Design Project Final Report

CleaverWall

Arda Barış Örtlek

Ali Emre Aydoğmuş

Onur Korkmaz

Selahattin Cem Öztürk

Yekta Seçkin Satır

Supervisor: Özcan Öztürk

Jury Members: Erhan Dolak, Tağmaç Topal

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. 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 are used. Static and dynamic analysis are conducted to create different feature vectors for the models. Those different classifiers work collaboratively to decrease the false positive occurrences.

CleaverWall consists of 4 labels which are Benign, Trojan, Virus and Worm. Benign is utilized to represent non harmful executables. Other labels represent 3 different types of malware families.

In static analysis, the application has 2 different approaches. The first approach collects features from PE header and classifies the executable according to them.

Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion
SizeOfCode	SizeOfInitializedData	SizeOfUninitializedData	AddressOfEntryPoint	BaseOfCode
BaseOfData	ImageBase	SectionAlignment	FileAlignment	MajorOperatingSystemVersion
MinorOperatingSystemVersion	MajorImageVersion	MinorImageVersion	MajorSubsystemVersion	MinorSubsystemVersion
SizeOfImage	SizeOfHeaders	CheckSum	Subsystem	DllCharacteristics
SizeOfStackReserve	SizeOfStackCommit	SizeOfHeapReserve	SizeOfHeapCommit	LoaderFlags
NumberOfRvaAndSizes	SectionsNb	SectionsMeanEntropy	SectionsMinEntropy	SectionsMaxEntropy
SectionsMeanRawsize	SectionsMinRawsize	SectionMaxRawsize	SectionsMeanVirtualsize	SectionsMinVirtualsize
SectionMaxVirtualsize	ImportsNbDLL	ImportsNb	ImportsNbOrdinal	ExportNb
ResourcesNb	ResourcesMeanEntropy	ResourcesMinEntropy	ResourcesMaxEntropy	ResourcesMeanSize

ResourcesMinSize	ResourcesMaxSize	LoadConfigurationSize		

Table 1: Feature list for the first static 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 families are similar to each other while they are distinct from images of other families [1], [2], [3].

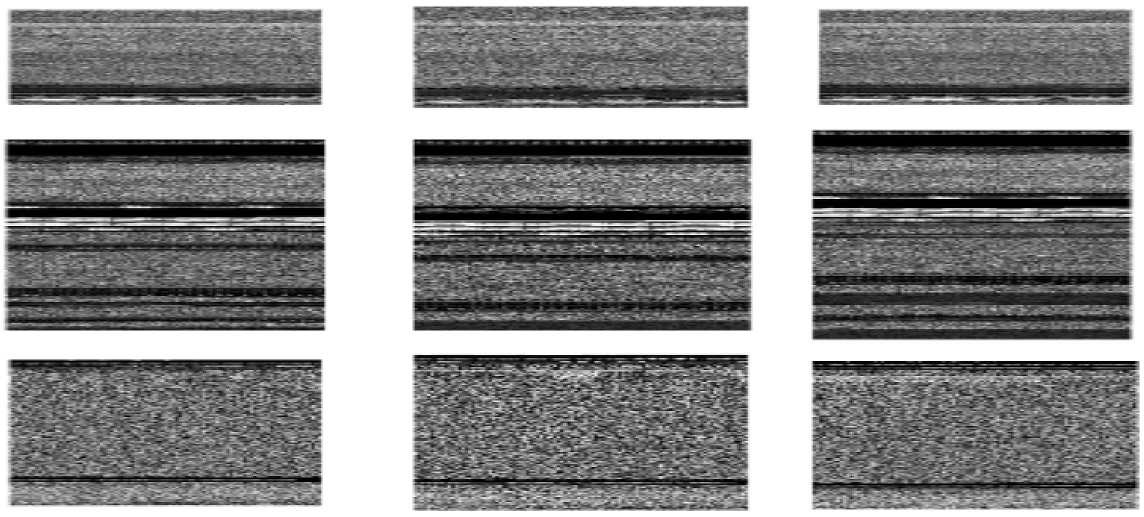


Figure 1: The first row represents Yuner.A samples, the second row represents VB.AT samples and the third row represents Skintrim.N samples [4].

In dynamic analysis, a portable executable's Windows API call sequence is analyzed to create the feature vector [5]. For this purpose, an executable is run on a sandbox for an amount of time. After that, the sandbox returns information regarding the API call sequence. Cuckoo Sandbox [6] is utilized as the sandbox.

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. As we improve our application, we want to change the case that reliability is expensive.

2. Requirements Details

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.

1.1 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 newly 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.

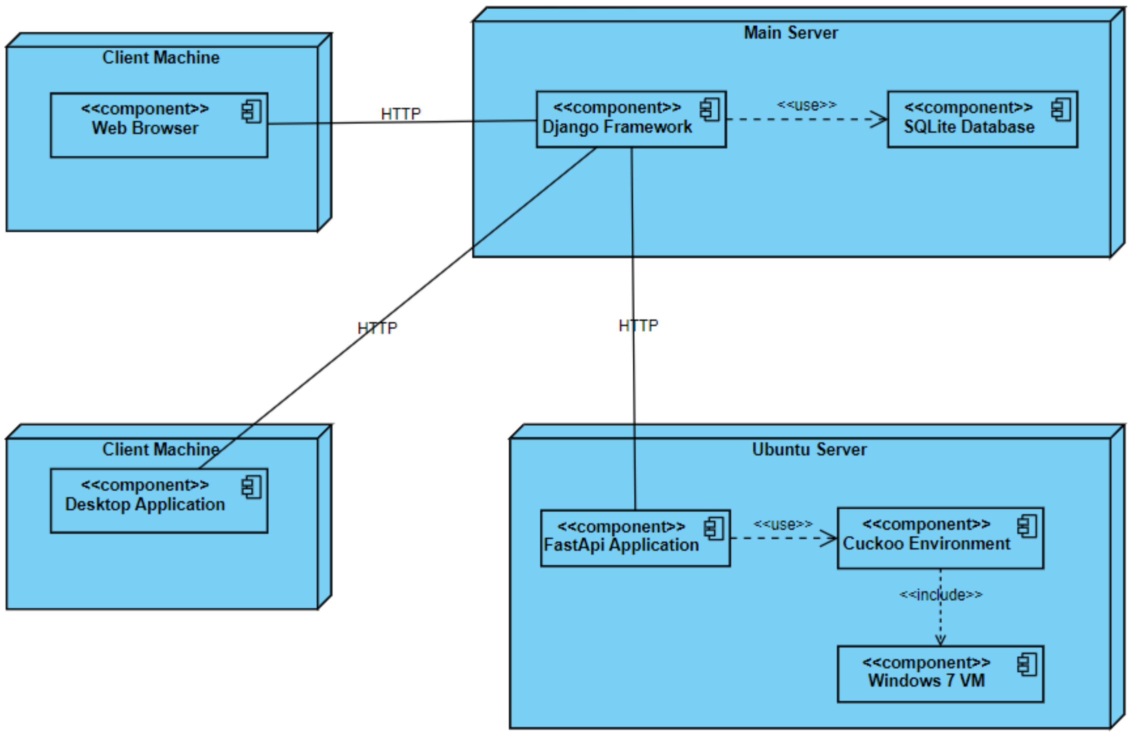
2.2.5 Maintainability

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

3. Final Architecture and Design Details

We have used several technologies in the development of the project. For the client side, we have used Flutter to implement the user interface and controller logic. Flutter provides building the application for both web and desktop clients. For the main server side, we have used Django Framework to implement the back end. Any requests for static analysis and information about past submissions are handled on the main server. The data about submissions and users are stored in an SQLite database. Two static analysis models that classify the file by the header information and image recognition reside on the main server. Any dynamic analysis request is directed to the Ubuntu server by an HTTP request. The back end for the Ubuntu server is implemented using the FastApi framework since the workload for the back end is low. The information required for the Dynamic analysis model is obtained by Cuckoo Sandbox. The sandbox runs the portable executable file on a Windows 7 virtual machine that is emulated by the QEMU framework. Then, it sends the information about the execution like the number of API calls to the management logic which classifies the file by the dynamic analysis model. Finally, the result is stored in the main server and sent to the client. The models are implemented by Tensorflow and optimized using XGBoost.

Here is the diagram for the final architecture. The subsystem decomposition and its services are explained in the next section.



Client Machine
<<component>> Web Browser
HTTP
Main Server
<<component>> Django Framework
<<use>>
<<component>> SQLite Database
HTTP
HTTP
Client Machine
<<component>> Desktop Application
Ubuntu Server
<<component>> FastApi Application
<<use>>
<<component>> Cuckoo Environment
<<include>>
<<component>> Windows 7 VM

Figure 2: Diagram for final architecture: Hardware-Software mapping.

3.1 Subsystem decomposition

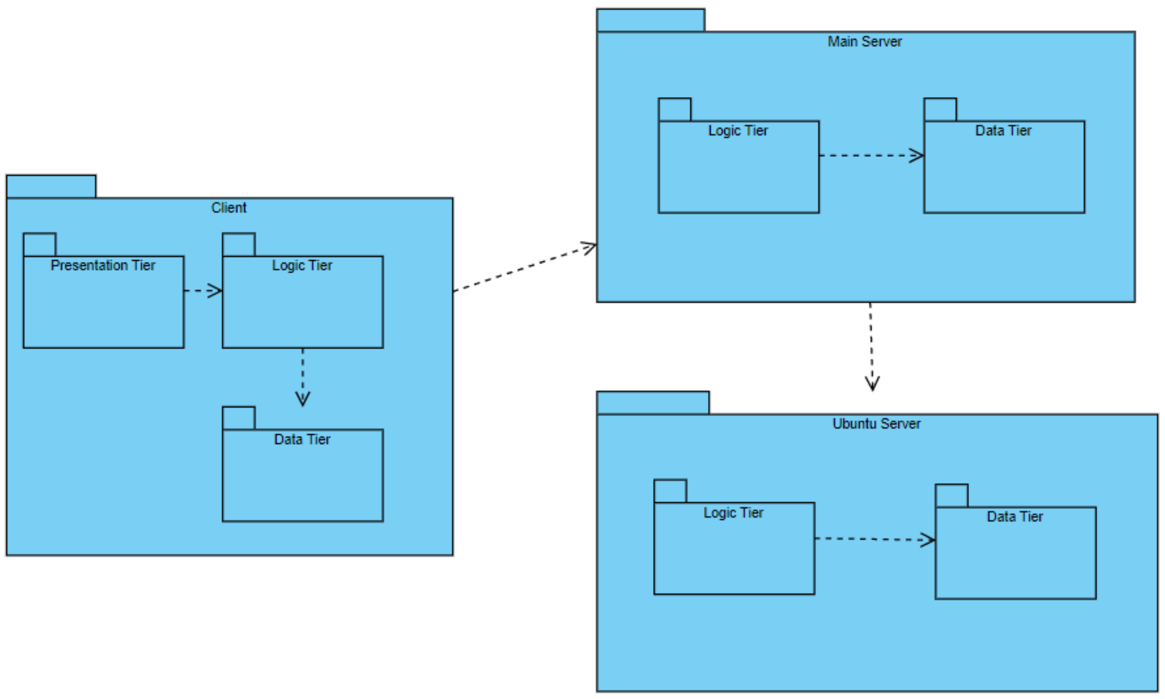


Figure 3: Subsystem decomposition of the whole system.

CleaverWall is based on server-client architecture. There are two types of applications on the client side, the web application and the desktop application. Both sides are being developed using Flutter. There are three subsystems on the client side, Presentation Tier and Logic Tier, and Data Tier. Presentation Tier will handle basic user interface functionalities. Specifically, it is the layer of widgets and states environment in the Flutter framework. Logic Tier will handle processing the data sent by the Main Server, logging in, and switching between pages. Data Tier will handle repositories and data structures that are used to illustrate in the view. After developing the web client, the code will be transferred to the desktop application using Flutter's service. Then, Logic Tier of the desktop application will be extended for more functionalities such as offline static analysis.

There are two servers that will be used in the project. A main server will be used to communicate with the client, maintain crud operations in the database, do static analysis, and communicate with the side server when the dynamic analysis is needed. Main Server is decomposed into two subsystems, Logic Tier and Data Tier. Logic Tier contains the core logic of the application. Data Tier handles the storage of information of users and submissions. The second server handles the operations of dynamic analysis. Because Cuckoo Sandbox needs an Ubuntu environment to run, the side server will run with Ubuntu. The side server is also decomposed into two subsystems, Logic Tier and Data Tier. Logic Tier of the Ubuntu server actuates the Cuckoo Sandbox and operates on it by requests from the Main Server. After getting results, it runs the dynamic analysis model to do classification, then sends the outputs to the main server. Data Tier of the Ubuntu server stores requests from the Main server temporarily.

3.2 Subsystem Services

3.2.1 Client

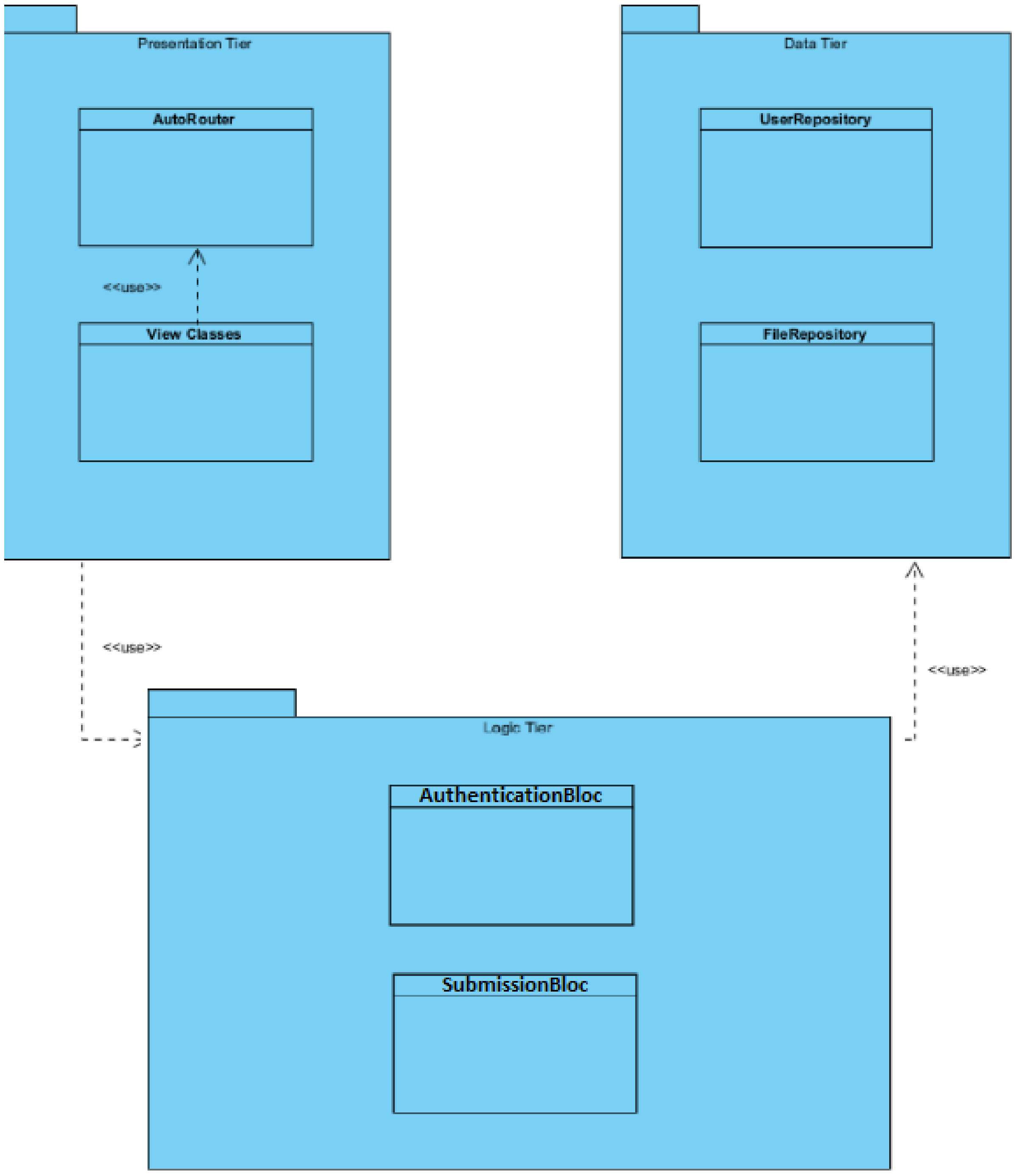


Figure 4: Subsystem decomposition of the Client Side.

CleaverWall client consists of mobile and desktop, however at its current iteration they both have the same functionalities. The client subsystem is divided into three tiers: Presentation, Logic and Data. Presentation tier is used to display and allow the user to interact with the project. While the Presentation tier is a dummy, Logic tier handles all the functionalities. Data tier works mostly like a storage, and is responsible for both managing local temporary files and requesting data from the server, as well as sending data. The presentation tier interacts with the Logic tier in order to make the UI functional, and the Logic tier interacts with the Data tier to make the functionality meaningful.

3.2.1.1 Presentation Tier

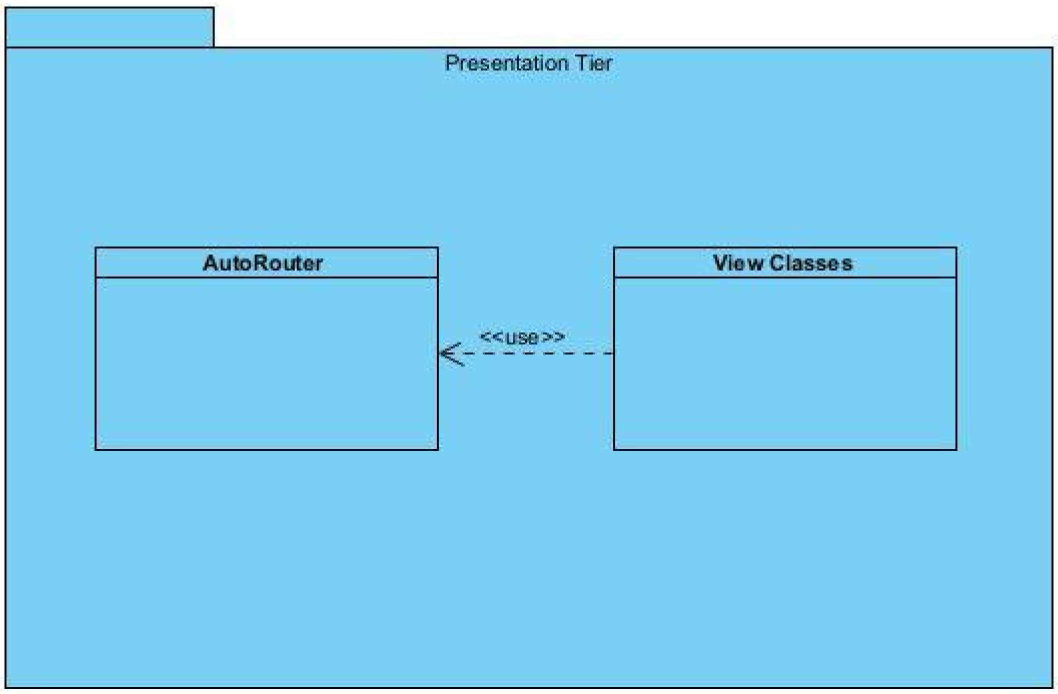


Figure 5: Subsystem decomposition of the Client's Presentation Tier.

The Presentation tier consists of AutoRouter and all the view classes. Autorouter is a flutter library that allows easy transition between the views. Additionally, it automatically handles page URLs, making it more user friendly on the web-side. View classes are flutter classes that return build functions that build the UIs as described previously with the mock UIs. They contain no functionality whatsoever -other than navigating through the app and viewing server data etc.- , except for the uploadFileRoute, which requires the file upload pop-up.

3.2.1.2 Logic Tier

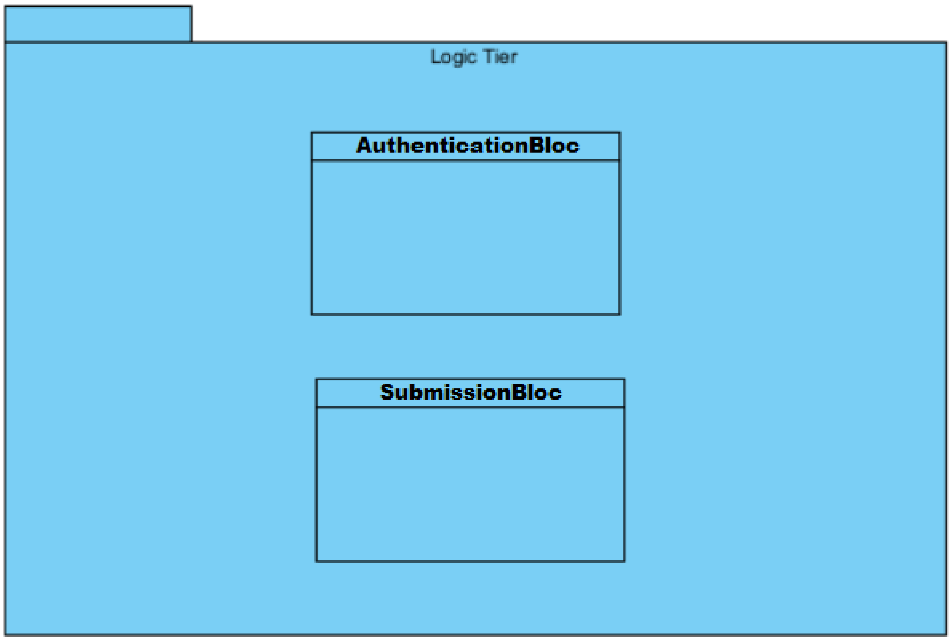


Figure 6: Subsystem decomposition of the Client's Logic Tier.

The Logic tier consists of business logic components (blocs) matching the functionality contexts.

Blocs consist of 3 classes: bloc, state and event. States are blocs' mutable variable storage. Every bloc has only one state. Events are fired through the UI on specific interactions and are used to notify the bloc that it needs to do something. Bloc itself is where the logic runs: it tells the repositories to change or to request some data, and then modifies its states. In return, UIs listening to the related bloc update themselves according to the new bloc state.

AuthenticationBloc: This bloc handles the user login and signup. Also includes validation logic for text boxes on both login and signup. Renamed from **UserActionsBloc.**

SubmissionBloc: The previous FilesBloc and Analysis Bloc have been merged into one SubmissionBloc. It is responsible for temporary file data storage, and for requesting analysis results from the server.

3.2.1.3 Data Tier

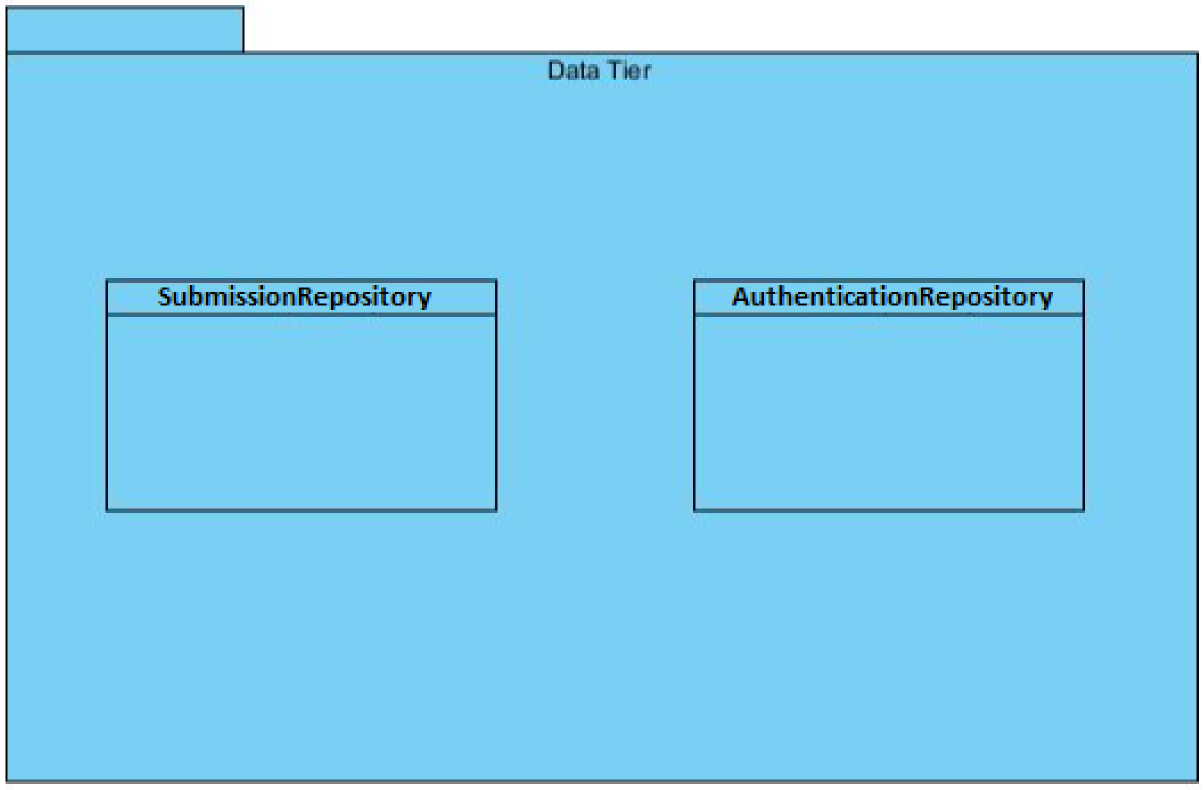


Figure 7: Subsystem decomposition of the Client's Data Tier.

The Data tier stores and requests data and delivers it to the blocs whenever necessary.

AuthenticationRepository: This repository handles the current user data and requests such as storing the user token and requesting a login. Renamed from **UserRepository.**

SubmissionRepository: This repository is responsible for storing the uploaded file until it's sent as well as the analysis data that is received from both the latest upload and from the analysis history. Both analysis and files are currently managed in one repository because of a potential future merge on the analysis and file classes. Renamed from **FileRepository.**

3.2.2 Main Server

3.2.2.1 Logic Tier

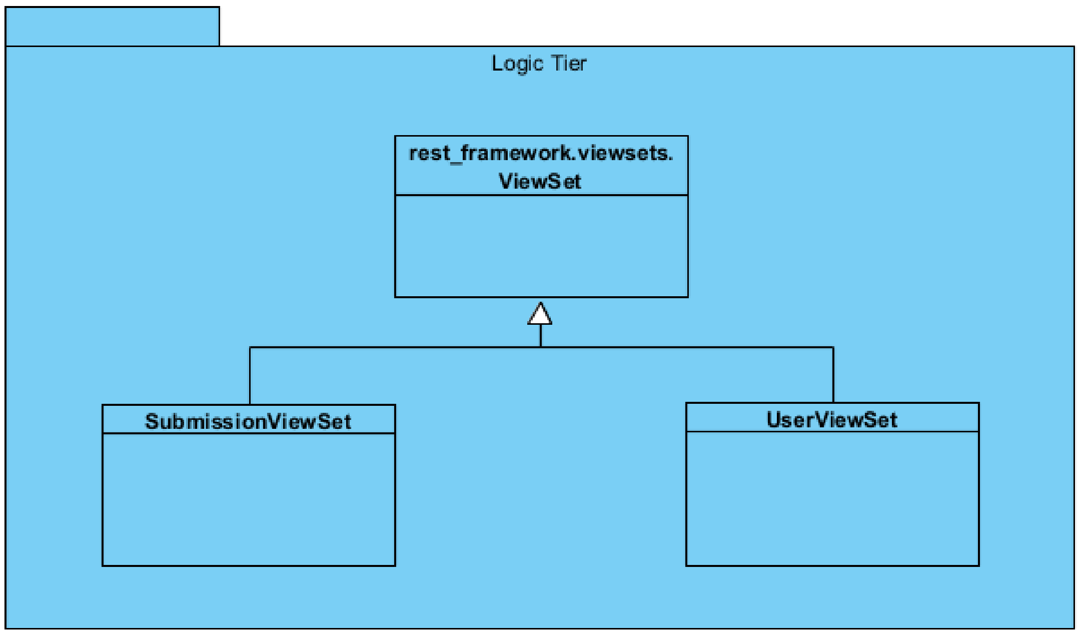


Figure 8: Subsystem decomposition of the Main Server's Logic Tier.

Logic tier of the server is responsible for business decisions: using different modes, holding states during script executions, communication with the Ubuntu server when needed. It is the equivalent of Controller classes in other popular backend frameworks such as ASP.NET. This layer also provides endpoints for clients in sets.

- UserViewSet: Handles the registration, login, and logout operations for the user.
- SubmissionViewSet: Implements create, list, and retrieve functionalities for Submission objects. Accesses miscellaneous utils files and scripts to execute analysis. If the submission mode requires a simple type of analysis, classifies and returns the result. Else requests the Ubuntu server to do more expensive operations.

3.2.2.2 Data Tier

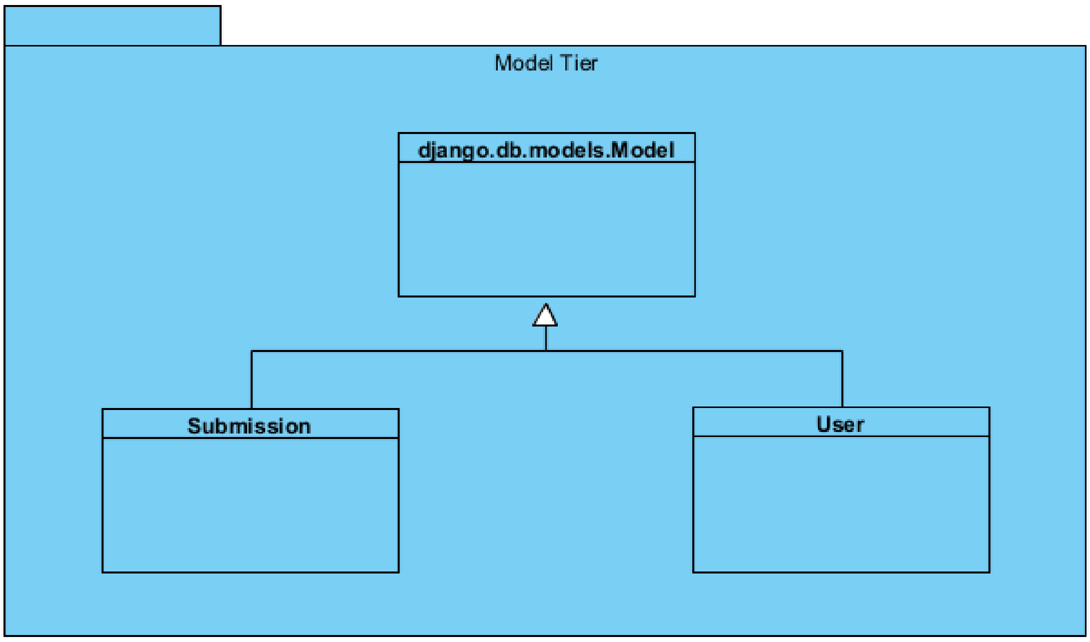


Figure 9: Subsystem decomposition of the Main Server's Data Tier.

Data tier of the main server consists of model classes. It defines the entities and their fields for the SQLite engine.

- User: Holds basic user data
- Submission: Holds information about submissions regarding the file, submission details, results and whether they are still valid.

3.2.3 Ubuntu Server

3.2.3.1 Logic Tier

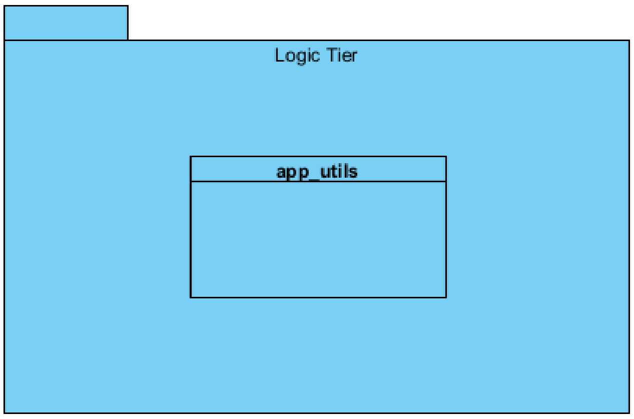


Figure 10: Subsystem decomposition of the Ubuntu Server's Logic Tier.

Logic tier of the Ubuntu server is responsible for managing expensive operations with the Cuckoo sandbox and ensuring a stable communication with it and the main server.

- app_utils: Only the class of this application that contains business logic. Manages analysis requests, and contains functions to communicate with the Cuckoo sandbox and apply classification.

3.2.3.2 Data Tier

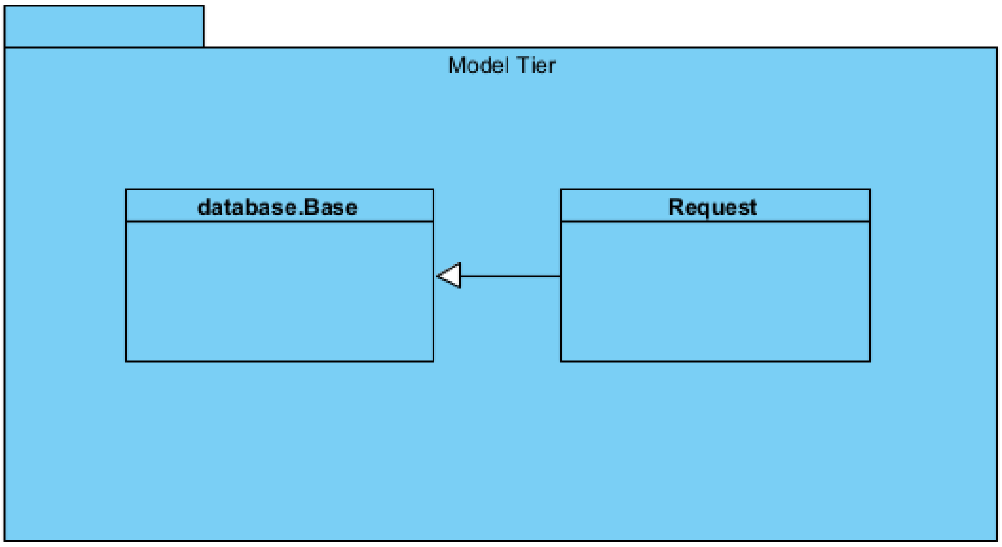


Figure 11: Subsystem decomposition of the Ubuntu Server's Data Tier.

Data tier of the Ubuntu server is only to store Requests for practicality.

- Requests: Holds basic information about current requests temporarily.

4. Development/Implementation Details

4.1 Client Side

The client-side uses Flutter. Other than the small helper packages -which can be peeked at the pubspec.yaml file-, the project has three major dependencies: AutoRoute, Dio and Bloc.

AutoRoute is responsible for the navigation and navigation management throughout the app. Through AutoRoute the project can easily create new views and integrate them into the navigation tree. Also it enables a more clean URL management for the web client.

Dio is used for requests to the server-side. It excels more than the other request management packages since it handles asynchronous requests better and it is more easily traceable/debuggable with its interceptor availability.

Bloc is used for abstracting the client-side structure. Everything the user interacts with is dumb, thanks to Bloc. Rather, they fire the corresponding event for the related bloc, and the bloc handles the functionality for the UI. This does not only mean that there are no direct “onPressed()” methods on the interface, but also that the interface never interacts with the repositories directly. Such an abstraction opens up space for scalability.

The repositories also have their corresponding APIs. All APIs in the app utilize the same Dio client. The reason why APIs are separated from the repositories is so that one part of the app is only responsible for requests, and the other is responsible for handling/processing the results of those requests.

In the end the client-side request structure looks along the lines of:

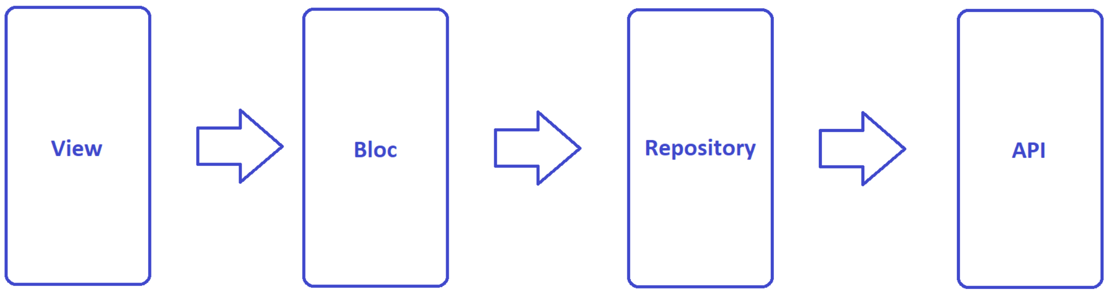


Figure 12: Client-side request structure

4.2 Server Side

On the server-side, we have 3 servers. Two are running backend applications we created, we call: Main server and Ubuntu server. Main server is written with Django Rest Framework, and the Ubuntu server is written with FastAPI Framework. Third one is the Cuckoo server running on the same machine with the Ubuntu server. It is implemented with Python2 by Cuckoo Foundation.
The Cuckoo server is responsible for dynamically analyzing (physically running the suspected file on our Windows 7 virtual machine) the suspected file, and returning the features to be fed to the machine learning model such as the Windows Api call sequence. The Ubuntu server runs the smaller backend application we created using FastAPI, a lightweight python backend framework. It is responsible for managing the Cuckoo server and providing an interface for it to outside of the machine. Because the Cuckoo server is not very stable, the Ubuntu server also benefits by separating a failure point without any significant overhead. Since Cuckoo server strictly runs in an Ubuntu machine, we named the FastAPI server managing it the Ubuntu Server. The Ubuntu server uses a sqlite3 server to store and log its requests.
The general application logic is implemented in the Main server as the Django rest Framework provides a scalable structure for applications with python. It uses an approach similar to Model-View-Controller. Data layer is implemented in model files, views files provide programmer interface to the application. The way business logic is implemented is

left up to the programmer and varies between conventions. Main server uses Django Database which also is implemented over sqlite3. Additionally, both server-side applications use various libraries related to data structures, databases, operating systems, machine learning, and http.

4.3 Machine Learning Side

The malware executable dataset is taken from VirusShare and Academic API of VirusTotal is used to label the dataset. Because of copyright issues, there are not huge datasets including benign executables. There are repositories such as PortableFreeware but creating a dataset by downloading executables from them is not practical. Therefore, we wrote a script to find all of the executable paths in our PC's. Then, extracted features from them to create a benign dataset.

Google Colab Pro is utilized as the training environment because it provides Nvidia A100 GPU, which facilitates the training process. Ubuntu Virtual Machines are used for both storing the malware executables and sandbox operations. Feature extraction for the training is done inside one of the virtual machines.

For the first static model, XGBoost and Feed Forward Neural Network Architectures are trained and their performance metrics are compared. Neural Network achieves 0.9636 validation accuracy and 0.06 false positive rate. XGBoost achieves 0.9835 validation accuracy and 0.04 false positive rate. Therefore, XGBoost is selected as the classifier for this static approach.

For the second static model, transfer learning method is used. The chosen base model for this project is ResNet50, which has demonstrated strong effectiveness in various computer vision tasks such as image classification, object detection, and segmentation. It has achieved outstanding performance on benchmark datasets like ImageNet and COCO. To align with ResNet50's input shape of (224,224,3), a Conv2D layer with 3 filters and a 3x3 shape is added before the base model. This conversion transforms the input shape from (224,224,1) to (224,224,3). The output layer of the base model is modified to a softmax layer with 4 units to match the dataset of this project. The base model utilizes pre-trained weights. This model achieves 0.9344 validation accuracy and 0.11 false positive rate.

Although the performance metrics of the second model is worse than the first model, its operation elapsed time is less than the first model's elapsed time for large sized executables. Therefore, users can decide between the more accurate response and the quicker response for their needs.

For the dynamic model, XGBoost is used. The model achieves 0.99 validation accuracy and there is not any false positive among 6586 benign executables in the validation set.

5. Test Cases and Results

5.1 Client Side Test Cases

Client-side tests are done through Android Studio's Flutter testing, where emulating necessary conditions and mocking data is made easier.

Test ID	CLS001
Test Type/Category	Authorization
Title	Successful Login
Procedure of testing steps	1. Have an unauthorized session. 2. Send a request to the Django server with credentials that are known to be true. 3. See that the response is successful and update the local token accordingly.
Expected results	Server should return a token upon sending it the right credentials, and the token should be cached locally to be able to use authorized functions.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS002
Test Type/Category	Authorization
Title	Token Expiration
Procedure of testing steps	1. Have an authorized session. 2. Send any request to the Django server that requires authorization. 3. See that the response is unsuccessful. 4. End the current session. 5. Clear the local token.
Expected results	Server should not do the operation and return the unauthorized statement. Upon doing so, the user will log out from the current session and the local token will be cleared from the cache.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS003
Test Type/Category	Authorization
Title	Manual Logout
Procedure of testing steps	1. Have an authorized session. 2. Send a logout request. 3. See that the response is successful. 4. End the current session. 5. Clear the local token.
Expected results	Server should return a success. Then, the session will end and the cache will be cleared.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS004
Test Type/Category	Authorization
Title	Create User
Procedure of testing steps	1. Have an unauthorized session. 2. Manually input a non-picked username and a password. 3. Send a request to the server to create a user with those credentials. 4. See that the response is successful.
Expected results	Server should successfully create a user with the given credentials and return a success message.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS005
Test Type/Category	Submissions
Title	File Type Restrictions
Procedure of testing steps	1. Have an authorized session. 2. Locally select a file that is not an executable. 3. See that the local client does not let it happen.
Expected results	The local client will not let the user select a non-executable file to upload for malware checking.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS006
Test Type/Category	Submissions
Title	File Size Restrictions
Procedure of testing steps	1. Have an authorized session. 2. Locally select a file that is not at a desirable size. 3. See that the local client does not let it happen.
Expected results	The local client will not let the user select a file too big to upload for malware checking.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS007
Test Type/Category	Submissions
Title	Queue Status
Procedure of testing steps	1. Have an authorized session. 2. Successfully upload a few big files for file checking. 3. Upload one last small file successfully for file checking. 4. Request its queue status from the server periodically until it is successful.
Expected results	The uploaded file will stay in queue for a while, which will be observed by the client, then it will be processed and have its result returned successfully.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS008
Test Type/Category	Submissions
Title	Retrieve History
Procedure of testing steps	1. Have an authorized session. 2. Send a request to the server to return the submission history. 3. See that the response is successful.
Expected results	The server will return the history data and return a successful message.
Priority/Severity	Minor
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS009
Test Type/Category	Submissions
Title	File Report
Procedure of testing steps	1. Have an authorized session. 2. Select a present submission id. 3. Request its report. 4. See that the server successfully and correctly returned its scan results.
Expected results	The server will successfully return the file submitted via the corresponding id's scan results.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	CLS010
Test Type/Category	Platform
Title	Platform Behavior Check
Procedure of testing steps	1. Two boolean checks to see whether the current client is on web or on desktop: `kIsWeb` and `defaultTargetPlatform == TargetPlatform.windows`
Expected results	The debug lines on boolean checks will show the result, showing whether the user is on a web or a desktop client.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

5.2 Server Side Test Cases

Test ID	SRV001
Test Type/Category	Integration
Title	Main Server Availability
Procedure of testing steps	1. Shutdown the Ubuntu server 2. Send a request to the main server that requires Ubuntu server 3. Verify that the main server responds successfully with an appropriate error message
Expected results	The main server should respond successfully and return an appropriate error message when the Ubuntu server is down.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV002
Test Type/Category	Integration
Title	Database Operations Race Condition
Procedure of testing steps	1. Trigger multiple requests to the server to the same resource simultaneously 2. Verify that the server returns accurate data without any conflict or errors
Expected results	The server should handle simultaneous requests without any race condition errors and return accurate data accordingly.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV003
Test Type/Category	Integration
Title	Main Server Update
Procedure of testing steps	1. Start an execution in the Ubuntu server 2. Monitor the main server for the update of the results 3. Verify that the main server updates results when the execution in the Ubuntu server is done
Expected results	The main server should update the results correctly when the execution in the Ubuntu server is done.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV004
Test Type/Category	Integration
Title	Valid Result Availability
Procedure of testing steps	1. Check if there is a valid result already exists 2. Send a request to the server with the valid data 3. Verify that the server responds immediately with the existing result
Expected results	The server should respond immediately if a valid result already exists and return the data accordingly.
Priority/Severity	Minor
Date Tested and Test Result	19.05.2023 and Failed. The server took time to respond.

Test ID	SRV005
Test Type/Category	Integration
Title	Model Update
Procedure of testing steps	1. Check if there is a valid result already exists 2. Send a request to the server to update the machine learning model 3. Verify that the server updates the valid boolean fields to false
Expected results	The server should update the valid boolean fields accordingly when the model is updated.
Priority/Severity	Minor
Date Tested and Test Result	19.05.2023 and Failed. Server did not automatically update the valid boolean fields accordingly.

Test ID	SRV006
Test Type/Category	Integration
Title	File Size Limit
Procedure of testing steps	1. Send a file to the server that exceeds the maximum file size limit 2. Verify that the server rejects the file and returns an appropriate error message
Expected results	The server should reject files that exceed the maximum file size limit and return an appropriate error message.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV007
Test Type/Category	Integration
Title	Executable File
Procedure of testing steps	1. Send a file to the server that is not a portable executable 2. Verify that the server rejects the file and returns an appropriate error message
Expected results	The server should only accept executable files without any issue.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV008
Test Type/Category	Integration
Title	Required Fields
Procedure of testing steps	1. Send a submission request to the server with some required fields missing 2. Verify that the server rejects the request and returns an appropriate error message
Expected results	The server should reject submission requests without the required fields and return an appropriate error message.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV009
Test Type/Category	Security
Title	User Access Control
Procedure of testing steps	1. Login as User A and attempt to access the submissions of User B 2. Verify that the server denies access and returns an appropriate error message
Expected results	The server should not allow users to access other users' submissions and return an appropriate error message.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV010
Test Type/Category	Integration
Title	Submission Listing
Procedure of testing steps	1. Login as User A and send multiple submissions to the server 2. Request the list of submissions for User A from the server 3. Verify that the server lists all the submissions of User A correctly
Expected results	The server should correctly list all the submissions of a user when requested.
Priority/Severity	Minor
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV011
Test Type/Category	Integration
Title	User Creation
Procedure of testing steps	1. Send a request to create a new user with valid fields 2. Verify that the server creates the user and returns an appropriate success message
Expected results	The server should create a new user with valid fields and return an appropriate success message.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV012
Test Type/Category	Integration
Title	Session Termination
Procedure of testing steps	1. Log in as User A 2. Log out as User A 3. Attempt to access any protected resource as User A 4. Verify that the server denies access and returns an appropriate error message
Expected results	The server should end the session of a user when they log out and deny access to any protected resource when not authenticated.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV013
Test Type/Category	Security
Title	User Access Control
Procedure of testing steps	1. Login as User A and create multiple submissions 2. Login as User B and attempt to access the submissions of User A 3. Verify that the server denies access and returns an appropriate error message 4. Login as User A and request the list of their own submissions 5. Verify that the server lists all the submissions of User A correctly
Expected results	The server should only allow users to access their own submissions and deny access to any other submissions.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV014
Test Type/Category	Security
Title	Non-authenticated Access
Procedure of testing steps	1. Attempt to access any protected resource without authentication 2. Verify that the server denies access and returns an appropriate error message
Expected results	The server should deny access to any protected resource without authentication and return an appropriate error message.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV015
Test Type/Category	Integration
Title	Dynamic Analysis
Procedure of testing steps	1. Send a request to perform dynamic analysis 2. Verify that the server performs dynamic analysis successfully and returns an appropriate success message
Expected results	The server should perform dynamic analysis successfully and return an appropriate success message.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV016
Test Type/Category	Integration
Title	Server-to-Server Communication
Procedure of testing steps	1. Send a request from outside of the main server to the Ubuntu server 2. Verify that the Ubuntu server only listens to requests from the main server
Expected results	The Ubuntu server should only listen to requests from the main server.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV017
Test Type/Category	Functional
Title	Test Cases for Duplicate Username Rejection
Procedure of testing steps	1. Register a new user with a username that has already been used.
Expected results	The system should reject the registration attempt and display an error message indicating that the username has already been taken.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	SRV018
Test Type/Category	Functional
Title	Test Cases for Cuckoo Server Recovery
Procedure of testing steps	1. Simulate a Cuckoo server failure. 2. Check if the Ubuntu server can recover from the failure and start running cuckoo.
Expected results	The Ubuntu server should be able to recover from the failure and start running cuckoo.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Failed. Ubuntu server could not recover without a re-run.

Test ID	SRV019
Test Type/Category	Functional
Title	Test Cases for File Submission Rejection
Procedure of testing steps	1. Submit a file without selecting any file.
Expected results	The system should reject the file submission and display an error message indicating that no file was selected.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

5.3 Test Cases for Algorithms for Machine Learning

Test ID	ML001
Test Type/Category	Functional
Title	Test Cases for Successful Size of Optional Header Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the size of optional header from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML002
Test Type/Category	Functional
Title	Test Cases for Successful Major Linker Version Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the major linker version from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML003
Test Type/Category	Functional
Title	Test Cases for Successful Minor Linker Version Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the minor linker version from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML004
Test Type/Category	Functional
Title	Test Cases for Successful Size of Code Header Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the size of code from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML005
Test Type/Category	Functional
Title	Test Cases for Successful Size of Initialized Data Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the size of initialized data from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML006
Test Type/Category	Functional
Title	Test Cases for Successful Size of Uninitialized Data Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the size of uninitialized data from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML007
Test Type/Category	Functional
Title	Test Cases for Successful Address of Entry Point Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the address of entry point from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML008
Test Type/Category	Functional
Title	Test Cases for Successful Base of Code Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the base of code from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML009
Test Type/Category	Functional
Title	Test Cases for Successful Base of Data Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the base of data from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML010
Test Type/Category	Functional
Title	Test Cases for Successful Image Base Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the image base from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML011
Test Type/Category	Functional
Title	Test Cases for Successful Section Alignment Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the section alignment from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML012
Test Type/Category	Functional
Title	Test Cases for Successful File Alignment Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the file alignment from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML013
Test Type/Category	Functional
Title	Test Cases for Successful Major Operating System Version
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the major operating system version from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML014
Test Type/Category	Functional
Title	Test Cases for Successful Minor Operating System Version Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the minor operating system version from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML015
Test Type/Category	Functional
Title	Test Cases for Successful Byte to Pixel Conversion
Procedure of testing steps	1. Input a byte data to the conversion algorithm.
Expected results	The algorithm should convert the byte data to pixel data successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML016
Test Type/Category	Functional
Title	Test Cases for Successful Image Resizing
Procedure of testing steps	1. Input a grayscale image to the resizing algorithm.
Expected results	The algorithm should resize the image to the specified dimensions successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML017
Test Type/Category	Functional
Title	Test Cases for Successful Output of First Static Model
Procedure of testing steps	1. Input a dataset to the first static model.
Expected results	The first static model should give softmax output successfully.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML018
Test Type/Category	Functional
Title	Test Cases for Successful Output of Second Static Model
Procedure of testing steps	1. Input a dataset to the second static model.
Expected results	The second static model should give softmax output successfully.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML019
Test Type/Category	Functional
Title	Test Cases for Successful Output of Dynamic Model
Procedure of testing steps	1. Input a dataset to the dynamic model.
Expected results	The dynamic model should give output successfully.
Priority/Severity	Major
Date Tested and Test Result	19.05.2023 and Passed.

Test ID	ML020
Test Type/Category	Functional
Title	Test Cases for Successful Size of Headers Extraction
Procedure of testing steps	1. Input a dataset to the feature extraction algorithm.
Expected results	The algorithm should extract the size of headers from the executable successfully.
Priority/Severity	Critical
Date Tested and Test Result	19.05.2023 and Passed.

6. Maintenance Plan and Details

To make our app maintainable, we will do the following checkups;

6.1 Regular Updates:

We will release updates on a regular basis to keep the frameworks, libraries, and dependencies up to date. We will use the most recent versions of Flutter, Django, FastAPI, and other related frameworks in the project. In the event of a breaking change in the dependencies, the codebase will be modified according to those changes.

6.2 Bug Fixes:

We will continuously monitor the reported bugs and issues to address them immediately. Reported issues will be prioritized and categorized according to their impact on the application, and those bugs will be solved with regular bug fixes and patches.

6.3 Performance Optimization:

In the case of the identification of any performance issue, we will optimize the database queries, API calls, and network communications to solve the bottlenecks in the application and increase the performance.

6.4 Security Updates:

To decrease the security risks, all the software components will be kept up to date. We will regularly perform vulnerability tests and security audits to make sure that the security of the app meets sectoral standards.

6.5 Database Maintenance:

Regular SQLite database backups will be performed to prevent data loss. To increase the performance of the database, it will be monitored regularly, and in the case of a performance issue, optimization of query execution, indexing will be performed. If needed, table structures will be updated.

6.6 Disaster Recovery and Backup Strategy:

We will make sure that the critical data is backed up to be used in the case of any server failures or other catastrophic events.

6.7 Continuous Testing and Quality Assurance:

We will implement some automated testing frameworks such as unit testing, end-to-end tests etc. Also, to identify potential issues in the code and improve code quality, we will use a static code analysis tool named *sonar* [7].

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

7.1.1 Public Safety

Every year, ransomware alone costs the public $20 billion. 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.

7.1.2 Public Welfare

Governments or agencies are the backbone of welfare since they provide items and services to the public. These entities employ a lot of personnel who are doing civil servant jobs and who generally sit at a desk and work on computers from 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 too 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 in these situations by providing 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.

7.1.3 Global Factors

Since CleaverWall is open source, its success means that people will dissect and analyze the project. Although it is not currently popular, a successful project that uses machine learning could spark 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.

7.1.4 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 a 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".

7.1.5 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.

Table 1: Factors that can affect analysis and design.

Factor	Effect level	Effect
Public health	0	The project has nothing to do with health.
Public safety	4	The static responses of the project are expected to protect its users from obvious scams by 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 in 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 programming culture.
Social factors	0	The project is unlikely to impact the current era of civilization.

7.2. Ethics and Professional Responsibilities

- Every academic paper and third-party organization that aided us was cited.
- We request the user's permission before adding the executable's features to the dataset..
- We respect and protect our customers' privacy and data.

7.3. Teamwork Details

7.3.1. Contributing and functioning effectively on the team

Each member of our team is expected to actively contribute and function effectively within their respective work packages and within the team as a whole. We recognize that effective teamwork necessitates clear communication, active participation, and a dedication to meeting deadlines and carrying out responsibilities.

We will establish communication channels to keep each other informed of project developments and provide regular updates to ensure that we are all on the same page. We will also be available to provide support and assistance to our teammates as needed.

In addition to our individual contributions, we recognize the value of working together to achieve our common goals. We will welcome feedback, suggestions, and ideas from all team members, regardless of role or seniority. We believe that by valuing and incorporating each other's points of view, we can gain a more comprehensive understanding of the project and achieve better results.

7.3.2. Helping creating a collaborative and inclusive environment

We believe that fostering a collaborative and inclusive environment is critical to the development of a successful team. We will collaborate to create a safe and welcoming environment in which everyone feels heard, respected, and valued.

We will actively listen to one another, provide constructive feedback, and encourage open and honest communication to accomplish this. We recognize that different points of view and experiences can help us identify opportunities and potential solutions that we might not have considered otherwise. As a result, we will actively seek out and incorporate ideas and suggestions from all team members.

We are committed to maintaining a sense of accountability and responsibility for our actions, in addition to creating a supportive team culture. We will hold ourselves and each other accountable for adhering to the principles of inclusivity and respect, and we will take appropriate action if these principles are violated.

7.3.3. Taking lead role and sharing leadership on the team

We recognize that effective leadership is essential to the success of any team. While leaders for each work package have been identified, we recognize that leadership is a shared responsibility that extends beyond formal roles. We believe that everyone has valuable skills and expertise to offer, and we encourage everyone on the team to take on leadership roles and share their knowledge.

We will be responsible as leaders for ensuring that our team members meet their deadlines and fulfill their responsibilities. We will also be available to provide advice and support as needed. We recognize, however, that our success as a team is dependent on our ability to collaborate and take collective ownership of our project. Therefore, we will actively seek out and incorporate input from all members of the team, and encourage everyone to take an active role in shaping our project's direction and success.

In conclusion, we believe that effective teamwork necessitates clear communication, active participation, and a dedication to creating a collaborative and inclusive environment. We think we can accomplish our common objectives and create a successful project by adhering to these values and cooperating.

7.3.4. Meeting objectives

The initial project objectives and their current status are shown in the table below;

Table 2: Project Objectives and Status

Objective #	Description	Status
Analysis	Discussion and determination of project-related specifications and requirements.	Finished
Design	Evaluating the Analysis and recommending a detailed system design	Finished
Malware Classifier Development 1	Creating a malware classifier using features from .asm files.	Cancelled
Malware Classifier Development 2	Creating a malware classifier using grayscale image of an executable.	Finished
Malware Classifier Development 3	Creating a multimodal classifier combining 2 different static classifiers.	Finished
Malware Classifier Development 4	Creating a malware classifier using Windows API sequence.	Finished
Server Side Development 1	Providing an API that enables client side applications to perform malware analysis remotely.	Finished
Server Side Development 2	Utilizing a virtual environment to perform dynamic analysis on suspected PE files.	Finished
Server Side Development 3	Storing the malware data on user's permission for potential use.	In progress
Client Side Development 1	Providing a clear and efficient UI and UX for the users for web interface.	Finished
Client Side Development 2	Providing a clear and efficient UI and UX for the users for the desktop interface.	In progress
Client Side Development 3	Linking the necessary tools for enabling MVP features such as static analysis and quarantining malicious files to the desktop application.	In progress

Objective #	Description	Status
Testing	Testing the implemented parts of the project	Finished

7.4 New Knowledge Acquired and Applied

- Setting up a Flutter desktop app.
- Using AutoRoute.
- Using Dio.
- Extracting tokens from HTTP.
- Picking files on a client.
- Utilizing better scalability practices.
- Using Django Rest Framework in general.
- Creating a FastAPI server.
- Utilizing different database engines such as sqlite3 and django database which is also implemented over sqlite3.
- Learned that different frameworks' approaches to asynchronous programming can differ vastly. Example: Some may even not support (Django rest framework) a thread created by an api call creating another thread, responding and terminating before the sub-thread.
- Restricting APIs' availability to outside: IP check versus using api keys (we chose to require api keys)
- Miscellaneous things about file structures etc: disassemblers, hashing files and uniques of those hashes (md5 versus sha256 very).
- Running ML models on files efficiently (what not to write to disc, when to load sth to memory, how python handles that by default)
- What to do and not to do in project organization with a lot of sub projects: readme and requirements file place organizations, using a git ignored keys.json.
- Carefully defining package versions in requirements
- Setting up the Cuckoo environment and testing malware in a Virtual Machine
- Developing in an Ubuntu environment
- Static malware analysis
- Dynamic malware analysis
- Utilizing Class Weights for training an imbalanced dataset.

8. Conclusion and Future Work

CleaverWall is an open-source malware detection and classification tool for malicious portable executable files. It employs static and dynamic analysis of the executable files. The two methods of its static analysis are analyzing PE header features of executables and converting executables into grayscale images to use Convolutional Neural Networks.

CleaverWall does its dynamic analysis by running the executable on a sandbox in order to analyze its Windows API call.

One of our key features is being flexible as we offer many analysis options for our users. We hope this flexibility becomes a signature of our project, leaving behind all the quick-scans and thorough-scans; and providing people with whatever that suits their current needs.

As stated before, CleaverWall is an open source project, and it aims to stay so. Our goal is to surpass other open source anti-malwares, given that they perform poorly. However, the real challenge begins with the big opponents. Although it is harsh, with considerable time and sizable users we believe we can match their huge datasets.

9. Glossary

Malware: Harmful software aiming to cause damage to computer systems.
Sandbox: Testing environment on which potentially harmful softwares can be run safely.

10. References

[1] 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.

[2] 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: 19-May-2023].

[3] 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.

[4] “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: 19-May-2023].

[5] 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.

[6] “Automated malware analysis,” *Cuckoo Sandbox - Automated Malware Analysis*. [Online]. Available: https://cuckoosandbox.org/. [Accessed: 19-May-2023].

[7] “What is clean code, all code fit for development and production.” *Sonar* [Online]. Available: https://www.sonarsource.com/solutions/clean-code/. Accessed: 19-May-2023