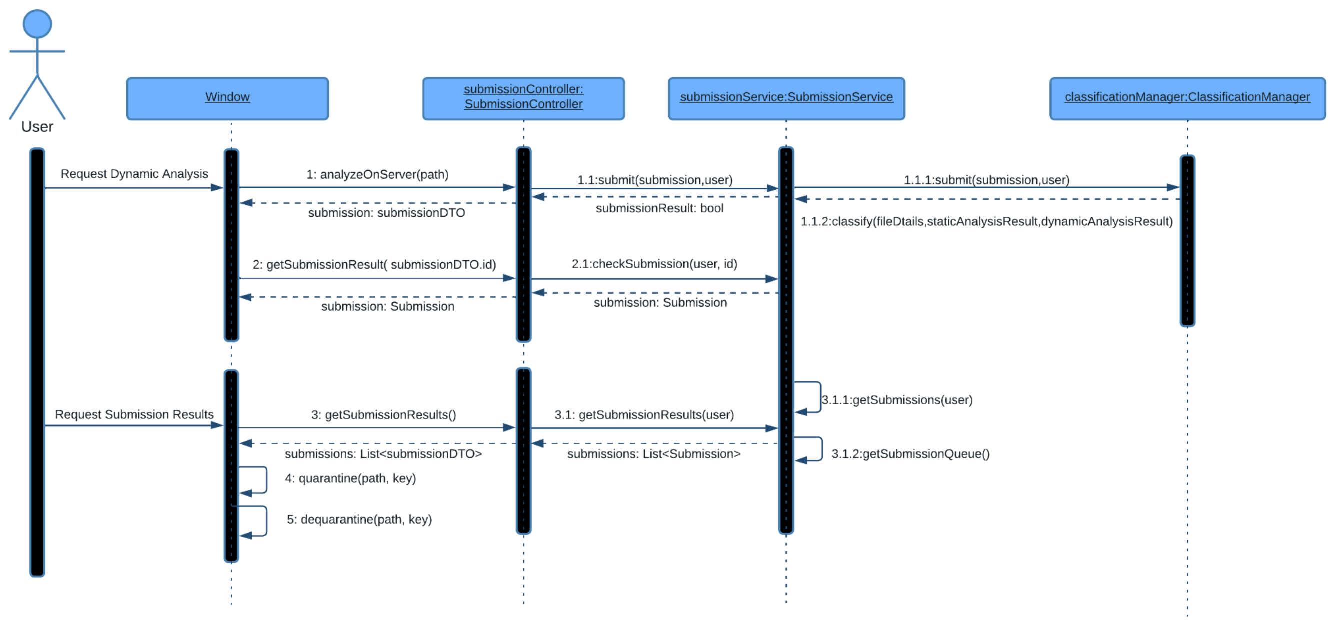

User
Window
submissionController: SubmissionController
submissionService:SubmissionService
classificationManager:ClassificationManager
Request Dynamic Analysis
1: analyzeOnServer(path)
1.1:submit(submission,user)
1.1.1:submit(submission,user)
submission: submissionDTO
submissionResult: bool
1.1.2:classify(fileDtails,staticAnalysisResult,dynamicAnalysisResult)
2: getSubmissionResult(submissionDTO.id)
2.1:checkSubmission(user, id)
submission: Submission
submission: Submission
3.1.1:getSubmissions(user)
Request Submission Results
3: getSubmissionResults()
3.1: getSubmissionResults(user)
submissions: List<submissionDTO>
submissions: List<Submission>
3.1.2:getSubmissionQueue()
4: quarantine(path, key)
5: dequarantine(path, key)

Figure 7: Sequence Diagram for Desktop

In this diagram the user indicates directory name. That directory will be statically analyzed first. If a further analysis is needed it will sended to the server and the result of the analysis will be displayed in the home page when it is ready. Also users can view the previous scans, dequarantine that scans or quarantine the newly analyzed files.

2.5.4.1 Activity Diagram for Web

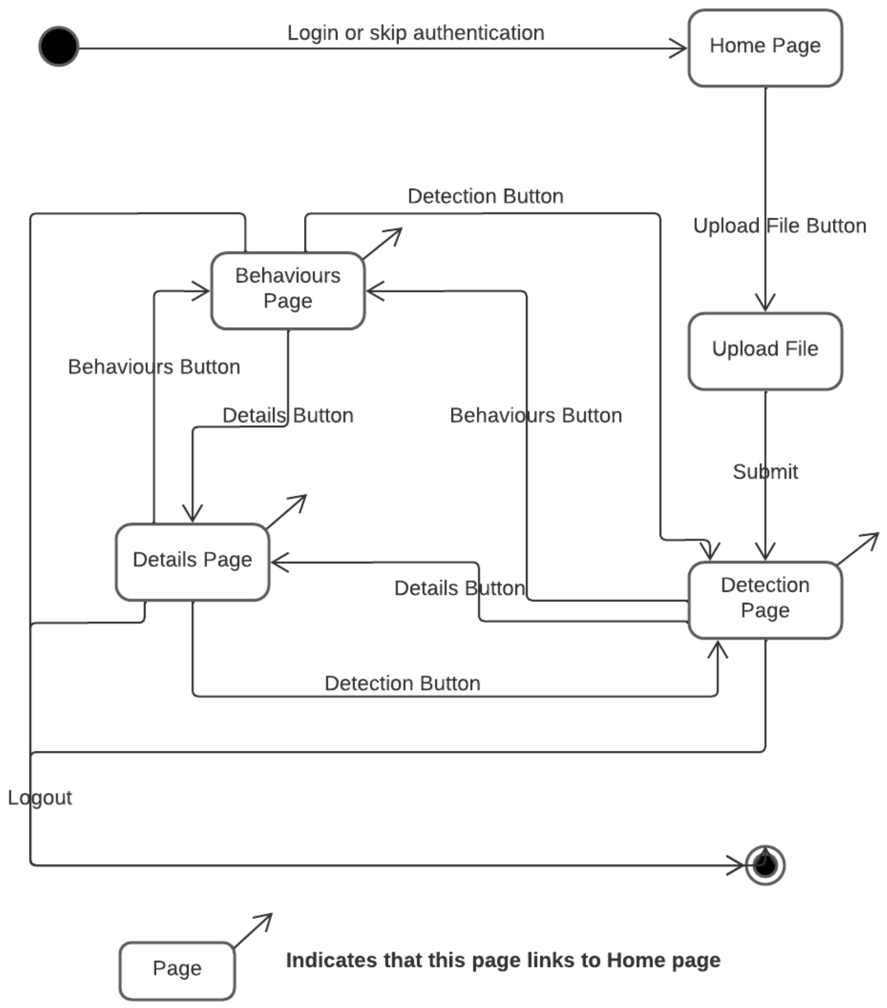


Figure 8: Activity Diagram for Website

2.5.4.1 Activity Diagram for Desktop

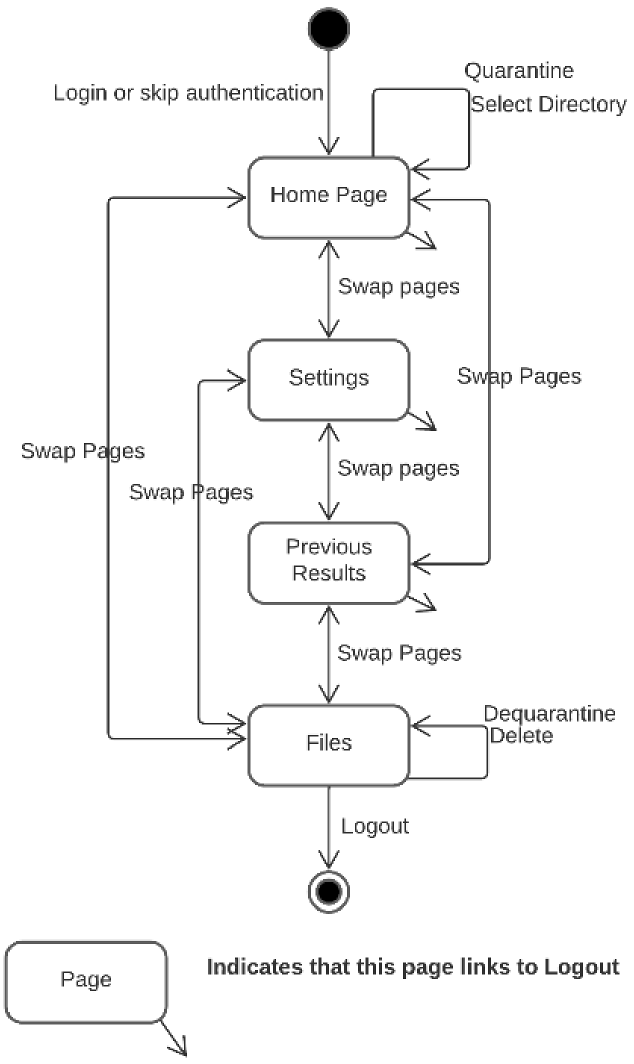


Figure 9: Activity Diagram for Desktop

2.5.5 User Interface

2.5.5.1 Mockups for Web Application

2.5.5.1.1 Login Page

This is the page where the user logs in by his email and password.

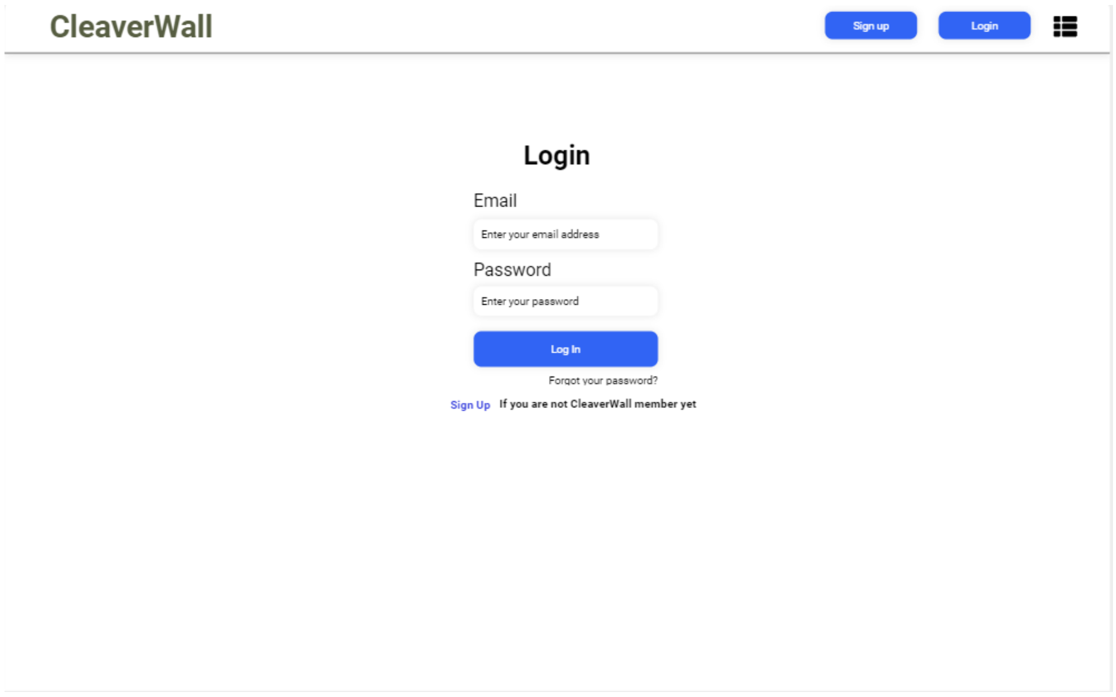


Figure 10: Login Screen for Web

2.5.5.1.2 Sign Up Page

The user can sign up by specifying an email and password for the account.

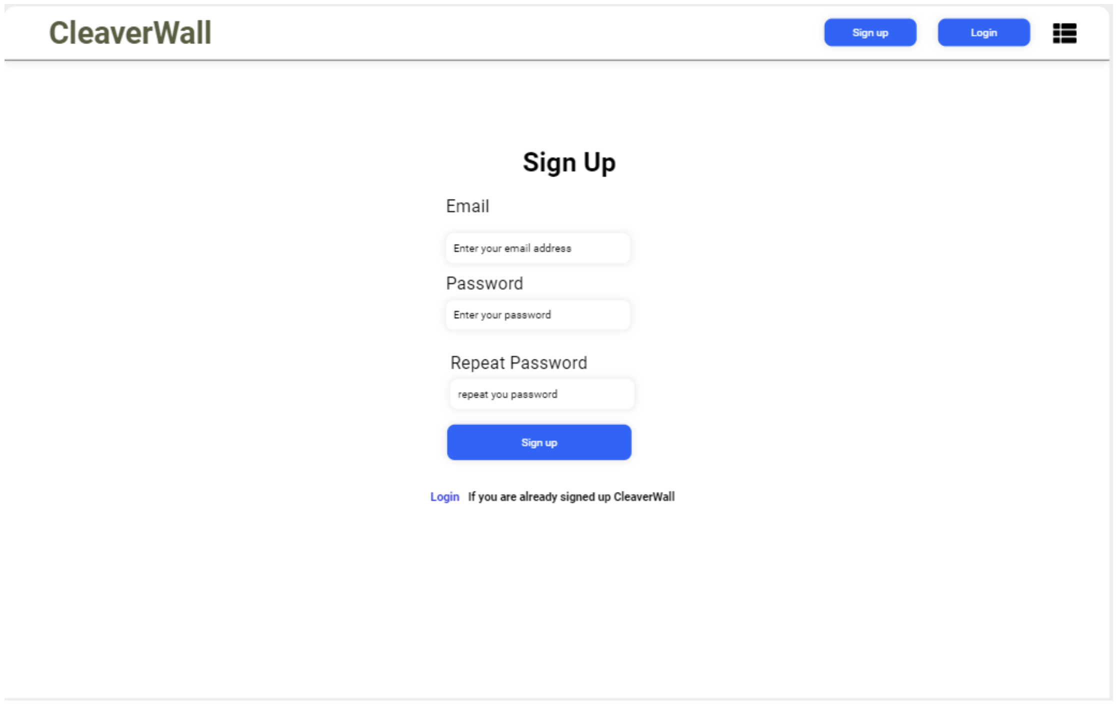


Figure 11: Sign-Up Screen for Web

2.5.5.1.3 Home Page

This is the page where the user uploads the file which will be examined. Also, the page redirects the user to download the desktop application or the Github page of the project.

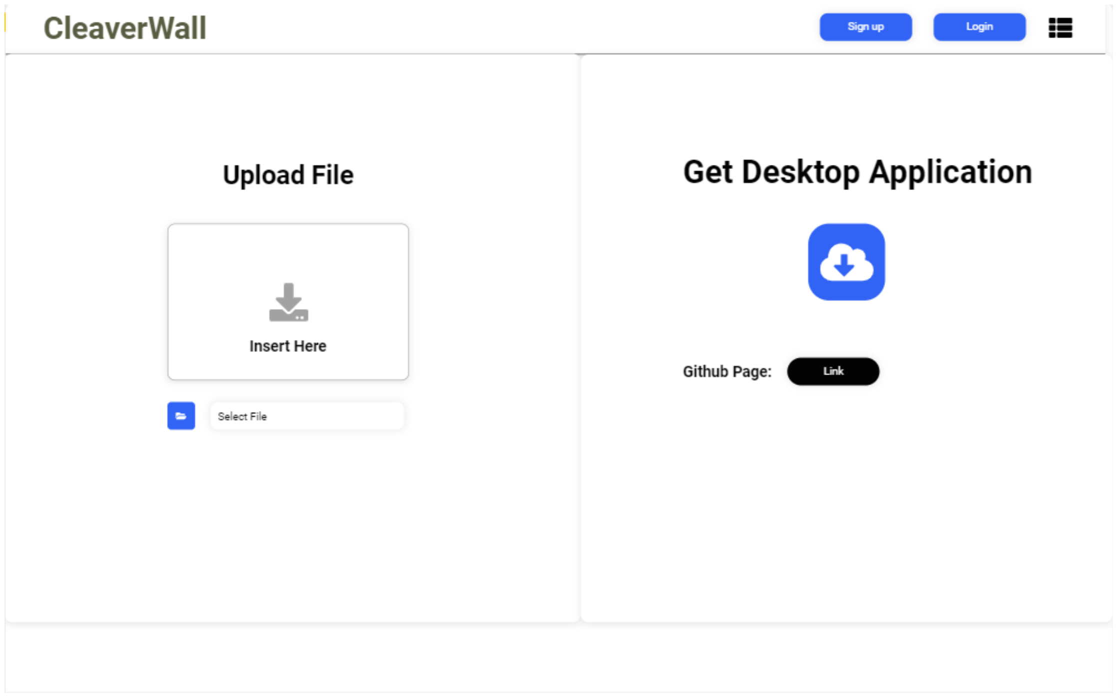
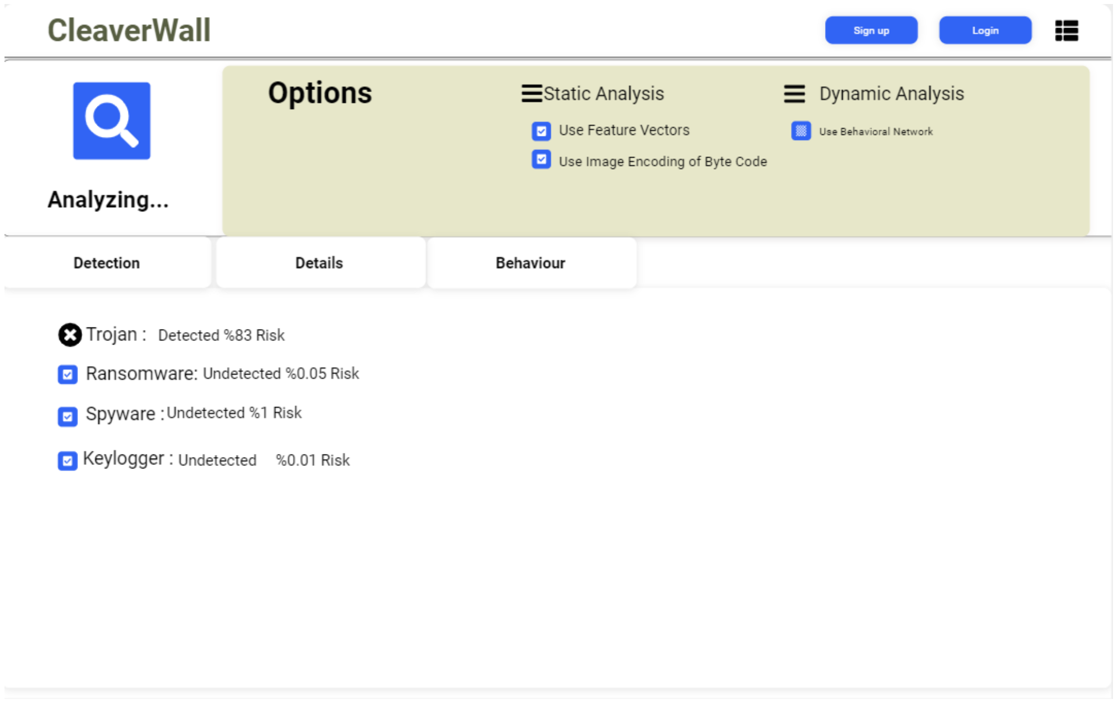


Figure 12: Home Page for Web

2.5.5.1.4 Detection Page

This page demonstrates the results of the machine learning algorithm. Confidence rates if the file contains malware from specific classes are displayed.



Trojan : Detected %83 Risk

Ransomware: Undetected %0.05 Risk

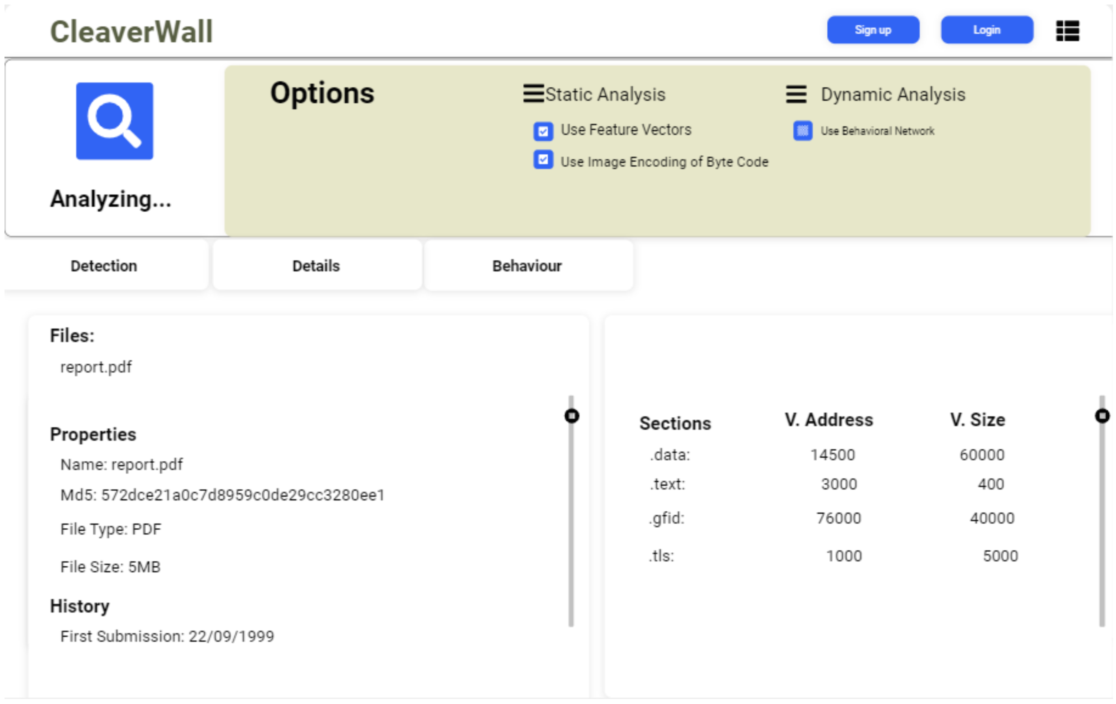
Spyware : Undetected %1 Risk

Keylogger : Undetected %0.01 Risk

Figure 13: Detection Page for Web

2.5.5.1.5 Details Page

This page contains the properties of the file and details that are used for static analysis such as data of sections of the portable executable file.



Use Feature Vectors ✓
Use Image Encoding of Byte Code ✓
Use Behavioral Network

Files:
report.pdf

Properties
Name: report.pdf
Md5: 572dce21a0c7d8959c0de29cc3280ee1
File Type: PDF
File Size: 5MB

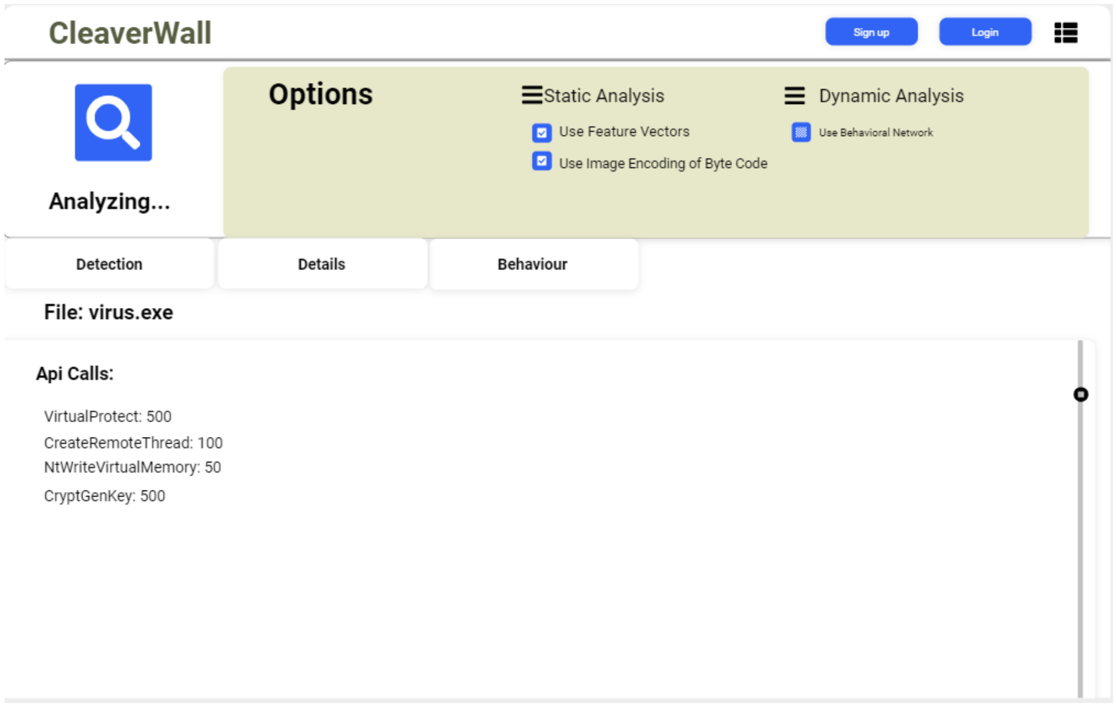
History
First Submission: 22/09/1999

Sections	V. Address	V. Size
.data:	14500	60000
.text:	3000	400
.gfid:	76000	40000
.tls:	1000	5000

Figure 14: Details Page for Web

2.5.5.1.6 Behaviour Page

This page contains the information obtained from the sandbox such as numbers of API calls.



Use Feature Vectors ✓
Use Image Encoding of Byte Code ✓
Use Behavioral Network

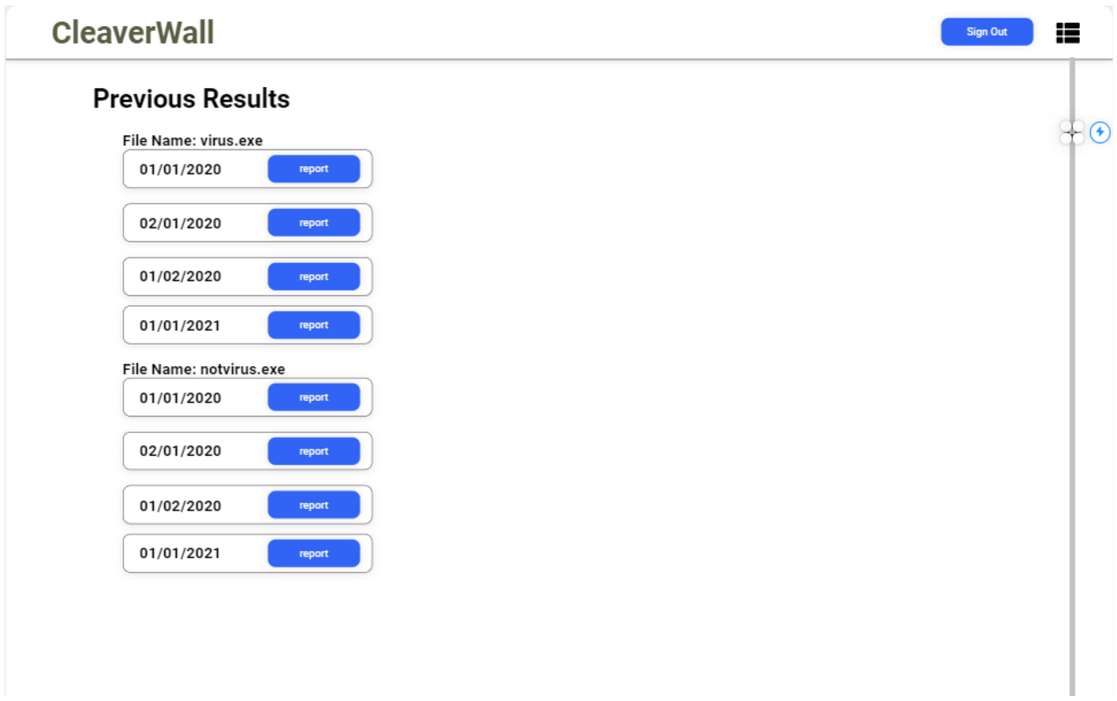
File: virus.exe

Api Calls:
VirtualProtect: 500
CreateRemoteThread: 100
NtWriteVirtualMemory: 50
CryptGenKey: 500

Figure 15: Behaviour Page for Web

2.5.5.1.7 Previous Results Page

This page shows previously investigated files by an account. The user can click on the report button to reach Detection, Details and Behaviour pages of the analysis done on a specified date.



CleaverWall

Sign Out

Previous Results

File Name: virus.exe

01/01/2020 report

02/01/2020 report

01/02/2020 report

01/01/2021 report

File Name: notvirus.exe

01/01/2020 report

02/01/2020 report

01/02/2020 report

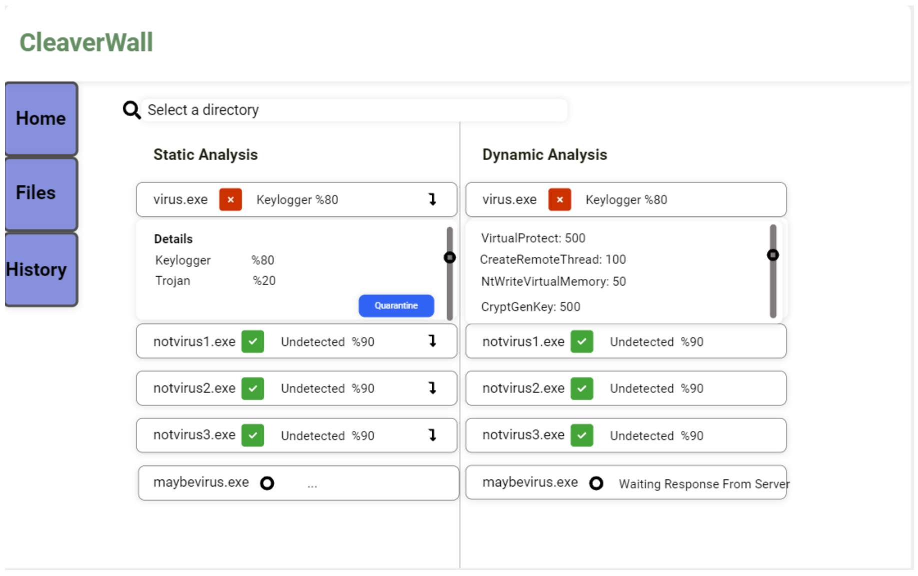
01/01/2021 report

Figure 16: Previous Results Page for Web

2.5.5.2 Mockups for Desktop Application

2.5.5.2.1 Home Page

This is the page managing the operations of malware detection. The user can select a directory to investigate all the files inside the directory. The page is split into two parts. One part shows the results of static analysis and the other part shows the results of dynamic analysis. The users can choose a file to display the details of the information about the file and the results on the same page. Also, files can be quarantined using



CleaverWall

Home

Files

History

Select a directory

Static Analysis

virus.exe ✗ Keylogger %80

Details

Keylogger %80

Trojan %20

Quarantine

notvirus1.exe ✓ Undetected %90

notvirus2.exe ✓ Undetected %90

notvirus3.exe ✓ Undetected %90

maybevirus.exe ...

Dynamic Analysis

virus.exe ✗ Keylogger %80

VirtualProtect: 500

CreateRemoteThread: 100

NtWriteVirtualMemory: 50

CryptGenKey: 500

notvirus1.exe ✓ Undetected %90

notvirus2.exe ✓ Undetected %90

notvirus3.exe ✓ Undetected %90

maybevirus.exe Waiting Response From Server

Figure 17: Home Page for Desktop

2.5.5.2.2 Files Page

This page manages operations about quarantined files. The user can delete a file, take the file out of quarantine or display details about the file using this page.

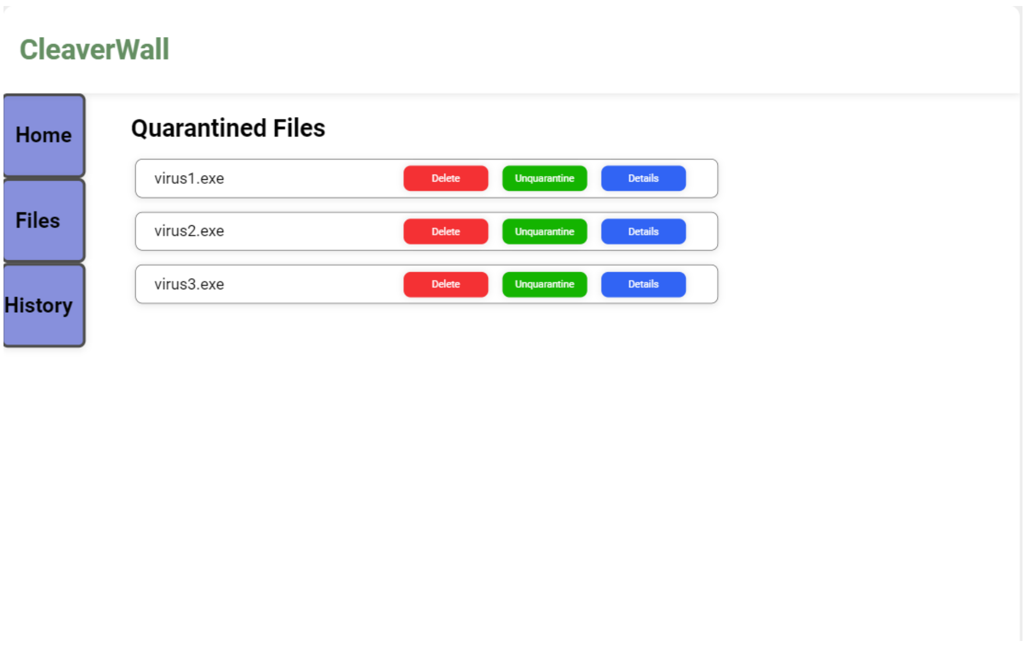
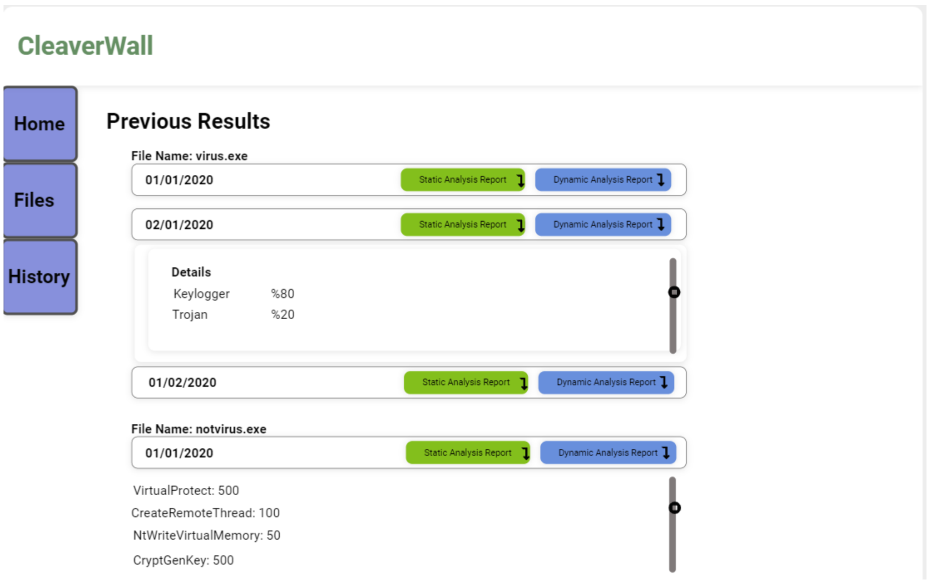


Figure 18: Files Page for Desktop

2.5.5.2.3 Previous Results Page

This page contains information about analyses that are done previously. The page demonstrates detailed information about static and dynamic analyses done in specified date by pop-ups.



File Name: virus.exe

01/01/2020

02/01/2020

Details

Keylogger	%80
Trojan	%20

01/02/2020

File Name: notvirus.exe

01/01/2020

VirtualProtect: 500
CreateRemoteThread: 100
NtWriteVirtualMemory: 50
CryptGenKey: 500

Figure 19: Previous Results Page for Desktop

3 Other Analysis Elements

3.1 Consideration of Various Factors in Engineering Design

3.1.1 Public Health

CleaverWall and public health are not correlated.

3.1.2 Public Safety

Every year, ransomware alone costs the public $20 billion yearly [17]. Although some of these attacks may not be caught by current signature-based anti-viruses, since CleaverWall utilizes quick static model responses for malware detection it could prevent some percentage of these attacks, potentially saving millions of dollars worldwide.

3.1.3 Public Welfare

Governments or agencies are backbones of welfare since they provide items and services to the public. These entities employ a lot of personnel who are doing civil servant jobs, who generally sit at a desk and work on computers 8 to 5. Although these people are expected to use computers very often, they are not always tech savvy and are more likely to fall for malware scams while browsing the internet. Their time is precious to waste on such setbacks and they're better off practicing their finesse for easing the bureaucracy or helping people. CleaverWall is a potential helping hand for these situations by providing a quick feedback for malicious software and preventing people from falling for potential scams. Also since it is open-source, these entities could modify CleaverWall for their own use, resulting in potential widespread use.

3.1.4 Global Factors

Since CleaverWall is open source, its success means that people will dissect and analyze the project. Although it is currently not popular, a successful project that uses machine learning only could light new ideas in the analyzers' heads. Additionally, the file recognition model could expand outside the malware detection application and find new uses in other fields.

3.1.5 Cultural Factors

The project could only have a meta effect on the culture, meaning it could affect the roots it came from: programmers. Although not for scale, the effect could be that of some algorithms often taught in programming. It could set a common ground for some certain set of coders, making them form sentences such as "It's similar to Dijkstra" or "It would be nice to use a Knapsack here".

3.1.6 Social Factors

CleaverWall has nothing to do with any social status such as age, gender, ethnicity, race etc. Therefore social factors are not determining factors for CleaverWall usage. Also it seems like CleaverWall will not have any effect on society.

3.2 Risks and Alternatives

- Both the desktop and the web application might not be implemented before the deadline. If this case occurs, we will focus on the web application implementation.
- Public datasets for malware samples generally have imbalanced distribution. This circumstance might cause high false positive rates despite high training accuracy rates. If this case occurs, we will apply methods such as data augmentation to overcome data imbalance.
- There is a possibility that an executable file is packed in order to hide the information in the disassembled version. For now, our static analysis approaches are vulnerable against it. To detect the packing type and unpack the executable, we can utilize unpack.py script [18]. However, this script runs the executable while unpacking it. Therefore, we might have to change the project structure for static analysis to run this script in a windows virtual machine which is downloaded to our server.

3.3 Project Plan

Table 1: Factors that can affect analysis and design.

	Effect level	Effect
Public health	0	The project has nothing to do with health.
Public safety	4	Static responses of the project are expected to protect its users from obvious scams of easy-viruses.
Public welfare	1	Governments or other agencies might decide to provide the project, or a version of theirs (since it is open source) to their civil servant jobs so that some simple setbacks could be avoided, which in turn could increase their efficiency.
Global factors	5	The project's success could lead to some breakthroughs on file recognition using ML, and even might extend the use outside of malware detection.
Cultural factors	2	If successful, the project may have an antsy effect on the programmer culture.
Social factors	0	The project is unlikely to impact the current era of civilization.

Table 2: Risks

	Likelihood	Effect on the project	B Plan Summary
Time constraint for implementation	3/10	Risk of not implementing both web and desktop sides.	We will focus on the web application.
Dataset Imbalance	5/10	Risk of high false positive rates..	We will apply methods such as data augmentation, undersampling etc.
Packed exe file	5/10	Risk of dealing with packed executable files in static analysis.	We will use public unpack.py script to unpack executables before disassembling them.

Table 3: List of work packages

WP#	Work package title	Leader	Members involved
WP1	Analysis	Onur Korkmaz	Everybody
WP2	Design	Selahattin Cem Öztürk	Everybody
WP3	Malware Classifier Development	Arda Barış Örtlek	Arda Barış Örtlek, Selahattin Cem Öztürk, Onur Korkmaz
WP4	Server Side Development	Yekta Seçkin Satır	Yekta Seçkin Satır, Ali Emre Aydoğmuş, Arda Barış Örtlek
WP5	Client Side Development	Ali Emre Aydoğmuş	Yekta Seçkin Satır, Ali Emre Aydoğmuş, Selahattin Cem Öztürk
WP6	Testing	Selahattin Cem Öztürk	Everybody

WP 1: Analysis			
Start date: *August 15, 2022* **End date:** *October 17, 2022*			
Leader:	*Onur Korkmaz*	**Members involved:**	*Arda Barış Örtlek* *Yekta Seçkin Satır* *Ali Emre Aydoğmuş* *Selahattin Cem Öztürk* *Onur Korkmaz*
Objectives: *Discuss and determine related specifications and requirements for the project.*			
Tasks: ***Task 1.1* <Constraints> :** *Determine implementation, economic, sustainability and resource constraints of the project* ***Task 1.2* <Requirements> :** *Determine functional, non-functional and pseudo requirements of the project.* ***Task 1.3* <System Modals> :** Draw use-case, class and dynamic models, also provide UI mockups.			
Deliverables ***D1.1:*** *Project Specification Document* ***D1.2:*** *Analysis and Requirement Report*			

WP 2: Design			
Start date: November 14,.2022 **End date:** *March 15, 2023*			
Leader:	*Selahattin Cem Öztürk*	**Members involved:**	*Arda Barış Örtlek* *Yekta Seçkin Satır* *Ali Emre Aydoğmuş* *Selahattin Cem Öztürk* *Onur Korkmaz*
Objectives: *Evaluating the Analysis and proposing a detailed system design*			
Tasks: ***Task 2.1 Evaluation of the Analysis*** **:** *Evaluating the analysis part for the design.* ***Task 2.2 High-Level Design*** **:** *Dividing the project into subsystems and defining the design goals of the project* ***Task 2.3 Low-Level Design*** **:** *Determining the functional logic of the applications and giving a detailed description of the system.*			
Deliverables ***D2.1:*** *Design Report*			

WP 3: Malware Classifier Development			
Start date: *November 14, 2022* **End date:** *May 15, 2023*			
Leader:	*Arda Barış Örtlek*	**Members involved:**	*Arda Barış Örtlek* *Selahattin Cem Öztürk* *Onur Korkmaz*
Objectives: ● *Create a malware classifier using features from .asm files.* ● *Create a malware classifier using grayscale image of an executable.* ● *Create a multimodal classifier combining 2 different static classifiers.* ● *Create a malware classifier using Windows API sequence.*			
Tasks: ***Task 3.1 Dataset Labeling*** **:** *Labeling datasets taken from VirusShare by using VirusTotal Academic API.* ***Task 3.2 Implementation of Disassembling*** **:** Implementing disassemble script by using Capstone and Pefile. ***Task 3.3 Feature Extraction for Static Analysis*****:** Implementing feature extraction script which can extract features from PE headers and disassembled executables. ***Task 3.4 Feature Selection for Static Analysis*****:** Apply chi-squared tests to determine best features for classification task. ***Task 3.5 Extracting Byte Data*****:** Extract byte data for every executable in the dataset and apply related operations to create grayscale representation. ***Task 3.6 Classifier Training for Static Analysis*****:** Train related classifiers and adjust their hyperparameters to get higher evaluation metrics. ***Task 3.7 Feature Extraction for Dynamic Analysis*****:** Extract features from the data coming from Cuckoo Sandbox. ***Task 3.8 Classifier Training for Dynamic Analysis*****:** Train classifiers constructed with dynamic analysis and adjust related hyperparameters to get higher evaluation metrics.			
Deliverables ***D3.1:*** *Malware classifiers constructed with static analysis* ***D3.2:*** *Malware classifier constructed with dynamic analysis*			

WP 4: Server Side Development			
Start date: *September 22, 2022* **End date:** *May 15, 2023*			
Leader:	*Yekta Seçkin Satır*	**Members involved:**	*Yekta Seçkin Satır* *Ali Emre Aydoğmuş* *Arda Barış Örtlek*
Objectives: ● Provide an API that enables client side applications to perform malware analysis remotely. ● Utilize a virtual environment to perform dynamic analysis on suspected PE files. ● Store the malware data on user's permission for potential use.			
Tasks: ***Task 4.1*** **API Endpoint and Project Structure:** Initiate the server side application in a sustainable structure. Provide endpoints to enable client-side development parallelization. ***Task 4.2*** **Core Functionality Implementation:** Handle the queue for requests. Apply static analysis. Provide classification. ***Task 4.3*** **Database and User operations:** Implement the CRUD operations on Submission entities with user relations. Implement an authentication process. Store the data accurately for In-app use and to feed to the classifier. ***Task 4.4*** **Utilize the Virtual Environment for Dynamic Analysis:** Utilize and run a Cuckoo Sandbox environment, in a VM. Apply dynamic analysis using the environment. Perform classification using the classier trained for this purpose.			
Deliverables ***D4.1:*** Server application that provides dummy functions ***D4.2:*** Functional server application ***D4.3:*** Server side application with VM			

WP 5: Client Side Development			
Start date: *November 14, 2022* **End date:** *May 15, 2023*			
Leader:	*Ali Emre Aydoğmuş*	**Members involved:**	*Ali Emre Aydoğmuş* *Yekta Seçkin Satır* *Selahattin Cem Öztürk*
Objectives: ● *Provide a clear and efficient UI and UX for the users for both the web and the desktop interfaces.* ● *Link the necessary tools for enabling MVP features such as static analysis and quarantining malicious files to the desktop application.*			
Tasks: ***Task 5.1 Deciding Design Choices***: *Decide which languages to use for web and desktop clients respectively and find out what are best practices in order to obtain a high maintainability standard.* ***Task 5.2 Implementing Web Client***: *Straightforward web development with authentication and file upload functionality.* ***Task 5.3 Implementing Basic Desktop Client***: Create an interface with a flow similar to web client, with dummy advanced features. ***Task 5.4 Linking Desktop Client with Advanced Features***: Search and implement shell functionalities for quarantining and deleting executables. Link static analysis functionality with the application.			
Deliverables ***D5.1:*** *Web Client Interface* ***D5.2:*** *Desktop Client Interface*			

WP 6: Testing			
Start date: *November 14, 2022* **End date:** *May 15, 2023*			
Leader:	*Selahattin Cem Öztürk*	**Members involved:**	*Arda Barış Örtlek* *Yekta Seçkin Satır* *Ali Emre Aydoğmuş* *Selahattin Cem Öztürk* *Onur Korkmaz*
Objectives: In the last work package, implemented parts of the project will be tested.			
Tasks: ***Task 6.1 Testing the Accuracy Rate***: *Malware classifier will be tested to understand the accuracy of the model.* ***Task 6.2 Server and Database Testing*** : *Flow of information between the application, server and the database will be tested.*			
Deliverables ***D6.1:*** *CleaverWall Desktop application.* ***D6.2:*** *CleaverWall Web application.*			

3.4 Ensuring Proper Teamwork

We divide the project into work packages for which everybody in the group is responsible. All of the team members are constantly in contact with each other to pursue the process of the project. There is a Github repository for the codebase of the project to which every team member has access. We are expecting everybody in the group to contribute to this repository.

3.5 Ethics and Professional Responsibilities

- We will cite every academic paper and third party organizations that benefit us.
- We need to get user's permission for adding the features of the executable to the dataset.
- We will not share any data that is uploaded by any user.

3.6 Planning for New Knowledge and Learning Strategies

We are reading academic papers for improving our domain specific knowledge. As we reference some of them in this report, there are a large number of papers that we don't reference but benefit from. Our learning strategies for implementation tools are watching online tutorials and reading related articles. For the machine learning side, we are following Andrew Ng's Machine Learning and Deep Learning Specialization courses. For the backend side, we utilize several Youtube tutorial videos related to Django and Cuckoo Sandbox.

4 Glossary

Malware: Harmful software aiming to cause damage to computer systems.

Sandbox: Testing environment on which potentially harmful softwares can be run safely.

5 References

[1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016.

[2] M. A. U. of Ballarat, M. Alazab, U. of Ballarat, U. of B. V. Profile, S. V. U. of Ballarat, S. Venkatraman, P. W. U. of Ballarat, P. Watters, M. A. D. University, M. Alazab, D. University, T. A. N. University, U. of Technology, and O. M. V. A. Metrics, "Zero-day malware detection based on supervised learning algorithms of API call signatures: Proceedings of the ninth Australasian Data Mining Conference - volume 121," *DL Hosted proceedings*, 01-Dec-2011. [Online]. Available: https://dl.acm.org/doi/10.5555/2483628.2483648. [Accessed: 16-Oct-2022].

[3] Capstone, "The ultimate disassembly framework," *Capstone*, 08-May-2020. [Online]. Available: https://www.capstone-engine.org/. [Accessed: 16-Oct-2022].

[4] Erocarrera, "Erocarrera/pefile: Pefile is a python module to read and work with PE (portable executable) files," *GitHub*. [Online]. Available: https://github.com/erocarrera/pefile. [Accessed: 16-Oct-2022].

[5] Karl-Bridge-Microsoft, "PE format - win32 apps," *Win32 apps | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format. [Accessed: 16-Oct-2022].

[6] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images,” *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11*, 2011.

[7] D. Gibert, C. Mateu, J. Planes, and R. Vicens, “Using convolutional neural networks for classification of malware represented as images - journal of computer virology and hacking techniques,” *SpringerLink*, 27-Aug-2018. [Online]. Available: https://link.springer.com/article/10.1007/s11416-018-0323-0. [Accessed: 16-Oct-2022].

[8] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, “Malware classification with deep convolutional Neural Networks,” *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018.

[9] “Grayscale images of malware samples belonging to different malware ...” [Online]. Available: https://www.researchgate.net/figure/Grayscale-images-of-malware-samples-belonging-to-different-malware-families-in-BIG-2015_fig3_358487073. [Accessed: 16-Oct-2022].

[10] D. Gibert, C. Mateu, and J. Planes, “Hydra: A multimodal deep learning framework for malware classification,” *Computers & Security*, 12-May-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820301462. [Accessed: 16-Oct-2022].

[11] T. G. Kim, B. J. Kang, M. Rho, S. Sezer, and E. G. Im, “A multimodal deep learning method for Android malware detection using various features,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.

[12] Y. Ki, E. Kim, and H. K. Kim, “A novel approach to detect malware based on API call sequence analysis,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 659101, 2015.

[13] “Automated malware analysis,” *Cuckoo Sandbox - Automated Malware Analysis*. [Online]. Available: https://cuckoosandbox.org/. [Accessed: 16-Oct-2022].

[14] *VirusShare.com*. [Online]. Available: https://virusshare.com/. [Accessed: 16-Oct-2022].

[15] *Virustotal*. [Online]. Available: https://www.virustotal.com/gui/home/upload. [Accessed: 16-Oct-2022].

[16] “Tensorflow,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/. [Accessed: 16-Oct-2022].

[17] “Ransomware Statistics in 2022: From Random Barrages to Targeted Hits”, Ivana Vojinovic, dataprot.net. [Online]. Available: https://dataprot.net/statistics/ransomware-statistics/

[18] “Unpack.py: Script using WinAppDbg to automatically unpack malware,” *Malware Musings*, 16-Sep-2014. [Online]. Available: https://malwaremusings.com/scripts/unpack.py/. [Accessed: 13-Nov-2022].