

**Project Report**

**Project Supervisor:** Joseph KEHOE



(Security, 2023)

**Student ID:** C00276038

**Student's Name:** Fionn Finane

**Course:** 4th Year Cybercrime and IT Security BSC (Honours)

# **Project Outline**

1. **General Issues**
	* Discuss problems encountered during the project and their resolutions.
	* Document achievements throughout the project.
	* Focus on aspects that we managed to achieve concerning the FYP.
	* Evaluate the lessons learned through the entire FYP process.
	* Consider approaches that we could have taken differently hypothetically
2. **Technical Issues**
	* Report any deviations from the initial design and additional research required.
	* Detail difficulties faced during project implementation.
	* Address issues with tools or software used in the development process.
	* Describe testing methods employed to assess the reliability of the software product.
3. **Documentation Standards**
	* Ensure all documentation follows the Institute's standards for formatting and presentation.

**Conclusion**

The outline above highlights the main objectives across the designated pages, ensuring clarity and thoroughness in the project report.

Our organised approach enhances the document's overall structure and guarantees it meets the lofty standards for a comprehensive report.

Concentrating on these key elements effectively communicates the project's goals and findings, offering readers a clear path to understanding the subject matter's complexities.

To conclude, this section has already provided an appropriate explanation to ensure we have an outline of the level of detail and the implementation of the higher standards in this report.

We have detailed specifications for the details in the core sections explored here, including the General Issues and all areas under this section that focus on the general key issues that focus on the project design, plan, and specifications over development or functionality.

The last key section is the Technical Issues and all the areas that relate to this section that focuses on technical documentation of the functionality and development, unlike General Issues except excluding all the related areas according to general issues we found in the project.

**Table Of Contents**

[**Project Outline** 2](#_Toc196681106)

[**Introduction** 4](#_Toc196681107)

[**General Issues** 5](#_Toc196681108)

[**Problems Encountered** 5](#_Toc196681109)

[**Achievements** 6](#_Toc196681110)

[**Unachieved Goals** 7](#_Toc196681111)

[**Lessons Learned** 7](#_Toc196681112)

[**Reflections for Future Projects** 8](#_Toc196681113)

[**Technical Issues** 9](#_Toc196681114)

[**Design Deviations** 9](#_Toc196681115)

[**Implementation Difficulties** 10](#_Toc196681116)

[**Tool and Software Issues** 11](#_Toc196681117)

[**Testing Methods** 12](#_Toc196681118)

[**References** 13](#_Toc196681119)

# **Introduction**

Hopefully, we will summarise and grant a solid introduction to the FYP Project Report.

In this report, we should accomplish a confident and well-documented report that provides appropriate detail and clarity on the key areas we will see delivered.

Starting with the general issues that arose throughout the FYP development process, including the learning outcomes, unforeseen errors and obstacles that we encountered through the development of the Final Year Project, how we resolved the issues or difficulties that include both the general or typical nonspecific issues, and the technicalities that are often involved in a project such is this.

What we mean by technicalities is simply the concerns and obstacles or talking about the development process as a whole, but more focused on the Project development that covers the technologies utilised, the functionalities, and the programmable side of things that will include the finalised coding libraries, programming languages utilised, the justification for the overall structure of the project in a more detailed format, and finally we will cover some aspects found in our general issues, we will see some crossover relating to the further development that we would strongly desire to pursue concerning future modifications that we may implement to for instance enhance or enable additional features or functionality, areas we manage to accomplish and unsuccessfully implement in the final project.

We will have the following sections that we will explore in this report, which goes as follows: Starting with our Project Outline for concise and enhanced readability and clarity, now we then move onto our Introduction, which gives a concise summary brief of the Project Report, followed by our Table Of Contents, to provide the entire structure and layout of the sections in this report.

Subsequent sections are the core sections of interest in this report: the General Issues, in which we focus on the aspects of the project, such as project planning and specifications, and the Technical Issues, focusing on the details attached to the project's development purpose.

Lastly, like all good documents, we will have a section that appropriately and correctly provides the proper citations and necessary sources referenced in this report.

With that said, we should hopefully accomplish and successfully cover all the areas for concern that should provide the reader with clear and concise documentation that provides and gives justice to the painstaking work that we put into this project to produce a functional and working final product, that we can gain a sense a proud from and with that said let us navigate to the next section in this report, an introduction into the General Issues and all areas that concern this section.

# **General Issues**

We will navigate to one of the two main sections at the core of this report, the General Issues section.

Here, we will cover the problems encountered, the accomplishments and unachieved goals, and the learning outcomes concerning project planning and finalised project specifications.

Finally, we will end this section with any future development we desire to implement into the Final Year Project.

## **Problems Encountered**

Here is a description of the challenges and resolutions.

Like all projects, there is no actual growth or progress in a project such as this if we do not encounter numerous challenges or problems when fine-tuning a reiteration of the initial specifications, only when we find it strictly necessary to alter the original project specifications.

However, in general, the problems here briefly mention and emphasise the overall problems in a summarised manner and do not focus on the specific project development technicalities and the overall programming-heavy perspective.

With that said, let us go through the problems we encountered.

We start with the most obvious one, which concerns this project.

We found this without reviewing the technical issues from the coding and project development perspective.

However, this project started with many adjustments to the prime candidate, which would become the backbone and hopefully handle a lot of the heavy lifting for us and become part of our overall Malware Analysis element of the project, which is a multi-architectural binary analysis tool called Angr ( Open-Source Tool ) which had to abandon due too many technical issues which we will go into when we discuss the Technical Issues.

The resolution to this problem, as mentioned, we decided to abandon Angr, which meant, unfortunately, we had to change strategy.

In the end, we instead used three prime candidates that are tools for overseeing the three main binary types we want our project to read, which are the MACH-O binary for IOS, Portable executable files for Windows OS, and finally, Executable and Linkable Files for Linux Distributions.

Again, we will go through the coding process and the tools, which are Python libraries, as part of GitHub public repositories in the Technical Issues section.

We note this in the General Issues because it is part of the project's specifications and impacts project planning.

The next problem we encountered was the desertion of the Angr tool, which significantly impacted the project planning and further progress of the project objectives across the project duration.

So, the impingement of time management caused the project schedule and objectives strictly required for completion through the many weeks and months of the project's duration to be delayed by two weeks out of the project's total duration.

For the resolution, we had previously explored two of the three tools during the early research stage of this project, and this means we only had to research and find one tool for the analysis of the IOS malware for the project.

We will discuss this issue with the aspects that affected the project development and functionality later in the report.

However, it is important to mention here that this problem was both a general and technical issue, which is why we mention it in this section.

Another issue was with an element of our specifications for the project, which are the exceptional goals for the project, which is implementing a malware classification system, which, because of the level of complexity involved, was not anticipated an additional week to complete due to complexity and functionality, which we will talk about in the Implementation Difficulties section part of the Technical Issues.

The resolution for this problem was because this did not become an issue that escalated further.

It improved the implementation, which we will discuss more in Technical Issues.

The last problem we did encounter is the web application framework to build.

Our web application encountered some more technical issues, which we will discuss in the Design Deviations and the Implementation Difficulties section, due to the computability issues that arose.

The resolution was simple.

We used a web application framework that was more compatible with Python, which made sense for lightweight web applications that fit our project.

However, we will discuss the technical difficulties in the relevant sections later in the report.

## **Achievements**

Like all projects, some accomplishments happened throughout the process.

Most notably, in our case, we have a decent number of achievements made throughout the journey, from design and planning to development and testing and, finally, our final project or the finished production.

Let us start with the most significant achievements, which were successful.

Despite all the issues we encountered throughout the research and development side of things, we still managed to solve through the journey.

The first achievement was completing the minimum specifications early on in the project's progress.

We had managed to implement a working prototype that could read in binary data, except that it must successfully exclude any file types or binary formats that are not part of our approved binary and file formats.

As stated in a previous section, we built our malware classification system to determine the likelihood of supposed malware files being both malicious and their appropriate malware category or type.

It is to accomplish the highest level of accuracy possible and peer-reviewed to provide clarity for further testing and development reasons accomplished by integrating the Virus Total API to gain a consensus on improvements or fixes needed to improve the malware classification logic.

We successfully implemented the web application frontend and backend through Python.

Finally, we implemented a functional project that allows the malware classification system to run in our simulated terminal window on the web application that can scan multiple files rather than only processing a single file at a time, which is impractical.

We will talk about these functionalities more in the Implementation Difficulties section in Technical Issues.

## **Unachieved Goals**

Exploring the aspects of the project that were unfortunately not achieved.

Luckily, in this journey through our project, we accomplished many of the goals we set out to achieve initially, but we still liked all projects.

We could not incorporate or think of everything we wanted to implement into the project, so here are some goals we did not accomplish.

We would start by incorporating and developing our malware analysis tool, which is a static analysis tool to implement capabilities that can function as a dynamic analysis tool, so a dual analysis tool and develop utilising libraries such as Scapy to implement network analysis element into this tool, would make it an interesting feature to develop.

The last thing we would have liked to include is the ability to set up a basic sandbox environment so we can observe the malware executing in an isolated environment through virtualisation and maybe containerisation with software such as VirtualBox and maybe Docker would be a cool thing to implement if we were working with a more extended project duration.

## **Lessons Learned**

Insights gained throughout the process.

The biggest lesson that we learnt through this process is that soft skills that would include determination, resilience, concentration and consistency were key in this project development and planning, setting out and sticking to strict deadlines the project supervisor set throughout the week to accomplish for the next week, and found that the process due to how high pressured and hands-on makes it very challenging, the ability to be adaptable to make the judgement calls that needed immediate decisions in order to develop a project and final product that we would be proud of dedicating the countless person-hours outside of college that speaks for how far long the project has progressed during the duration of this entire project duration from start to finish.

We have also gained an appreciation for malware analysis due to both researching side of things and a possible career area of interest shortly.

## **Reflections for Future Projects**

Noting all considerations for what could be improved.

Here, we ensure that through this project, we can always try to improve our projects, whether it is the coding to be more professional programming level.

However, we will mention that it would be nice to try and modify this process to make it more efficient and scalable.

One key improvement area could be the tool's ability to analyse and classify a broader range of malware types, making it more robust. In addition, we could refine the user interface to make it more intuitive for novice and advanced users, allowing for more straightforward navigation and understanding of the results.

Automation of specific processes, including integrating machine learning for better detection, could also be explored in future versions. The goal is to continuously enhance the tool's effectiveness while ensuring it remains adaptable to the evolving landscape of cybersecurity threats.

We should add depth to our reflections and provide a more comprehensive view of potential future improvements!

# **Technical Issues**

We will navigate to the second core section of this report, the Technical Issues section.

Here, we will cover the Design Deviations, ensuring we document any deviations affecting the project design and the rationale.

Implementation difficulties are the barriers encountered throughout the project development process and tool and software issues.

We specifically focus on issues concerning the tools and software used in the project development process.

The last section concerns the different Testing Methods we used to thoroughly assess the Final Year Project that benefits in developing the finalised product or project.

## **Design Deviations**

Here, we will explore the changes from the initial design that occurred and the rationale.

So, the project design deviations from the initial design are both about the malware analysis report tool the project is running in the background working on the backend of our website application that is the workaround to implement a GUI frontend for this project mirroring a similar structure to a file dropper web application with inspiration from problems like Virus Total with the addition of a simulated Live terminal window included within the Web Application frontend, along with showing the generated malware reports sorted into Markdown and PDF documents.

So, the design process changed throughout the development process.

We most notably started with a pure terminal or command line output to the screen within an integrated development environment.

In our case, we are using Visual Studio Code to run a given directory for testing purposes, which we will cover in the Testing Section.

We even implemented further development to have our Malware Classifier running through predefined terminal outputs that use our Malware Classifier's classification logic, with additional features that include mitigation recommendations applicable to each Malware type detected.

So, hopefully, we have painted an accurate picture of the genesis of the current design to the finalised product, which is our frontend Malware Analysis Report Generation Web application.

## **Implementation Difficulties**

Here, we mean the specific issues from a technical perspective that affected the overall approach and implementation of the functionalities that impact project development.

Let us start by going over the specifications for the project.

As part of the minimum specifications, our program must generate a professional-looking malware report exported in Markdown and PDF formats filled with the appropriate binary information extracted specifically included within each report, which includes extraction and classification of strings that include base64 detection, extraction and decoding, file entropic calculation through using two dependent python libraries called NumPy and importantly the library SciPy that is dependent on NumPy.

Communicate with the Virus Total Analysis report via the Virus Total Analysis API.

Lastly, we have included the capability of extracting file details through the Python libraries system and OS.

Now that we have the minimum specifications covered, the implementations we will focus on are the Base64 Extraction, Classification and Decoding functionality, so the idea of this function and why we include it in the malware reports to highlight any encoding that may be going on in a given binary file or file type, as this is an indicator of maybe some Base64 encoding implementation here but the problem arises when we attempted to extract the Base64, even though we tried implementing the strictest criteria possible to account for any hallucinations that might of occurred such as strings that specify file paths would be mistakenly slip past our criteria, so we decided to account for this by creating a simple block list to tell to reject all possible strings that may fit the criteria for Base64 encoded strings do to the file length being divisible by four and following the alphanumeric and safe URL encoding achieved through the use of padding being accounted as part of the matching a simple regular expression pattern we implemented into our Base64 extraction and classification logic as this ensures we reduce the number of false positives and ensure we have strong and accurate functionality as this does not cause a knock on affect to the Decoding process which is essential for the decoding process to work correctly.

Another implementation difficulty we encountered was the malware classification system.

We initially wanted to scan and classify our malware samples by scanning an extensive collection of Yara rules, which, since malware researchers curate and create these rules, should not require any additional programming logic.

However, we will mention more in the Tools and Software Issues section.

However, instead, it required us to create a classification logic for each malware type based on criteria like API calls that work for Portable Executables, Executable and Linkable Format and Macho.

Another thing we encountered was putting the YARA rules under its malware category so that we could still check our collection of YARA rules under its category.

## **Tool and Software Issues**

Here, we will go over the tools and software issues we faced throughout the project's development that affected the project's main functionality.

Here, we will cover the Tools and Software issues encountered, including trying to host our web application through a web hosting service that is most compatible with project files that make up our entire project, the issues with our web application, the tools and we will focus on the base64 functionality issue that we did not highlight already from a previous section, the issue that caused us to drop incorporating Angr entirely and highlight the primary tools we used for handling the binary classification for us, which was Pefile ( Portable Executables ), Pyelftools ( Executable and Linkable Format ) and finally for our MACHO utilising macholib,

Starting with the Base64 functionality issues, the problem we mentioned involved false positives.

We incorporated a fix that was our block list to ensure we could make our Base64 Extraction, Classification and Decoding feature robust.

However, we did before our fix even had to incorporate not only a stricter regex or regular expression to utilise the Safe URL encoding implementation to our Base64 standard criteria, but we still found that we were incorporating decoding outputs that are not the printable ASCI text to render the output readable, the reason we found is that unless we programmed extra conditional logic only to allow stricter criteria for the decoding process to work as intended, we have specified as well that unless Base64 string was of a specific length that would ensure that we only accept both the encoding scheme and the decoding output that we deemed appropriate to add into our malware reports.

So, it meant that we needed to ensure that not only did we accept decoded output that fits printable text but also, we had to incorporate logic that only added and detected Base64 encoded strings that were not just divisible by four but were of lengths that are multiples of four as well to ensure we only accept actual Base64 encoded strings.

The next thing is the Angr tool.

We initially started by utilising a preexisting open-source tool specialising in handling binary analysis for Linux, Windows, and macOS.

We encountered a lot of depreciated dependencies or libraries and functions, and it caused too much hardship than it was worth.

There was an intense learning curve in understanding and using Angr correctly, which we would like to incorporate if we had an extended project duration.

Unfortunately, we concluded that we had to abandon utilising this tool and use the tools we will mention next.

So the tools we are utilising are Macholib python module for handling the analysis and evaluation of macho binary, specify extracting the import and export functions and the string location since macho is the most complex when compared to Linux's ELF and Windows PE, which are more straightforward in there approach so that was difficulty because macho binary and its macho binary file types associated supports multi-architecture when it comes to MACHO application file types, it makes detecting the magic numbers associated with this binary type is not as straightforward here is typical application file types like .bundle, .a, dsmeg and so on.

Nevertheless, this required programming conditional logic to extract the strings from sections such as TEXT and DATA, the common areas for extracting strings, and the sections that import and export functions in MACHO, ELF, and PE.

We had to develop functions to classify, run, and execute our Yara rules, which we are utilising as a failsafe solution that involves collecting many Yara rules.

As these rules are not always valid, that was an unexpected issue.

As a result, we created the following functions to solve the issue.

We had to find a prime candidate to allow us to host our web application easily and integrate it with the Flask application framework.

After countless tries, we stumbled onto a platform called Render that allows us to host our web application by setting up a simple file filed with all the dependencies we need to successfully deploy our web application utilising our GitHub latest comments or changes made to our files that make up our web application and the whole project.

## **Testing Methods**

In this section, we aim to adequately describe all the testing procedures using the effectiveness metric in the overall journey through the functionality and development.

We started first by evaluating our malware analysis tool on existing malware samples.

The testing method here involved downloading a diverse collection of malware samples and searching the internet for existing malware samples that required searching for malware samples specific to each operating system I am trying to classify and detect effectively.

For example, we will source some of these resources that were everything from GitHub Public repositories like the official Practical Malware Analysis-Labs, Macos-Malware-Samples, and the Linux-Malware-Samples are just some of the many malware samples that were utilised in this project, as well as MalwareBazaar for some compare and contrasting between older samples of the same malware vs the latest versions of these malware samples.

We also used a collection of binary samples from a GitHub repository to compare the reliability of our malware analysis tool's binary analysis and extraction capabilities to ensure the primary minimum functionalities work as intended.

Finally, the primary testing method is the incorporation, from a programmable perspective, of much error handling through debugging of our code for both the Flask web application and all the files and programs that are essential for running and handling the processing of the malware samples working on the backend of things.

# **References**

Andrew Mccarthy, D. S.-k. (2023, November 30). *Getting started*. Retrieved from Django Software Foundation Django Project: https://docs.djangoproject.com/en/4.2/intro/

Foundation, P. S. (2024, November 9). *base64 — Base16, Base32, Base64, Base85 Data Encodings*. Retrieved November 11, 2024, from Python documentation: https://docs.python.org/3/library/base64.html

Foundation, P. S. (2024, August 16). *Markdown 3.7*, 3.7. (P. S. Foundation, Editor) Retrieved November 7, 2024, from PyPi: https://pypi.org/project/Markdown/

Foundation, P. S. (2024, June 11). *markdown-pdf 1.3*, 1.3. (P. S. Foundation, Editor) Retrieved November 7, 2024, from PyPi: https://pypi.org/project/markdown-pdf/

Foundation, P. S. (2024, November 9). *re — Regular expression operations*, 3.13.0. (P. S. 2., Editor) Retrieved November 10, 2024, from Python documentation: https://docs.python.org/3/library/re.html

Foundation, P. S. (2024, November 9). *string — Common string operations*, 3.13.0. Retrieved November 11, 2024, from Python documentation: https://docs.python.org/3/library/string.html

Foundation, P. S. (2024, May 26). *virustotal-python*, 1.1.0. (P. S. Foundation, Editor) Retrieved November 10, 2024, from PyPI: https://pypi.org/project/virustotal-python/

Gedam, P. a. (2024, January 1). *Core Concepts - angr documentation*. (P. Gedam, Editor) Retrieved November 10, 2024, from Angr documentation: https://docs.angr.io/en/latest/core-concepts/toplevel.html

Gedam, P. a. (2024, January 1). *Introduction - angr documentation*. (P. Gedam, Editor) Retrieved November 10, 2024, from Angr Documentation: https://docs.angr.io/en/latest/quickstart.html#citing-angr

GeekforGeeks. (2023, November 18). *Python sys Module*. Retrieved November 11, 2024, from GeekforGeeks: https://www.geeksforgeeks.org/python-sys-module/

GeeksforGeeks. (2023, December 21). *Python Math Module*. Retrieved November 11, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/python-math-module/

GeeksforGeeks. (2024, October 5). *Flask Tutorial*. Retrieved November 10, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/flask-tutorial/

GeeksforGeeks. (2024, August 1). *OS Module in Python with Examples*. Retrieved November 11, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/os-module-python-examples/

GeeksforGeeks. (2024, February 26). *What is Hashing?* Retrieved November 08, 2024, from GeeksforGeeks: https://www.geeksforgeeks.org/what-is-hashing/

getbootstrap.com. (2023, November 22). *Themes*. Retrieved from Get bootstrap: https://themes.getbootstrap.com/guide/

Hartman, K. G. (2013, May 18). Calculate File Entropy. Retrieved November 10, 2024, from https://kennethghartman.com/blog/calculate-file-entropy/

Honig, M. S. (2012). Basic Static Techniques. In S. Honig, *Practical Malware Analysis* (pp. 13-15). San Francisco, California , America: William Pollock. Retrieved November 11, 2024

joxeankoret. (2024, January 1). *Malware Analysis*. Retrieved November 7, 2024, from GitHub: https://github.com/joxeankoret/pyew/wiki/MalwareAnalysis

Mikesiko. (2025, April 27). *PracticalMalwareAnalysis-Labs: Binaries for the book Practical Malware Analysis*. Retrieved from Github: https://github.com/mikesiko/PracticalMalwareAnalysis-Labs

Mitchell Hashimoto, C.-F. o. (2025, April 27). *Cloud Application Platform | Render*. Retrieved from Render: https://render.com/

Philippe Biondi & Scapy Community. (2024, January 1). *Introduction — Scapy 2.6.1 documentation*, 2.6.1. (P. Biondi, Editor) Retrieved November 10, 2024, from Scapy Read the Doc: https://scapy.readthedocs.io/en/stable/introduction.html

Revision, V. (Ed.). (2022, January 1). *Welcome to YARA's documentation! — yara 4.4.0 documentation*, 4.4.0. Retrieved November 10, 2024, from Yara Read the Doc: https://yara.readthedocs.io/en/stable/index.html

ronaldoussoren. (2025, April 26). *ronaldoussoren/macholib: macholib can be used to analyze and edit Mach-O headers, the executable format used by Mac OS X. It's typically used as a dependency analysis tool, and also to rewrite dylib references in Mach-O headers to be @executable\_path rel*. Retrieved from Github: https://github.com/ronaldoussoren/macholib

Schools, W. (2023, November 22). *Bootstrap 5 Tutorial*. Retrieved from W3 Schools: https://www.w3schools.com/bootstrap5/index.php

Schools, W. (2023, 11 30). *Django Tutorial*. Retrieved from W3 Schools: https://www.w3schools.com/django/django\_intro.php

Security, R. (2023, January 1). Malware analysis image. Viale Marco Polo,11900154, Roma, Italy. Retrieved November 7, 2023, from https://www.redfrogsecurity.it/red-site/wp-content/uploads/2023/11/Malware\_Analysis.jpg

Shoshitaishvili, Y. a. (2015, January 1). Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. *NDSS*. Retrieved November 10, 2024

Shoshitaishvili, Y. a. (2016, January 1). SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. *IEEE Symposium on Security and Privacy*. Retrieved November 10, 2024

Software, C. (2024, January 1). *Dependency Walker 2.2.6000.9*, 2.2.6000.9. Retrieved November 10, 2024, from Chocolatey Software: https://community.chocolatey.org/packages/dependencywalker

Software, H. (2024, January 1). *PE.Explorer. Resource Editor. Disassembler.* Retrieved November 8, 2024, from Heaventools Software: https://www.pe-explorer.com/

Stephens, N. a. (2016, January 1). Driller: Augmenting Fuzzing Through Selective Symbolic Execution. *NDSS*. Retrieved November 10, 2024

Team, C. G. (2024, January 1). *GitHub - CyberGrandChallenge/samples: DARPA Cyber Grand Challenge Sample Challenges*. (T. Vidas, Editor) Retrieved November 10, 2024, from GitHub: https://github.com/CyberGrandChallenge/samples

team, N. (2024, August 18). *NumPy*, 2.1. (N. team, Editor) Retrieved November 10, 2024, from NumPy: https://numpy.org/

team, T. M. (2024, January 1). *Matplotlib — Visualization with Python*. (T. M. team, Editor) Retrieved November 10, 2024, from Matplotlib: https://matplotlib.org/

Team, V. S. (2025, April 26). *MalwareSamples/Linux-Malware-Samples: Linux Malware Sample Archive including various types of malicious ELF binaries and viruses.* Retrieved from Github: https://github.com/MalwareSamples/Linux-Malware-Samples

Team, V. S. (2025, April 27). *MalwareSamples/Macos-Malware-Samples: Macos Malware Samples*. Retrieved from Github: https://github.com/MalwareSamples/Macos-Malware-Samples