The project is titled “Exploration of Open-Source Anti-Malware Solutions with Machine-Learning Capabilities on Windows Platform”.

The objective of the project is to perform in-depth research into open-source anti-malware solutions that are equipped with machine-learning capabilities, for the Windows Operating System (OS). Hence, the scope of the project revolved around anti-malware, machine-learning and Windows OS.

Based on research done into open-source anti-malware, I developed anti-malware and security solutions for Windows machines.Besides developing anti-virus functionalities, the project also included collecting training data and building machine-learning models for the detection of malicious content such as PEs, PDFs, and Office Documents.

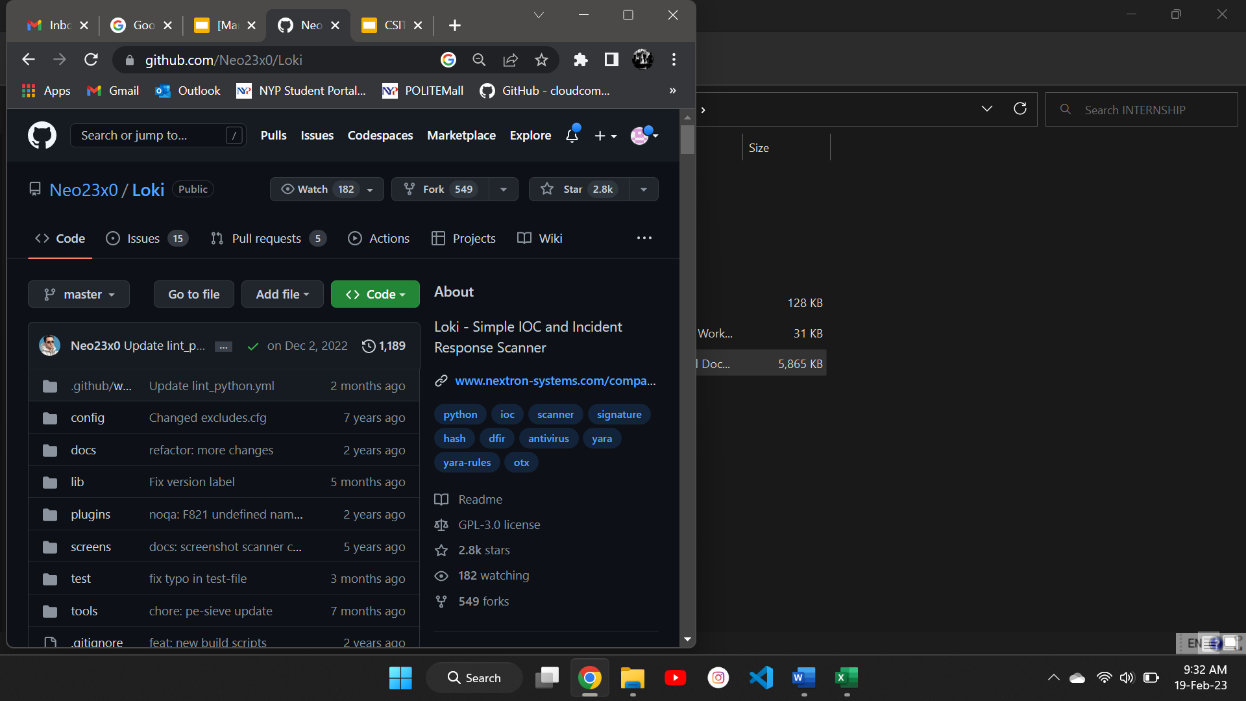
**Task 1: Research on Open-Source Anti-Virus Tools equipped with Machine Learning Capabilities for Windows Operating System (OS)**

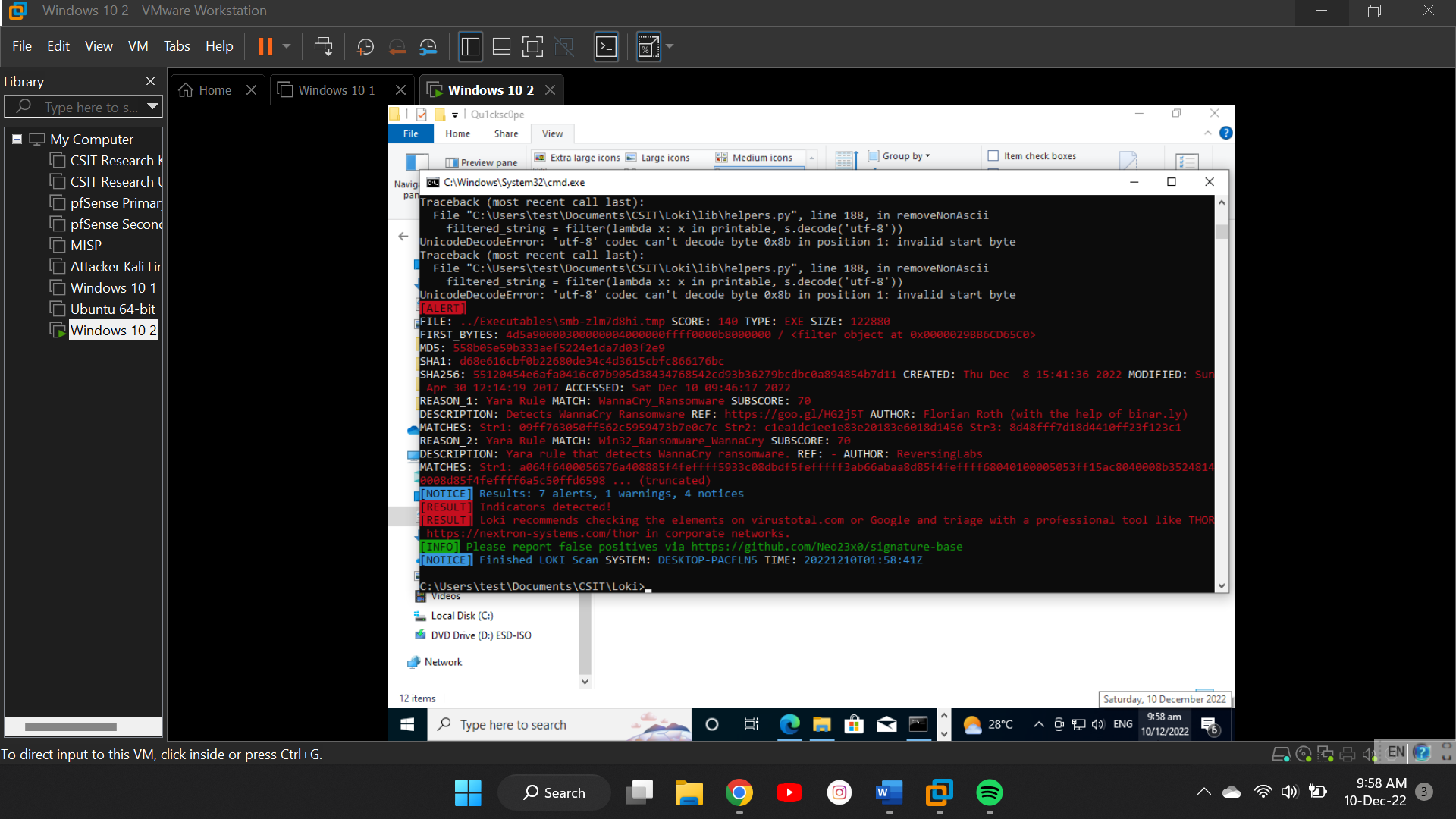
Description

* I started by researching on the various machine-learning terminologies, algorithms, features, and performance to understand how the different algorithms are applied in real-world context.
* I also tried out a few online machine-learning tutorials, where I got to train, test and validate simple machine-learning models (e.g. Decision Tree and Random Forest) using libraries such as TensorFlow and scikit-learn.
* Afterwards, I started the research on existing open-source implementations.
* For each identified tool, I documented its functionalities, usage, strengths, and weaknesses. I also set up the tools within a Windows Virtual Machine (VM) to get a better feel of the tool’s usability, responsiveness, performance, etc.
* After trying out the anti-virus tools, I evaluated that Loki (<https://github.com/Neo23x0/Loki>), was the most suitable for further developments. Its detection was based on indicators of compromise (IOCs) such as hashes and YARA rules:
* As for the machine-learning tools, most of my findings were related to prediction of Portable Executable (PE) malware, such as executables (EXEs) and dynamic-link libraries (DLLs). I proposed to work on an existing tool called MDML (<https://github.com/mohamedbenchikh/MDML>). It came along with a training dataset of ~18000 legitimate and malicious PEs.

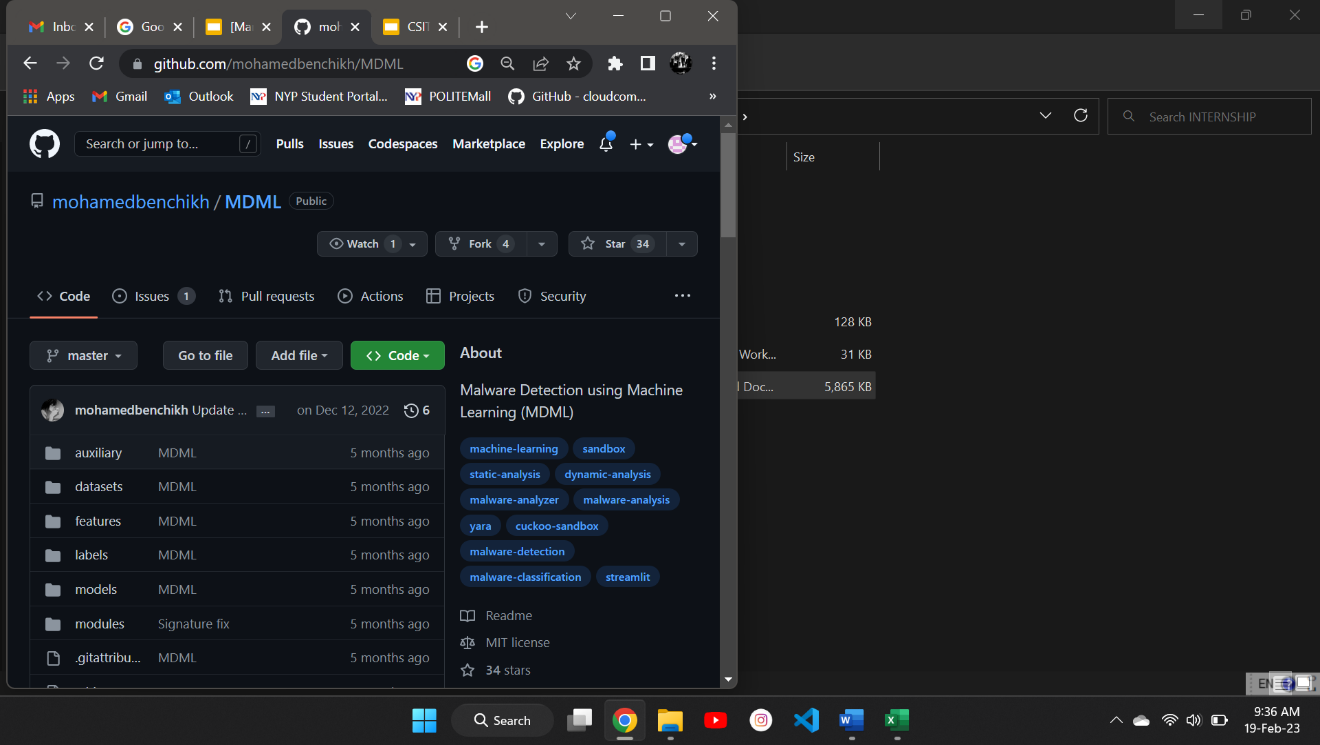
Deliverables

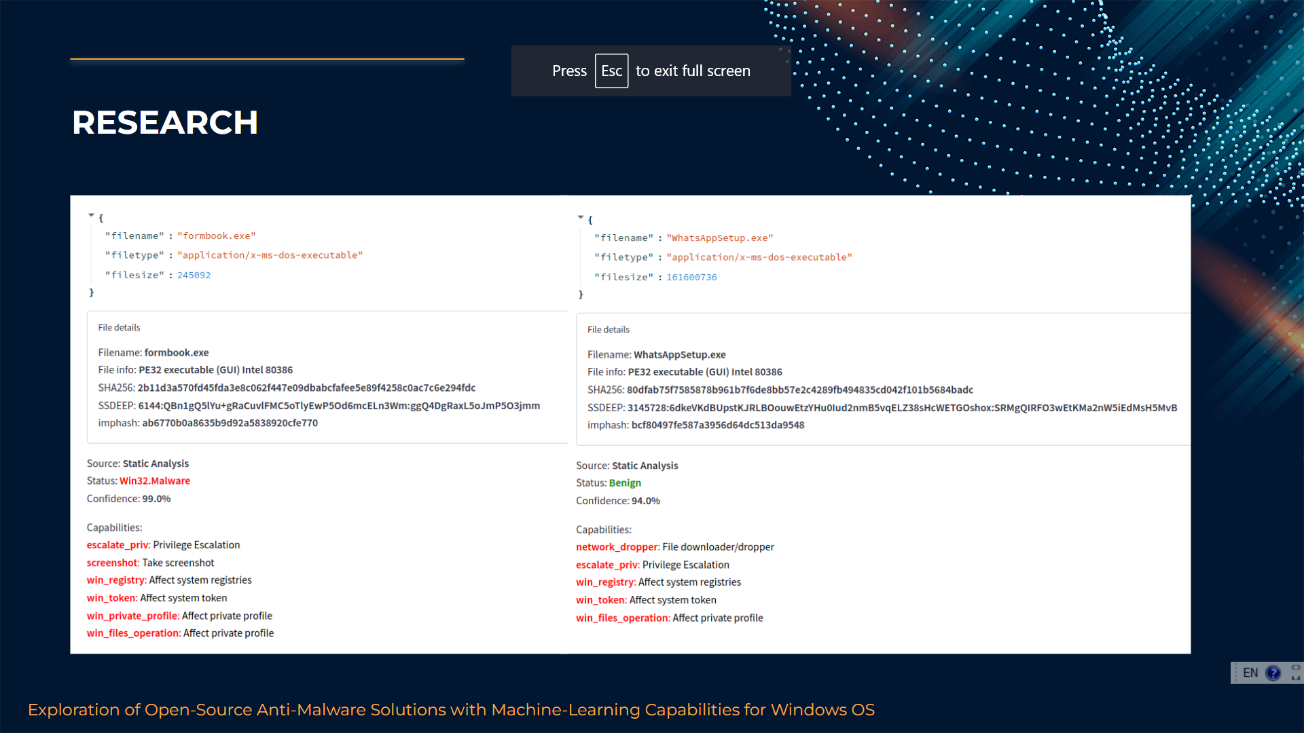
* Evaluate and select suitable anti-malware tools.
* I decided to work with Loki and MDML (there could be an integration done).
* Loki:





* MDML:





Problems Faced

* While performing the research, I faced an issue – there were no full-fledged anti-viruses equipped with machine-learning capabilities.
* I split the research into 2 parts: the first part focuses on selecting a suitable anti-virus tool, and the second part focuses on selecting useful machine-learning algorithms for malware detection.
* Based on these 2 focus areas, there could be a possible integration done later as part of the project. Hence, I continued my research in that direction and eventually managed to find useful tools belonging to each category.

Experience Gained / Lessons Learnt

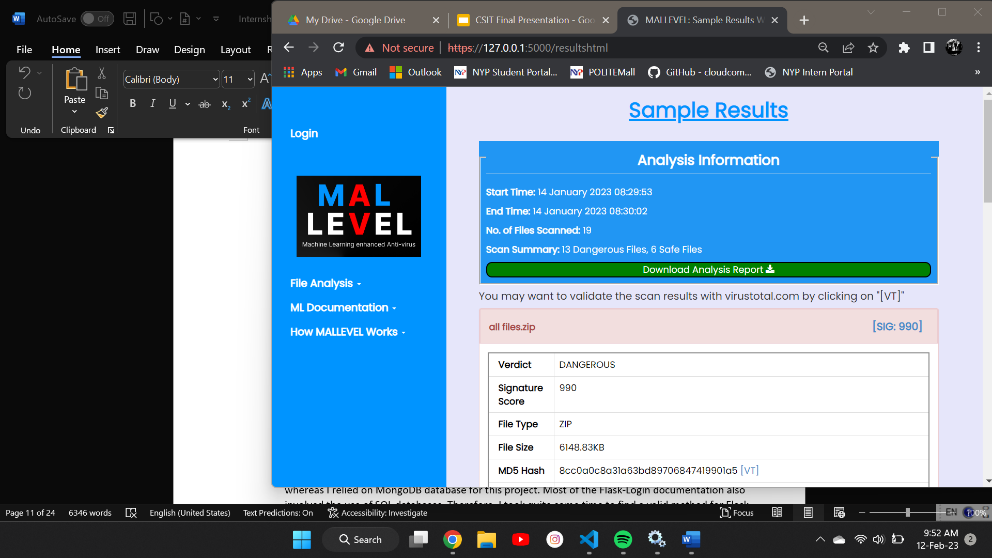
* Before working on this project, I did not have much exposure to machine-learning. However, performing this research has allowed me to pick up useful knowledge about how different machine-learning algorithms can be utilized for detection of malware.
* I have also learnt about some guidelines (e.g. suitable ratio for splitting the training and testing datasets, implementing cross-validation to obtain optimal parameters, etc.). Hence, this task had provided me with deeper insights into machine-learning.

**Task 2: Enhancement of Chosen Open-Source Anti-Virus (Loki)**

Description

* Before my enhancements, Loki was a command-line interface (CLI) tool, where all the logs and scan results were printed in the CLI, but not saved. It was also a standalone tool which only scans files on the system it was installed on.
* To improve Loki’s overall functionality, I did a thorough code review to understand the different modules and functions, such that I could make the appropriate improvements later on.
* Firstly, I utilized the Python Flask framework to develop the anti-virus server, and designed a user-friendly web interface for invoking Loki’s core functionalities, such as updating of malware signatures and scanning of files.

Graphical user interface, application

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

* This involved the creation of HTML templates and CSS files.
* I also generated data visualizations in HTML using Chart.js.
* Secondly, I worked on creating logs and file analysis reports for the anti-virus server.
* There were 3 main types of logs:

1. Initialization logs – contain information about server startup.
2. Analysis logs – contain information about file scanning process.
3. Update logs – contain information about signature updates.

* At the end of the file analysis, all the results will be saved to a JSON report which contains information on the statistics of the scan, and individual file information. A sample report is shown below:

A screenshot of a computer

Description automatically generated with medium confidence

* Thirdly, I worked on adding support for scanning of compressed archives.
* I started by doing extensive research on the various archives available. Areas of consideration included the proper handling of different file formats and the possibility of nested archives.
* Based on these considerations, I found a useful Python module called pyunpack. Hence, I created scripts that utilized the pyunpack module to extract files from different archive formats and dealing with nested archives using recursion.
* After implementing this feature, Loki was able to automatically extract the submitted archives, scan individual files within them, and alert if malware was found. However, a limitation of this implementation is that it is currently unable to deal with password-protected archives.
* The fourth subtask was to connect the anti-virus server to a database, to store data such as file analysis reports.
* Before setting up and working with a database, I did research and evaluation of whether an SQL database or MongoDB database would be more suitable for the final product. Eventually, I settled on using MongoDB because it is faster, more scalable, and more flexible as compared to SQL.
* MongoDB also stores its data in Binary JSON (BSON) format, which corresponds to the JSON analysis reports produced at the end of every file scan. Hence, I installed MongoDB and performed the basic configurations to set it up successfully.
* Afterwards, I also installed MongoDB Compass which was a GUI for interacting with the MongoDB database (i.e. preview data, modify data values). Below is a screenshot of some stored data, viewed using MongoDB Compass:

Graphical user interface, text, application

Description automatically generated

* Once MongoDB and MongoDB Compass were set up, I searched for Python scripts to perform Create, Read, Update and Delete (CRUD) operations on the MongoDB database and found a module called pymongo. Therefore, I relied on the module for functions like saving file scan reports, adding users, and verifying user credentials.
* My fifth subtask was to secure the web UI.
* I used the Flask-Login library for implementing authentication in the web UI, and the MongoDB database for storing user credentials.
* Subsequently, I tried to find a proper method for Flask authentication by fetching data from a MongoDB database.
* After writing the codes and troubleshooting some runtime errors, I managed to get the authentication mechanism completed – regular users can upload files for scanning via the web UI, but they cannot access privileged functions which requires authentication by administrators.
* The final subtask was to improve the coverage of signature-based detection. Despite weekly signature updates, Loki’s signature database (of around 50000 hashes and YARA rules) was relatively small compared to commercial anti-virus solutions.
* I proposed to leverage open-source threat intelligence platforms and malware repositories for importing more Indicators Of Compromise (IOCs) such as hashes, Code Signing Certificate Blocklists (CSCBs), and malicious URLs. I relied on 3 main platforms for this enhancement: MalwareBazaar, AlienVault and PhishTank.
* MalwareBazaar provides a text file containing SHA256 hashes of malware samples and a CSCB CSV file, which contains information about certificates used by threat actors for signing malware. Hence, I created scripts to fetch and process the files.
* To leverage on the CSCB, I created scripts to parse portable executables (PEs) using the Python lief module and obtain the file’s code signing certificate information; if the serial number of the certificate existed in the CSCB, the file would be marked as malicious.
* AlienVault is a threat intelligence platform which offers daily updates of new IOCs and threat events. I signed up for a free account and obtained an API key, which allows access to information within the platform.
* I subsequently searched for the API documentation and implemented Python scripts for querying AlienVault to obtain the latest IOCs, such as SHA256 hashes and malicious domains. The reason for importing malicious domains is to continually update the URL blacklist used for the machine-learning framework to predict malicious websites.
* PhishTank is a platform that offers data on the latest phishing links. I created a simple script to query the endpoint for obtaining the phishing data in JSON format. These links are regularly updated within the URL blacklist too.

Deliverables

* Convert the chosen anti-virus (Loki) from a standalone tool to a server.
* Allows centralized scanning of files from multiple endpoints, which is done in later parts of the project.
* The server was built using the Python Flask framework.
* Develop a web interface for invoking Loki’s functionalities.
* Design considerations included user-friendliness and responsiveness.
* Generate process logs and file analysis reports.
* Add support to Loki for scanning compressed archives.
* Users would not have to extract archives in order to submit the contained files for scanning – they can submit the archive directly.
* Currently only supports scanning of .zip, .7z, .gz and .bz2 files.
* Connect the anti-virus server to a database.
* Allows organized storage of data such as file analysis reports.
* MongoDB was chosen as the because of its flexibility and scalability.
* Implement authentication and administrative controls for web interface.
* The Flask-Login library was used to develop the authentication mechanism.
* While regular users can only access a limited set of functions, administrators are able to access privileged functions, such as viewing dashboards, updating signatures and whitelisting certain files.
* Improve coverage of signature-based detection.
* Initially, Loki only had around 50,000 IOCs. After bringing in other IOCs from MalwareBazaar, AlienVault and PhishTank, the anti-virus server now has around 710,000 IOCs, improving the malware detection coverage greatly.

Experience Gained / Lessons Learnt

* While working on this task, I felt that I have refined my skills in Python and gained exposure to many useful modules such as pyunpack, pymongo, etc.
* I also got to experience working with technology like MongoDB and threat intelligence platforms like AlienVault which I have not used in my previous works. In the process, I better understood the differences between MongoDB and SQL databases.
* Lots of independent learning was involved as I had to plan out the enhancements, perform the necessary research on how to implement them, and discover the methods to troubleshoot runtime and logic errors in the development process.

**Task 3: Heuristics-Based Detection of Malicious Portable Executables (PEs)**

Description

* I performed research to find out what heuristics were and how it could be used to detect malicious files. I looked up websites by commercial anti-virus providers such as Kaspersky and Norton, and found that there were 2 main forms of heuristics – static and dynamic.
* Considering the runtime overhead of dynamic heuristic analysis, it would not be feasible to include it in the anti-virus solution. However, static heuristic analysis is much more efficient and it could play an important role in detecting polymorphic malware which have bypassed signature-based detection.
* I subsequently focused on searching for GitHub projects that involved static heuristic analysis, to understand how heuristics could be implemented in codes. I also referenced malware analysis articles to obtain a better understanding of the PE file format and how certain properties could indicate that it is a malware.
* Afterwards, I developed a script for heuristics-based detection. Some of the detection rules included checking whether the PE uses anti-debugging techniques, and whether the PE has sections with high entropy:

A screenshot of a computer

Description automatically generated

* I also integrated the heuristics detection system within the anti-virus server. If a PE managed to bypass signature-based detection (meaning it did not match any signature), then it would be sent to the heuristics detection system for scanning.
* I tested the system using 100 sample PE malware provided by my IS. The detection rate was fairly low as it only detected 27 of them to be dangerous.

Deliverables

* Develop heuristics-based detection system for malicious PEs.
* Includes detection rules such as sections with high entropy, anti-debugging techniques, anti-VM techniques, suspicious entry points, and suspicious resources.
* Evaluate feasibility of heuristics-based detection system.
* Although it is currently implemented within the anti-virus server, I evaluated that it was not really feasible due to the low detection rate of 27% on the samples found.
* There could be improvements made to the tuning and implementation of rules to reduce the number of false positives and false negatives, hence increasing the overall accuracy of the system.

Experience Gained / Lessons Learnt

* While working on this task, I have read many articles that involved decompiling a PE and looking for suspicious properties. This helped to deepen my understanding towards the PE file format as I got to pick up new concepts like entropy, entry points and resources. Hence, this task allowed me to get a feel of performing malware analysis.

**Task 4: Machine-Learning Framework for Prediction of Malicious Portable Executables (PEs)**

Description

* To perform prediction of PE malware, I chose to work with the GitHub repository called MDML, which I found during the research phase of the project. It came along with a pre-trained XGBoost model and a training dataset of ~18000 legitimate and malicious PEs
* The legitimate PEs were sourced from online software hosting websites such as SourceForge and CNET, while malicious PEs were mainly sourced from the malware repository MalwareBazaar.
* I trained and tested a mix of algorithms on the dataset, to evaluate which one performed the best and use it for the prediction of PE malware.
* I chose to work with 4 different machine-learning algorithms – Decision Tree, Random Forest, Gradient Boosting and XGBoost. This was because they generally performed well in many scenarios and could be used for classification problems in machine-learning – in my case, it involves binary classification of the PE (legitimate or malicious).
* Below are sample codes that I have developed for training and testing:

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

* As seen from the above codes, I used cross validation to experiment with different parameters for each algorithm (e.g. for Random Forest, train and test it with 50 to 150 estimators/trees).
* The training and testing were carried out on Google Colaboratory (a Jupyter Notebook environment in the cloud) as it provided adequate resources to achieve faster training and testing. I consolidated the training and testing results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Decision Tree | Random Forest | Gradient Boosting | XGBoost |
| Training Accuracy (%) | 97.80 | 98.66 | 98.14 | **98.68** |
| Testing Accuracy (%) | 97.54 | 98.54 | 97.89 | **98.57** |
| False Positive Rate (%) | 2.35 | **1.40** | 2.40 | 1.55 |
| False Negative Rate (%) | 2.58 | 1.52 | 1.76 | **1.29** |

\* The bolded numbers are the best figures for each row.

* As the XGBoost algorithm performed the best, I saved and integrated the trained XGBoost model into the anti-virus server such that PE files could be piped to this model for prediction of whether it is safe or malicious.
* To assess the detection capabilities of the model, I tested the performance of the model on 100 sample PE malware found. It performed relatively well as it managed to predict 78 of them to be dangerous.

Deliverables

* Create machine-learning model for prediction of Portable Executables (PEs).
* The XGBoost model was created and integrated into the existing solution due to its high training and testing accuracy of 98.68% and 98.57% respectively.
* The models are saved as joblib files.
* Evaluate feasibility of the created model.
* It was desirable to keep the model because of its high accuracy.
* The performance testing results were also presented as part of the final presentation, to prove the feasibility of the model.

Problems Faced

* Since I have never done anything related to machine-learning before, I had to learn from online tutorials and documentation in order to create my own models. For example, I went through the machine-learning crash course by Google (<https://developers.google.com/machine-learning/crash-course/ml-intro>) and learnt about the different theoretical concepts, and had a better understanding of how to work with the TensorFlow API.
* Training and testing the models were also slow and resource-intensive on my own laptop. Luckily, I managed to find out about the Google Colaboratory platform which helped to speed up the entire process.

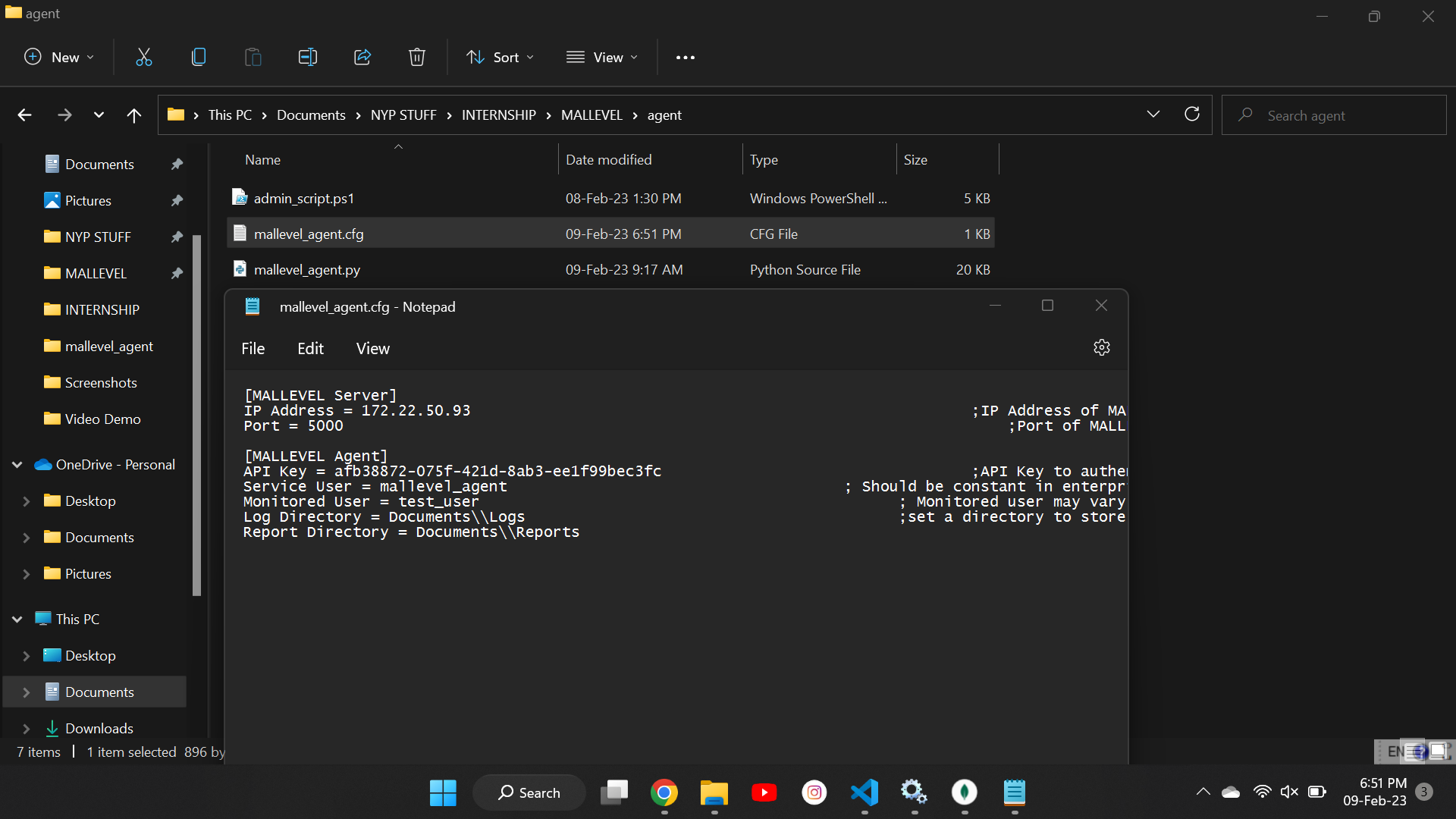
Experience Gained / Lessons Learnt

* It was an eye-opener working on this task as I managed to gain exposure to useful and relevant platforms like Kaggle and Google Colaboratory, which I have never utilized in the past.
* I learnt that Kaggle provided a huge range of machine-learning datasets for various use cases, and it was also a good platform for learning the methods that different machine-learning algorithms could be implemented.
* If I were to work on machine-learning projects in the near future, I would definitely leverage on Google Colaboratory because of its efficiency and support for many machine-learning libraries.
* I also experienced working with relevant Python libraries such as scikit-learn, TensorFlow and pandas in the process of experimenting with different machine-learning algorithms. Reading up on the various documentations has also provided me with insights on how I could tune the different models, carry out oversampling/undersampling, and perform cross-validation.

**Task 5: Designing the Anti-Virus Agent-Server Architecture**

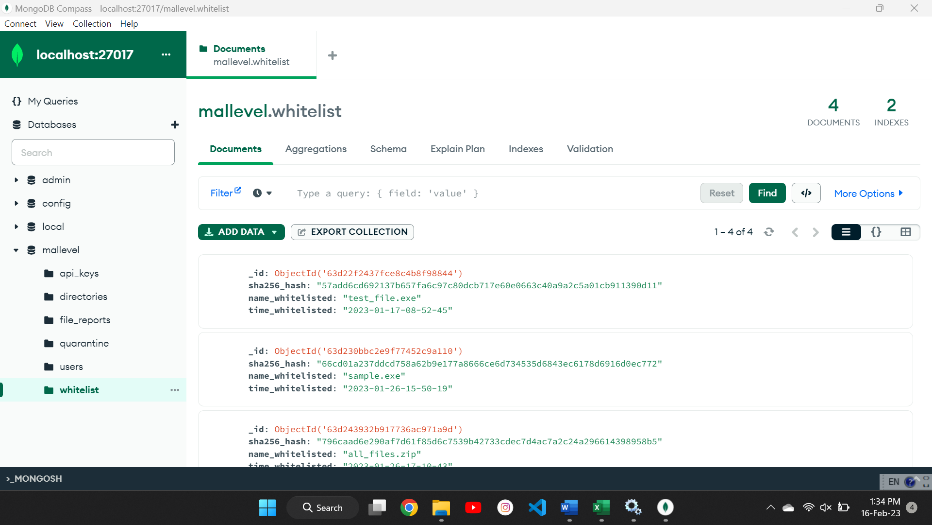
Description

* At this point in time, the integrated anti-virus solution was named “MALLEVEL”.
* The first subtask was to ensure that the files coming into various endpoints will be regularly scanned by the MALLEVEL server.
* To achieve this, I created a Python agent script to run on the endpoints. When the agent first connects to the server, it will fetch predefined configurations (e.g. directory to monitor, monitoring interval, etc.) from the server.
* Once new files are detected within the directory, the agent will send a POST request to upload all the new files to the server for scanning. The server would then scan the files and return the file analysis report to the endpoint.
* The agent will also regularly query the server to check if any updates have been made to the configurations (e.g. monitoring interval increased from 60 seconds to 3600 seconds) and make the necessary changes. This ensures that the agent’s configurations are in sync with the latest configurations.
* Therefore, besides developing the agent script from scratch, I also had to add new functions to the server scripts to respond to agent connections.
* Subsequently, I implemented authentication for the agent-server communication. This involved the use of API keys.
* Unique API keys were generated using the Python uuid module. Whenever an administrator creates a user account, an API key will also be assigned to that account. The API key should be placed in the agent configuration file as it will be read by the agent script:



* Whenever the agent connects to the server, the API key will be sent as part of the request header. The server checks against the MongoDB database and returns a relevant response if the API key is valid. Otherwise, the server terminates the connection and the agent would not be monitored by the server. This helps to ensure that only validated hosts will be regularly monitored by the server for any malicious files.
* The second subtask involved allowing an administrator to whitelist specific files.
* When the administrator submits files for whitelisting, the server will hash each file (using the SHA256 algorithm) and save the hashes to the MongoDB database.
* The agent queries the server for the latest list of whitelisted file hashes; if a whitelisted file appears within a monitored directory, there is no need to send it to the server for scanning, hence reducing the load on the server.
* I created a webpage to support this whitelisting feature, and defined a new data table within the MongoDB database for storing whitelisted file hashes:

A screenshot of a computer

Description automatically generated

* The third subtask was to implement a file quarantine system. The objective of quarantining is to isolate potentially infected files within a computer so that they would not be accidentally executed.
* Firstly, I created a quarantine directory that would store malicious files found within the monitored user’s directory.
* Secondly, I modified the agent script to include the main quarantine function – move malicious files from the monitored user’s directory to the quarantine directory.
* With this quarantine system in place, there is also a need for restoration or permanent deletion of files. Hence, I created webpages for users to view the status of their quarantined files, and for administrators to view status of quarantined files across all endpoints.

|  |  |
| --- | --- |
| **User POV** | **Admin POV** |
|  | A screenshot of a computer  Description automatically generated |

* I subsequently implemented the release and blacklist features, which relies on CRUD operations with the MongoDB database too. Releasing a file means that it would be restored to the original path in the endpoint, while blacklisting a file would permanently remove it from the endpoint.
* The fourth subtask was to send the anti-virus process logs to a Splunk instance for real-time event analysis and visualization. Example of processes include updating of signatures and actions taken during the file scanning.
* As a start, I downloaded a trial version of Splunk Enterprise and performed the basic setup to ensure that it was running smoothly.
* Afterwards, I did some research on the various methods available for the Splunk instance to receive data inputs using Python, and found that there was this feature called HTTP Event Collector (HEC). It allows for sending of logs and reports to the Splunk instance over HTTPS.
* Hence, I configured the necessary HEC settings as shown below:

Graphical user interface, text, application

Description automatically generated

* After configuring the receiver, I proceeded to create a Python script for the MALLEVEL server to send logs of different classification levels (e.g. debug, error, info) to the Splunk instance.They could then be viewed real-time in the Splunk search function.
* I subsequently performed field extractions on the logs sent to Splunk, and created 2 dashboards.
* The first dashboard provided insights into the file scan results. It includes reports on the number of files scanned, number of files that went through machine-learning prediction, the classification of files based on their verdict and file type, as well as information about detected and predicted malicious files:

Chart, bubble chart

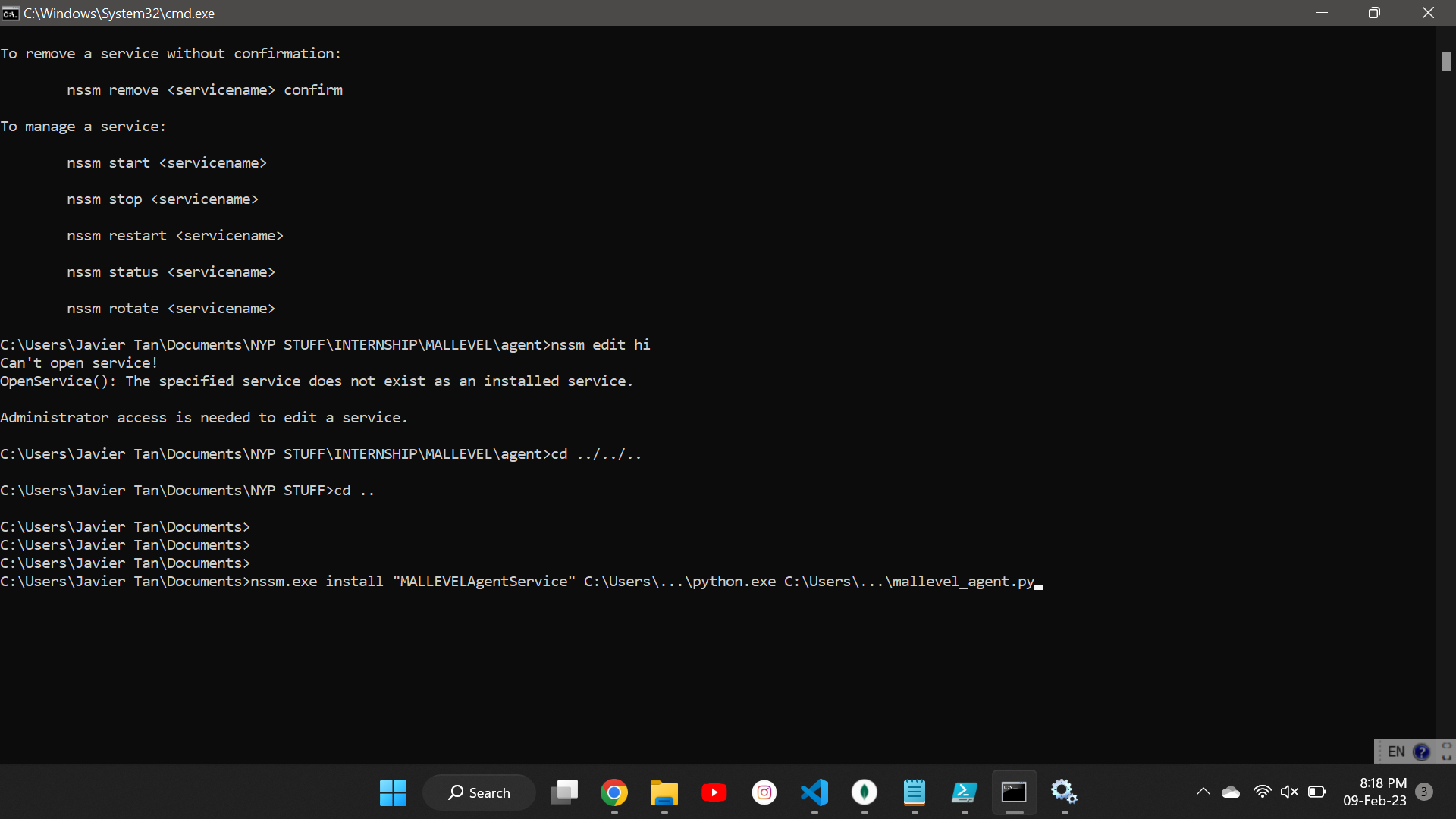
Description automatically generated

* The second dashboard provides a simple display on the operational statistics of the MALLEVEL server. It includes reports on the number of times the server was started, the number of signature database updates, and a general summary of the most recent scan results:

Graphical user interface, table

Description automatically generated

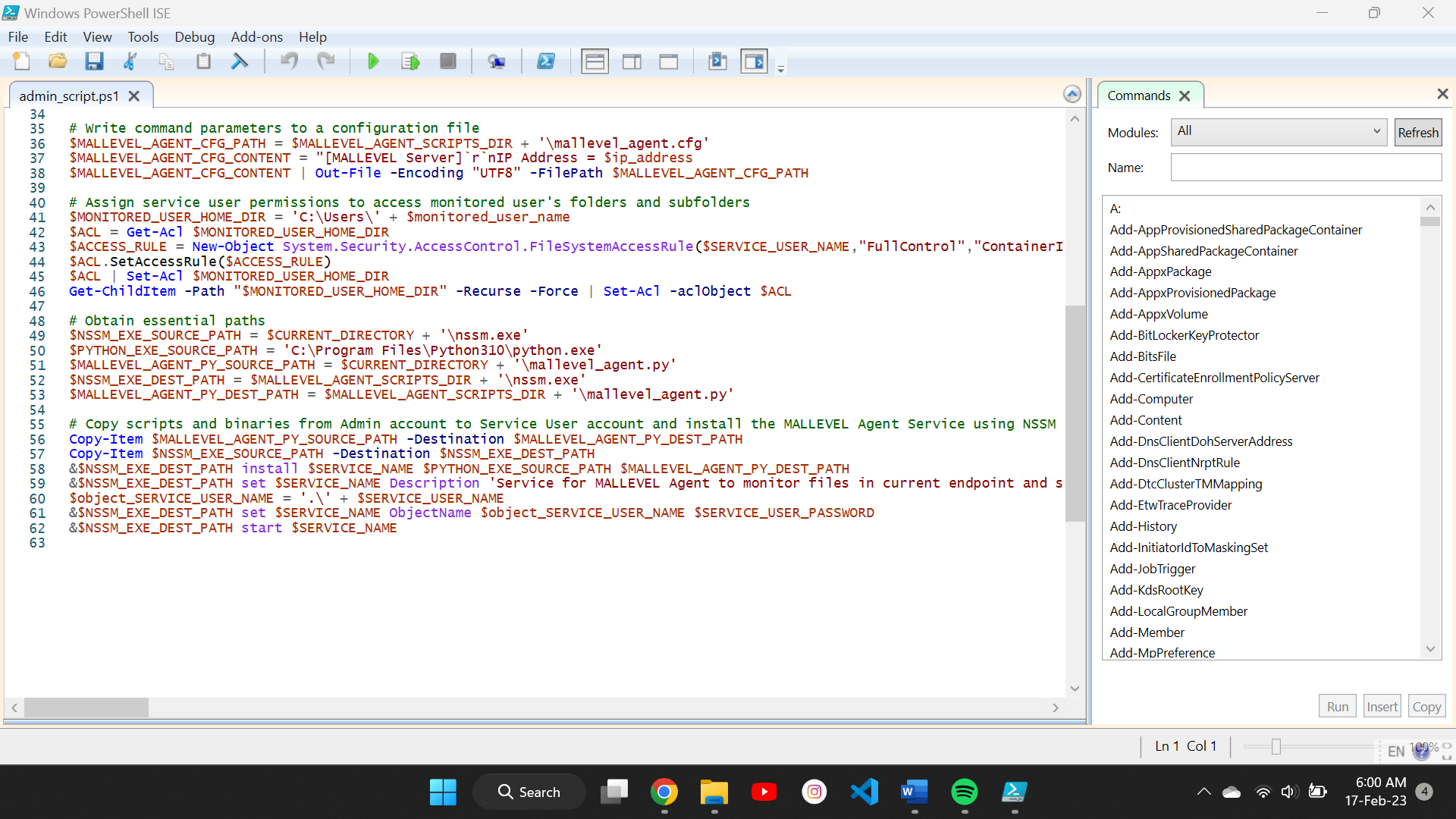
* My fifth subtask was to create a script that would perform automated installation of the agent across multiple endpoints.
* Firstly, I researched on PowerShell commands and syntax.
* Secondly, I planned out some core functions the script should perform: create a service user profile, copy the agent to the service user's directory, and install the service.
* I subsequently implemented PowerShell commands that installs the agent Python script as a service and runs it with administrator privileges. While finding possible methods to install a Python script as a Windows service, I came across this useful tool called Non-Sucking Service Manager (NSSM). Passing in the path to the Python executable and script would create the new service.



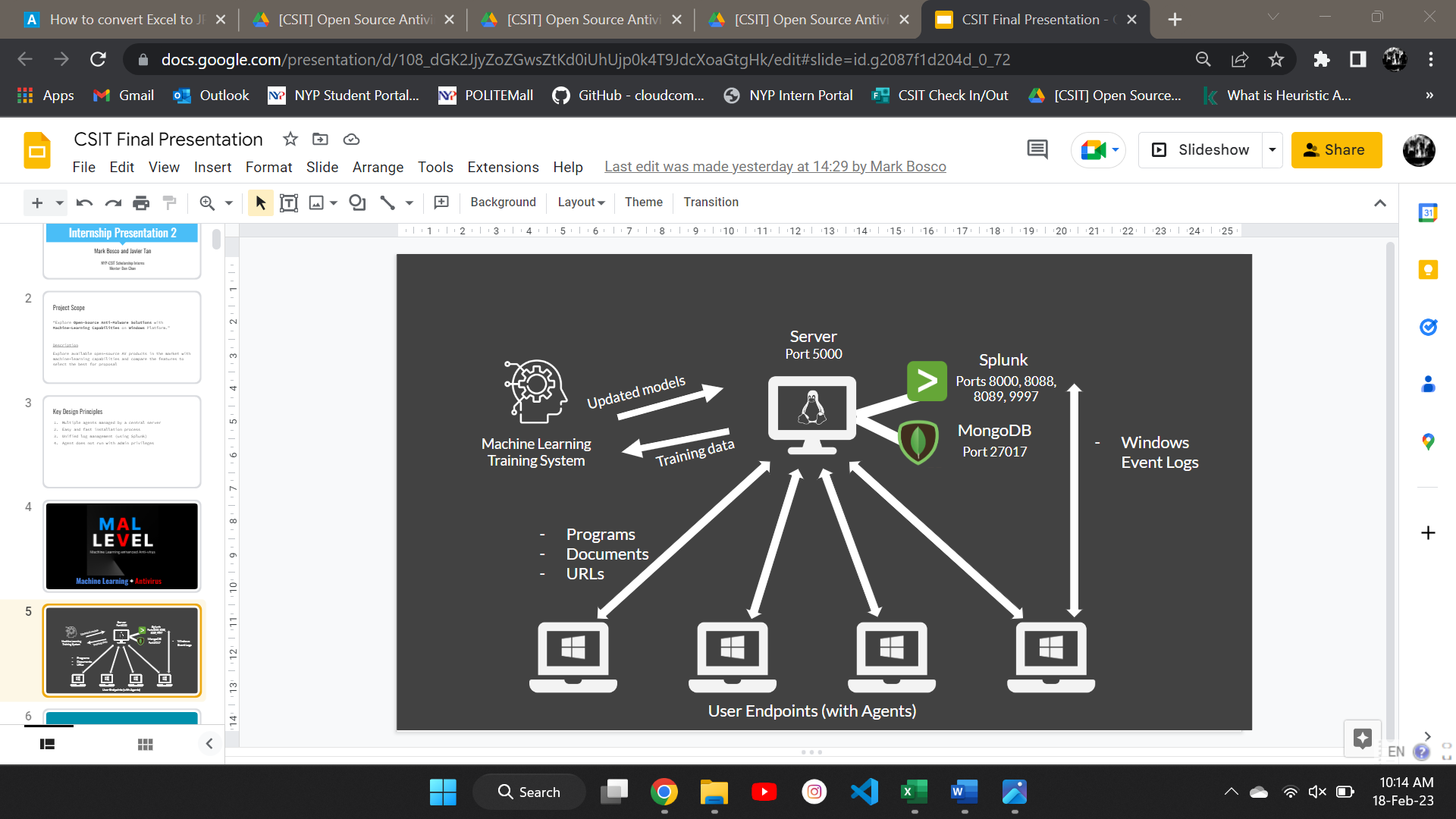
* Once the service functioned expectedly, I researched on areas like creating a service user and 'log on as a service' rights, so that the service would not require administrator rights to run. I also added the PowerShell commands to perform those functions.
* I subsequently tested the installation script on a laptop and verified that the agent service started automatically, establishing a connection to the anti-virus server. Shown below are screenshots of the created service and the PowerShell script:

Graphical user interface, text, application

Description automatically generated



* Shown below is the diagram of the complete agent-server architecture:



Deliverables

* Create agent script for scheduled file monitoring on endpoint.
* The agent script was developed in Python just like the server scripts.
* It would monitor specific directories (e.g. Downloads) at a specific interval (e.g. hourly) and send any new files to the MALLEVEL server for scanning.
* Implement file whitelisting feature.
* Mitigate the issue of false positives generated by machine-learning models.
* Prevent unnecessary scanning of known good files.
* Develop file quarantine system.
* Allows administrator to release safe files (restore them to original path within endpoint) and blacklist dangerous files (permanently remove them).
* Connect MALLEVEL server to Splunk for real-time event analysis.
* Besides sending the server analysis logs, update logs, and initialization logs to Splunk, dashboards were also created to provide insightful visualizations on the files being scanned by the server.
* Create script for automated installation of agent across endpoints.
* In an enterprise setting, many endpoints have to be regularly monitored.
* Hence, developing this script allows automating the entire agent installation process to improve efficiency.
* The installation script only requires a one-off execution with administrative privileges – the agent service will be run by the service user account. This conforms to the security principle of least privilege.

Problems Faced

* I felt that developing the PowerShell script was the most challenging part of this project. This was because I had not done anything related to PowerShell for a long time, and I faced numerous errors while creating the script.
* In the process of implementing some functions, I faced issues such as privilege errors, and failure to start the created service. I managed to solve some of these errors by looking for similar issues faced online, while I had to trial-and-error to troubleshoot some of the other issues.
* I initially thought of converting the Python agent script to an executable using tools like pyinstaller or py2exe. However, the executable did not run successfully after the conversion due to issues with the dependencies.
* Although I spent some time to try and troubleshoot the errors, the packaged program still did not manage to run successfully.
* Instead of wasting more time on trying to solve the errors, I tried to source for alternative methods to run the Python agent script itself as a Windows service. Fortunately, I managed to find NSSM after some research – the tool was easy to use and suitable for this use case.
* The final integrated solution was working perfectly on my Windows laptop. However, when I ported it over to the Linux laptop, the anti-virus server had issues starting up. It took awhile to pinpoint the issues.
* There were inconsistent encodings in some malware signature files which led to issues when the anti-virus server was loading the signatures, i.e. the files did not have Unix encoding in the Linux laptop, leading to the Python UnicodeDecodeError. I fixed this by using the Python codecs library to ensure all files were saved with consistent encoding, such that it worked on both Windows and Linux.
* There was also a difference in path separators between Windows and Linux which led to errors with processing files, due to invalid splits of file paths based on the separators. Therefore, I had to review the codes again and rectify the path separators used in the scripts.

Experience Gained / Lessons Learnt

* I worked on building features similar to those implemented in real-world anti-viruses. This allowed me to learn about some inner workings of existing anti-virus solutions such as how whitelisting and quarantining are actually implemented.
* It was a valuable experience to develop this anti-virus agent-server architecture that could be deployed in an enterprise setting.
* I also managed to refine my debugging skills as there were countless errors faced during development, but I eventually troubleshooted them successfully by doing continuous research and not to give up easily.
* I also learnt about the importance of self-learning; I had to do lots of research to find out the various methods to implement certain features and be able to decide which approach was the most suitable one for the scenario.

**Enhancement 1: Improving Support for Scanning Compressed Archives**

* The anti-virus server is currently only able to auto-extract a limited number of archive formats (e.g. .zip, .7z, .gz, .bz2).
* Hence, a possible enhancement for this feature would be to include support for other archive formats such as .rar, .tar, etc. Support for more file types would definitely improve the overall functionality of the solution.
* Another improvement could be to deal with password-protected archives – possibly allow users to input the password via the web UI, so that the anti-virus server can extract the archive with the specified password. However, this may not work well in the agent-server architecture, since new archives would be automatically sent to the server for scanning.

**Enhancement 2: Dynamic Machine-Learning Training System**

* What this means is that the machine-learning models will be continually trained on the latest data or samples (e.g. PEs, PDFs, Office Documents).
* This may involve relying on malware repositories (e.g. MalwareBazaar) and threat intelligence platforms (e.g. AlienVault OTX) to continually fetch new data and samples for training and testing of machine-learning models.
* Files submitted to the MALLEVEL server can also be used as new training and testing data for these models – this feature would require verification of files (whether it is actually safe or malicious) to avoid the issue of unreliable data which can lead to high false positives and false negatives during deployment.
* Implementing a dynamic machine-learning training system may help to improve the prediction capabilities of the machine-learning models, as they keep up to date with zero-day exploits and threats.

**Enhancement 3: Improved Customizability of Existing Solution**

* In an enterprise setting, there might be different user groups to manage and monitor.
* Therefore, the solution could be improved by offering more granular configurations to cater to the different user groups.
* Examples include different directories to be monitored within different endpoints, and different scan schedules for different endpoints.