Logo, company name

Description automatically generated

CTEC3451 – Development Project

Finley McGonigle P2595958

**Exploring the Effectiveness of Keyloggers for the Use of Penetration Testing**

Supervisor: Nick Ayres

Date submitted: 04/05/2023.

Word count: 7744

Written by P2595958

**Acknowledgements**

I would like to thank my Supervisor Nick Ayres for his continuing guidance and support throughout my project. I would also like to express my gratitude to my family and my girlfriend, for their continued support which made finishing the project possible. Finally, I would like to extend my thanks to the coding communities of the web, in which I have learned valuable knowledge which I will never forget.

**Abstract**

This dissertation will document the process of developing a Keylogger for the utilisation of penetration testing. A keylogger refers to a piece of spyware, adapted to obtain more advanced features, which could be developed further and become a common tool used by penetration testers. This project centres on the enhancements and developments made to the basic concept of a ‘keylogger’, in addition to the method the tester uses to execute the programme onto the target system. Hence, the objective of this project is to gain a better understanding of ‘Keyloggers’, including the information that it is possible to harvest using them. In addition, this project aimed to provide a greater overall depth of skills, derived from system development.  The key findings of this project encompassed learning the skills required to code in Python, as well as developing a strong understanding of the architecture of malware; permitting it to be adapted and built to fit a specific purpose. It is anticipated that the Keylogger developed could contribute to the gap within the cybersecurity industry, which currently presents little evidence of Keyloggers being employed in a professional penetration testing environment. Hence, this project and the features the software holds could modify this. To conclude, this project adds features usually associated with spyware, onto a robust malware program that could be considered usual for Penetration testing.

Table of Contents

[Table of Figures 5](#_Toc134113570)

[Introduction 6](#_Toc134113571)

[Objectives 9](#_Toc134113572)

[The System and Development 10](#_Toc134113573)

[Development Lifecycle 10](#_Toc134113574)

[Main Code 12](#_Toc134113575)

[Encryption 26](#_Toc134113576)

[Disguising the Software 29](#_Toc134113577)

[Critical Evaluation 32](#_Toc134113578)

[Product Evaluation 32](#_Toc134113579)

[Project Evaluation 33](#_Toc134113580)

[Evaluation of Tools 34](#_Toc134113581)

[**Visual Studios Code** 34](#_Toc134113582)

[**GitHub** 35](#_Toc134113583)

[**Reddit & Stack Overflow/ Coding Communities** 35](#_Toc134113584)

[**Resource Hacker** 36](#_Toc134113585)

[**Libraries** 36](#_Toc134113586)

[**Problems Faced** 37](#_Toc134113587)

[**Product Appraisal** 37](#_Toc134113588)

[Conclusion 37](#_Toc134113589)

[References 38](#_Toc134113590)

# **Table of Figures**

[Figure 1: An example of AES encryption program 8](#_Toc134057800)

[Figure 2: Use case diagram originally designed for the code. 10](#_Toc134057801)

[Figure 3: A image showing the original libraries and modules used. 13](#_Toc134057802)

[Figure 4: An image which shows the naming of the keystroke file and email detail variables. 13](#_Toc134057803)

[Figure 5: A snippet of code that shows the timer and the network information included in the email. 14](#_Toc134057804)

[Figure 6: The code that made the main body of the email and the timer, as well as the attachments. 15](#_Toc134057805)

[Figure 7: the one-time use ‘app password’ generated on Gmail. 18](#_Toc134057806)

[Figure 8: The result of a successful test of the prototype. 19](#_Toc134057807)

[Figure 9: The output of the prototype keylogging file. 20](#_Toc134057808)

[Figure 10: A folder showing multiple tested executable prototypes. 20](#_Toc134057809)

[Figure 11: A snippet showing how the clipboard data is acquired. 21](#_Toc134057810)

[Figure 12: A snippet showing the updated and final email body. 21](#_Toc134057811)

[Figure 13: A snippet showing the code that utilises the MIME module which attaches things to emails. 22](#_Toc134057812)

[Figure 14: The second timer which controls the frequency of requested data. 22](#_Toc134057813)

[Figure 15: A snippet that shows the function to grab microphone audio. 24](#_Toc134057814)

[Figure 16: A snippet which shows how the microphone and audio is attached. 25](#_Toc134057815)

[Figure 17: A screenshot that shows a successful test. 26](#_Toc134057816)

[Figure 18: A screenshot of the 15-frame result of the webcam footage. 26](#_Toc134057817)

[Figure 19: A snippet of the code required to implement encryption. 27](#_Toc134057818)

[Figure 20: a snippet showing the decryption file. 28](#_Toc134057819)

[Figure 21: A image showing a successfully encrypted keystroke file. 29](#_Toc134057820)

[Figure 22: An image showing a successfully decrypted keystroke file. 29](#_Toc134057821)

[Figure 23: an image showing the resource hacker GUI. 31](#_Toc134057822)

[Figure 24: The icon used to mask the executable. 31](#_Toc134057823)

[Figure 25: An image showing the executable properties after it’s been masked, proving it’s still executable. 31](#_Toc134057824)

# **Introduction**

Penetration testing refers to the practice of directing an authorised attack on a system/network, with the primary function of detecting vulnerabilities and weaknesses in the system’s integrity (National Cyber Security Centre, 2017). Penetration testing can be categorised into three subtypes: Black Box, Grey Box and Clear Box. Black Box is the first level of testing, characterised by simulating an attack with no knowledge of the internal systems and how the networks operate; and is the one this tool will be designed for. Grey Box testing, however, involves a tester with limited knowledge of the system, while finally, Clear Box testing involves a tester with full knowledge of the system. All three sub-types can be strategically employed in different scenarios when attacking a company, as each one utilises differing tools (Redscan, 2019). For example, a Black Box tester uses distinctive tools for the testing process, such as ‘Fuzzing’, which involves sending malformed bits of information towards a system, and identifying if it returns unexpecting errors which could expose vulnerabilities. In addition, brute force attacks could also be utilised, where programmes like John the Ripper can be employed to guess passwords and access sensitive information potentially stored on the system. Another alternative is social engineering, which involves deceiving employees into disclosing confidential information regarding the system, which could assist the attacker in gaining unauthorised access. While only highlighting a few penetration testing techniques, their effectiveness in gaining access to a system is particularly evident. Thus, the report aims to develop a keylogger tool that can be used for a penetration testing scenario.

This software aims to assist in penetration testing within a controlled and ethical environment to maximise the legality of the product. This software would be deployed by testers usually using black box methodology during penetration testing, since the software gathers a wealth of network information, such as the IP and MAC address. This information can be used for further investigation into the systems, and identification of vulnerabilities if the right approach is used. The main functions of the software are as followed: to access hardware information; establish currently running processes; as well as collect network information of the target machine. These being fairly industry standard, it was determined necessary to use additional intrusive techniques to obtain further sensitive information. This includes acquiring passwords stored on keychains of devices, which can be subsequently encrypted and sent via email in an XML file to the software users’ inbox. Furthermore, keystroke information would be sent via email in a txt file, also encrypted for supplementary security. The network information, however, will appear in a separate email. The email can be sent via the Simple Mail Transfer Protocol (SMTP) using the “smtplib” library in Python, which enables the user to enter a secure email address and password, and also select an email address to send the information to (Rhodes and Goerzen, 2014).

In addition to the functions previously mentioned, the software will obtain the capability to use the device’s peripherals, including the webcam and microphone, if they are equipped with one. The design choice was taken to implement these features with interchangeable options, such as the interval length between microphone recordings and webcam screenshots. This is conducted using libraries such as “PyAudio” and “OpenCV”, which function by capturing the microphone and webcam information (Python, 2020). This facilitates software versatility for diverse penetration testing scenarios. Furthermore, the implementation of the purposeful customisable screenshot feature permits the tester to acquire screenshots of the machine, provided via an encrypted email. This was carried out by testing several libraries such as “Pillow”, “pyscreenshot” and “scrot”, in order to adapt the software for use on Linux if that’s what the tester requires. In addition, clipboard information will similarly be pasted and included in a labelled email. This can be achieved using a cross-platform library called “paperclip” which can interact with a device’s clipboard. These emails will have an adjustable time interval option embedded within the code. Following the successful encrypting and emailing of all the aforementioned information, a decryption program was developed in order to decrypt all the recovered data from the successfully executed program. This was completed by the AES (Advanced Encryption Standard) encryption algorithm, defined as high-level encryption typically used by US government officials, due to its complexity. To use this method, the “pycryptodome” library was imported and employed. The next step included obtaining the key and initialisation vector (IV) (Heron, 2009). A short example of how the code will look in the product can be seen below.

Text

Description automatically generated

Figure 1: An example of AES encryption program

When designing the programme, it was important to consider how the keylogger would be practically executed on a target machine. Ultimately, it was decided to employ a phishing-style email, due to its effectiveness in assisting penetration tester access to a system (Thomas and Yorkshire, 2014). A silent system that allows you to acquire network information, as well as machine information, is renowned to be extremely damaging to the integrity of a system. The explicit use of webcam photos obtains high beneficial value for the attacker, as an advantage can be taken of any employees who leave sensitive information or passwords unprotected around offices or desks. This aligns with the research statistics, whereby 29% of data breaches are due to data negligence, such as weak password policies (Vojinovic, 2019). This would allow the attacker to gain access to the system and escalate privileges. This could consequently result in system browsing, and the ability to install further tools on the system, resulting in total loss of control.

## **Objectives**

The main objective of this development project is to advance my personal knowledge and implement it in practical applications. Having never studied Python, it was a recognisable challenge, but it was considered an opportunity to push the development, to ensure the final project encompassed all the features established in the development stage of the project. However, applying the cohesive knowledge gained from both the software development cycle and penetration testing modules, in addition to existing foundation knowledge gained from studying Cybersecurity, the optimism regarding the success of the project was justified. In conjunction with these resources, it was decided to also utilise supplementary materials found online, such as videos and courses provided by other universities, to learn Python and become familiar with the necessary topics required for this project. The knowledge acquired throughout and following the completion of the project will be utilised to enhance my current portfolio. Thus, meaning the academic objectives for this project are to become fluent in Python, a programming language frequently used for website development, data analysis and machine learning. Proficiency in Python will be extremely advantageous following university, due to its desirability and sought-after nature in the cybersecurity industry. Alongside this, a key objective is to advance cryptography knowledge by researching and developing the encryption and decryption program for keylogger data. While my cryptography knowledge is sufficient, anyone would appreciate the benefits of expanding foundational skills to not only better comprehend the topic but be able to apply it beyond just this particular research project. The final academic objective is to develop a thorough testing plan, due to its utility in identifying the methods used to develop and test the project and allow for critical evaluation which can guide improvements for future applications.

Diagram

Description automatically generated

Figure 2: Use case diagram originally designed for the code.

**The System and Development**

## **Development Lifecycle**

During the development process, it was decided that an agile prototyping methodology would be adopted for the project. Agile prototyping refers to an iterative approach, which will allow incremental progress (Abrahamsson et al., 2017). Agile emphasises using collaboration traditional, alongside flexibility and rapid development and feedback. Being the sole developer, an adapted approach would have to be taken in order to fit the development model. Instead of receiving feedback on the project from what would traditionally be provided by team members or stakeholders, a substituted approach will involve multiple different testing types. To fit my project development of the keylogger, it could commence with a prototype file with the basics, which include the email feature and the keystroke logger with network information. Using the knowledge obtained through research, this would be a fairly simple process, as a firm understanding of the libraries is required. Through the development process, the prototype will undergo continuous testing and refinement, with feedback and required improvements being established following each run.

Another development life cycle that was considered was Waterfall, whereby the project progresses through a series of pre-arranged phases (Bassil, 2012). Developers start gathering the requirements, and it concludes with the completion of the development/ deployment. Waterfall works chronologically, meaning the next phase cannot begin before the previous one ends. This contrasts with agile, which focuses on working in small increments to come together and produce a final cohesive project. The primary reason as to why Agile was opted for over Waterfall, was attributed to the amount of testing required. For example, Agile allowed the code to be completed and refined as it was being worked through, without limiting the sections being attended to. In addition to this, Agile was chosen due to security concerns. To elaborate, the project has the potential to collect sensitive data, so the encryption and decryption would need to be perfected, which may be time-consuming and need constant adjusting and development, which Agile permitted.

The approach to testing will be using unit and integration testing initially. Unit testing will benefit the project, due to its focus on testing individual components and sections of the code. For example, after developing the prototype and testing the initial components’ work, an additional section would be incorporated one at a time, which would be further tested and re-evaluated (Runeson, 2006). Integration testing will also be utilised further down the development schedule, examining how the different components of the software interact, and how they function as a whole (Wynn and Eckert, 2016). For example, how the network information appears in the email, how it is formatted, or even if the network grab was successful. These tests will be run using my own machine to comply with ethical boundaries, but it must be noted a virtual machine will be employed as a preventative measure, to reduce the risk of damaging the hardware should there be an issue. During the latter stages of development, acceptance testing will be conducted to test whether the system meets the requirements initially discussed within the first deliverable. This is a vital test in order to establish success in the project. By employing a combination of testing methods, a completed system will be developed, which achieves the goals defined within the specification. In addition to critical self-reflection and consistent project reviews, frequent meetings with a project supervisor occurred. This enabled the provision of a unique and insightful perspective regarding the development process, and further guidance in addressing any issues. For effective task management using the Agile method, ‘to-do’ lists were formed via ‘Notion’. This allowed for tasks to be visually written and broken down chronologically, which once completed, could be ticked off, thus remaining on track. This no doubt had a positive impact on development, by aiding productivity and the overall organisation of the project.

## **Main Code**

While it must be acknowledged that developing the code with no prior knowledge of Python was a significant undertaking, the most logical approach was to commence by constructing a keylogger that stripped the functions down to the bones. For example, by first creating a keylogger with the basic function of logging keys, and building on the until it has the features intended. The prototype required the ability to have basic keylogger functions, which could be built upon to establish the end goals of the project. The prototype starts with importing the libraries to utilise the functions of the code.

Text

Description automatically generated with medium confidence

Figure 3: A image showing the original libraries and modules used.

Considering the keylogger needed to attain email functionality, the first library to be imported was ‘smtplib’, which permits the sending and receiving of emails. This is essential to complete the functions originally defined, so the attacker can review the key log and network information gathered. Additionally, ‘socket’ is required to obtain network information from the system the file is ‘obtained’ on. The ‘uuid’ is a library which provides for generating Universally unique identifiers, and is used to obtain the MAC address of the system that the file is executed on. ‘Pynput.keyboard’ is a library that grants the keylogger to monitor keyboard input. The final function refers to ‘threading’, which will be used to develop a timer. This allows the attacker to change the functionality of the keylogger, and how long it runs, including the interval length between emails. Combined, these create the necessary libraries and functions to create a prototype system.

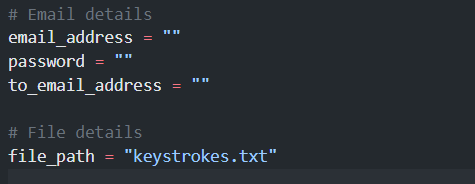


Figure 4: An imagine which shows the naming of the keystroke file and email detail variables.

Following this, the next few lines of code require a few variables which would feature profoundly, which will help define the code used throughout. Simply, we define what we want the keylogger information document to be named. This can be edited depending on what the attacker desires, enabling customisation. The email to which the keylogger information is sent can also be altered and modified for ease of use, as well as the email address the information will be sent.

Despite this, one thing to note, is that if ‘Gmail’ is used over Outlook, the account settings must have ‘allow less secure apps’ turned on, in order to send the email from that account. This is accredited to Google’s high security.

Text

Description automatically generated

Figure 5: A snippet of code that shows the timer and the network information included in the email.

The next step in the process consists of collecting network information on the target machine, which includes network information, IP address, Network name and MAC address. The ‘socket’ module was utilised in this segment of code. Firstly, the hostname of the machine is acquired, this allows the following lines to attain the IP address associated with the hostname. The same process is used to acquire the network name and the MAC address of the machine. Nevertheless, the MAC address is slightly more complex, using the UUID module to get the unique identifier for the network adapter, followed by formatting it as a string. The timer is a vital component of the keylogger, it allows the user to define how much information they want from the system, and how much. The timer is a simply created aspect of the keylogger, its duration is changed in seconds. The interval references how often you require the keystroke information to be sent. A function to handle the key presses is then created. This is completed by using the ‘on\_press(key)’ function, which is called by the ‘Listener’ object. This takes the keystrokes and processes them as ‘key’ which then opens the ‘keystrokes.txt’ file, and subsequently inputs the keystroke followed by a newline character.

Text

Description automatically generated

Figure 6: The code that made the main body of the email and the timer, as well as the attachments.

Sending the email attachments via Python required some research but was accomplished using resources available online. This included stack overflow and other communities knowledgeable about the Python code. Based on this, it was determined the most effective technique to achieve this objective was to use the ‘open’ function with ‘rb’ mode. This means that the file opens in binary mode, and allows the contents to be ‘read’ using the method previously used and stores the data in the variable named ‘file\_data’ (Foong, 2020). Then a simple email was composed and added to the code, by calling the variable names to input the information into the email body. It was important that this was made clear, so the attacker obtains clear and readable facts regarding the system, to assist with penetration testing. Following this, it was decided to create a function that handled the email attachments. The function that creates the connection between the SMTP (simple mail transfer protocol) using the ‘smtplib’, results in the ‘smtp.SMTP’ class, which takes the address and port and translates them as arguments. This then initiates a connection using the ‘starttls’ method, which logs into the email (which the user with identify earlier on) and accesses it using the login method with the credentials provided by the user. Next, ‘sendmail’ is used to identify a method which sends the mail, and does this by taking the email and the necessary information and sending it as bytes, with the attachments being sent via arguments. Furthermore, it was decided that the object decidedly named ‘t’ should be created with the ‘Timer’ class, this would then be used to call the ‘email\_file’ function. This functions to control the frequency of the emails that get sent to the user of the logger, to increase the functionality and ability to adapt it to suit the penetration testing situation. Once again, a timer object is created- named ‘t2’ accordingly. This is responsible for stopping the keylogger after a duration (seconds) has passed, which is specified by the ‘Timer’ class construct, specified by the user. Finally, to end the prototype program, the keylogger is started with ‘listener.join()’ method. This allows the main thread of code to be blocked and waits for the timer to expire and allow the program to finish executing. After editing the script to achieve the desired prototype features, it was time to commence testing to ensure the correctness and functionality of the code. Initially, it was determined that the Python script should be able to be converted to an executable, so the system in which the file is being executed doesn’t need to have the dependent libraries installed. Whereas if it was executed as a .py file inside an API, such as Visual Studios Code, (where it was created) you’d need the dependencies installed, which lacks practicality for a system designed for stealth. It was decided after some research that the ‘auto-py-to-exe’ program was to be alternatively used. This was a simple install with clear instructions shown on the website. Once the commands on CMD had been executed, an easy-to-follow GUI is displayed, allowing you to select the file from a file browser, and select whether or not the terminal would be visible while executing the program. Obviously, it is undesirable for the terminal to be viewed in this instance, to prevent the target from being aware of its execution. Once the file is generated, it is automatically added to an Output folder which makes it easy to identify. Once this was completed, the first test was able to be run.

Running the executable was initially unsuccessful, resulting from an error in the code, whereby an imported module had been mistakenly left. To resolve this, it was swiftly removed, which was an easy fix due to an error made by neglecting the imports as they had been made fairly early in development. Nonetheless, after fixing the script and generating a new executable, it was then tested again which resulted in greater success. The keylogger had been successful in generating a .txt file, which allowed the keys to be logged for the test designated time of ’60 seconds’. However, the email containing this attachment and the network information failed to be sent, thus requiring further testing and investigation.

After some research, it was established that in May 2022, Gmail changed its email policy, which wouldn’t allow third-party apps to log in and send or receive mail. This required thorough investigating, using both official Google scholar documents and informal resources such as ‘Reddit’ and ‘Quora’. Ultimately, the solution was identified on YouTube, which detailed a video which allowed third-party apps to have passwords generated for each different app as long as two-factor authentication was enabled. Once this was done, a single use, 16-digit password could be generated and substituted into the code for the actual password. Assuming this had fixed the issue, the code was run inside the Visual Studios IDE. However, an error was returned within the function that controlled the email attachments and sending of the email. Testing the code in the IDE again, returned a new error further on in the code, stating ‘tuple object has no attribute “lower”. This essentially means that the tuple object was trying to call the “lower()” method, however, tuples do not have this method access because they are immutable, and you cannot edit their methods. Furthermore, this error was in the ‘sendmail’ method, within the ‘smtplib.SMTP’. This error occurred because the method was called with four arguments, instead of the correct three: the recipient’s email address, the sender’s address and finally the email message, with any attachments.

Graphical user interface, application, Teams

Description automatically generated

Figure 7: the one-time use ‘app password’ generated on Gmail.

Graphical user interface, text

Description automatically generated

The steps taken were to move the code that creates the email address and attachment and put it into a new function which handles the email address of the sender, recipient, and the file data as an argument. Then, returns a tuple which contains the message information of the email, including the attachment. Following this, code was added which called the new ‘create\_email\_message’ function from the ‘email\_file’ function, which enables the creation of the email message and the attachments. Furthermore, the method containing the code to pass the tuple was modified to pass the correct three arguments. The code was once again tested within the IDE, resulting in the code being successfully executed, and an email containing all the network information, followed by the Keystroke attachment was received from the tester’s Gmail account. Following the success of the prototype, the code can be developed further.

Graphical user interface, text, application

Description automatically generated

Figure 8: The result of a successful test of the prototype.

Table

Description automatically generated with low confidence

Figure 9: The output of the prototype keylogging file.

Graphical user interface

Description automatically generated

Figure 10: A folder showing multiple tested executable prototypes.

The next feature to be developed was the component which allows clipboard information to be added to the email body. As well as this, features which added microphone audio and webcam recordings were due to be implemented next. Starting development, the aim of the clipboard feature was to feature it in a separate text document, similar to the keystroke.txt document that the program outputs. However, during development, it was decided it would be more time efficient to add the clipboard information into the body of the email, which was relatively simple. The development commenced with changing the email output, meaning that the body was changed; the clipboard information was included in the email. Secondly, it was decided to use the ‘win32clipboard’ module. This is the same as ‘pyperclip’ which was originally planned to be used, but further research highlighted that ‘win32clipboard’ was more widely researched, and utilised within the industry for similar projects (Steve Whims, 2022). The method ‘win32Clipboard.OpenClipbard’ was employed, which reads the current text stored on the clipboard, and can paste it into the desired location, such as the email body.

Graphical user interface, text

Description automatically generated

Figure 11: A snippet showing how the clipboard data is acquired.

Text

Description automatically generated

Figure 12: A snippet showing the updated and final email body.

Furthermore, during this development, the decision was made to add as many features as possible, test them all at once, and manage the errors encountered as they occur. With this in mind, the next task was to implement the feature which took screenshots of the device every 5 seconds and emailed this as an attachment. The library ‘pyautogui’ was used, which can be used to take screenshots and control the mouse and keyboard too. This method was used to capture a screenshot of the user’s screen, this allows the image to be returned as a ‘Pillow’ image object. This is then saved as an image called ‘screenshot.png’ using the method named ‘screenshot.save.(screenshot\_path)’. After the screenshot is saved, the file is opened in binary, similarly to the method used earlier on in the process using ‘open()’ function. The contents are then read in a variable named ‘screenshot\_data’. This is then attached to the email using the same method as the other attachments; using the ‘MIMEBase’ and ‘Encoders’ classes which are included inside the ‘email’ library. Similarly, the other attachments are added to the email using the same method used previously, the ‘message.attach()’ method.

Text

Description automatically generated

Figure 13: A snippet showing the code that utilises the MIME module which attaches things to emails.

Graphical user interface, text

Description automatically generated

Figure 14: The second timer which controls the frequency of requested data.

The microphone recording feature was implanted next, which features a customizable integer, allowing for the duration and frequency to be changed, catering to the user’s different and unique requirements depending on the situation. This, however, was one of the more challenging features that was implemented, many things have to be added into the code. This includes ‘CHUNK’, which defines the number of audio samples per buffer, the format of the audio, the duration, the output name and finally the number of audio channels being used in the recording. The function ‘record\_audio()’ uses the above parameters to start the audio stream using ‘PyAudio’s’ libraries and included methods. Breaking that down, the method then reads the audio sample, that runs for a designated number of seconds, previously defined in the above parameters. Once all parameters have been satisfied, the loop is then closed using ‘stop\_stream()’ and ‘close()’. Furthermore, the audio files are then written as a WAV file using the ‘wave’ module previously installed. This provides a way for the functions to read and write into the WAV files. Once the recording has been completed and saved to a file, it is then attached to an email using the same method as the keystroke logs. Thus, reading the contents of the file encodes it as a base64, and finally, attaches it to a MIME message within the email read to be sent to the designated address.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 15: A snippet that shows the function to grab microphone audio.

Following the developmental success of the audio, it was decided that the webcam feature would next be implemented. This is intended to record a webcam recording without audio, for a designated amount of time. This can be combined with the audio to make a complete video. The code is designed around the ‘cv2’ library, which accesses the user’s webcam and captures video frames. The function ‘record webcam()’ function starts the recording process. The function starts a loop which captures 15 frames, each one second apart, this is done by using the ‘cap.read()’ method. The frames are then concatenated horizontally to merge together to make a single image using the ‘np.concatrate’ method. The resulting file is then saved using the ‘cv2.imwirte’ method and attached to the existing body using the ‘email\_file()’ function, which has been used throughout the process.

Text

Description automatically generated

Figure 16: A snippet which shows how the microphone and audio is attached.

With the addition of these features, the code was ready to be tested. This was conducted using the same method as originally chosen. Firstly, the code encountered some User error issues, which resulted due to either spelling issues or misplacing a colon. However, once all minor bugs had been ironed out, the code was run and tested again within the Visual Studios IDE. Subsequently, a few seconds after the code had been run, a light lit up next to the webcam on the machine the test has been conducted on, indicating the test had been successful. After the time had expired on the timer, which had been input to a short 20 seconds to run tests efficiently, an email was received. This contained all the correct files, including, an mp4 file of webcam footage, audio recordings, clipboard information and Keystrokes logged. This indicated the short testing time had been successful, and the progression to adding encryption processes into the code could now occur.

Graphical user interface, application, Word

Description automatically generated

Figure 17: A screenshot that shows a successful test.

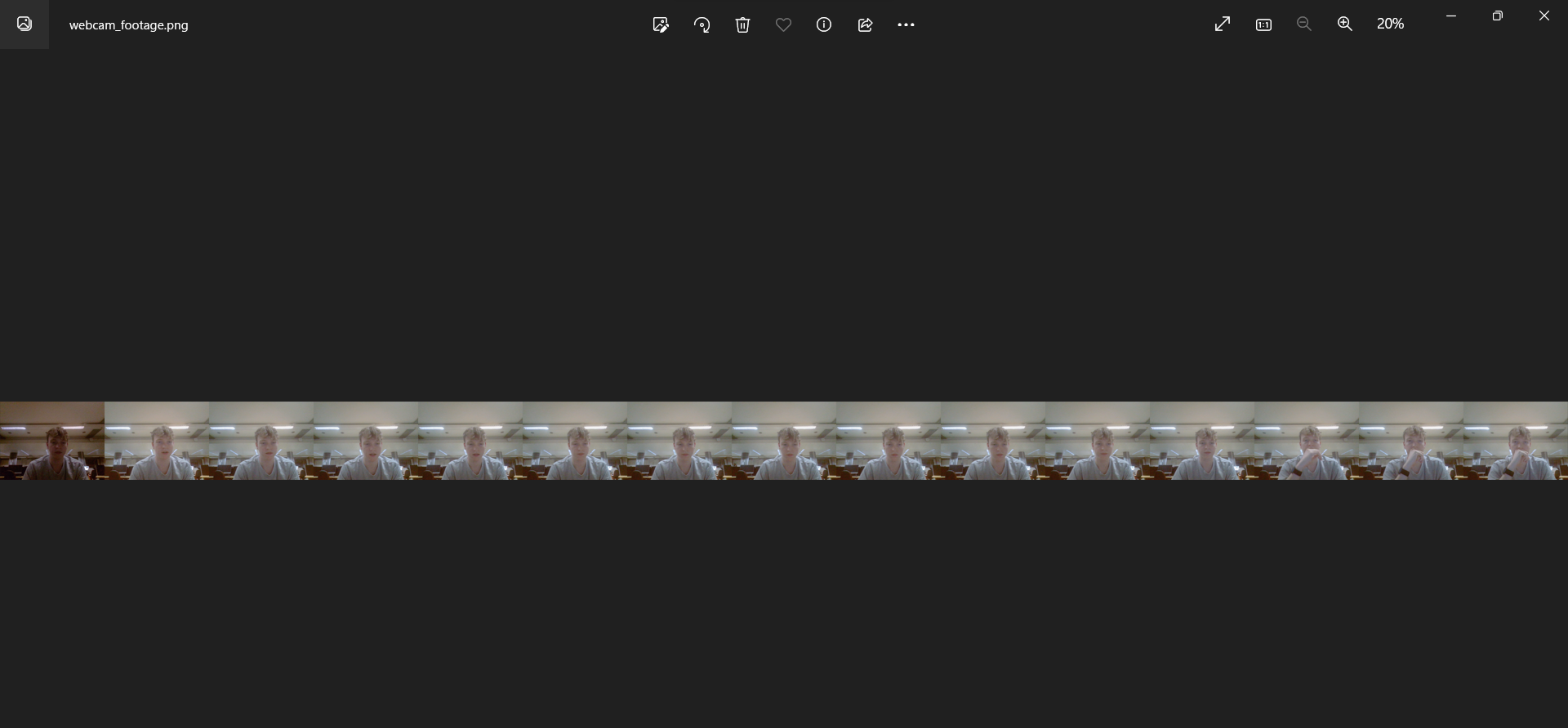


Figure 18: A screenshot of the 15-frame result of the webcam footage.

## **Encryption**

Following the successful testing of the code with the desired features included the next step was to start configuring encryption. It was evident that AES encryption would be most easily integrated into Python, and due to previous familiarity with this type of encryption, it was consequently the chosen technique (Ibarrondo and Viand, 2021). It was decided that only the keystroke’s where to be encrypted, as that document could contain the most sensitive type of information from the target’s system. The process includes using either a self-generated or randomly generated secret key, either 128,192 or 256 bits. While using a randomly generated key would obtain greater security, it was decided for the purpose of small-scale testing, as well as not possessing the ability and knowledge to create a program that decrypts a random secret key. The process of integrating the encryption was fairly straightforward; completed by first importing the necessary ‘cryptography’ libraries. The intended encrypted file is passed through the variable ‘data’ in binary mode, which is followed by the ’data’ and ‘key’ arguments. This results in the data being stored in a new document which is rightly titled ‘keystrokes\_encrpyted.bin’ using the ‘wb’ mode in the open() function. The encrypted file is then substituted in place of the unencrypted file in the same method as the previous keystroke log.

Text

Description automatically generated

Figure 19: A snippet of the code required to implement encryption.

Subsequently, a decryption file is to be created to enable it to run when the program receives the encrypted file. This is completed by developing a simple 20 script, which includes the cypher that combined the cypher, the key and the IV (initialization vector). This creates a new cypher object, called ‘AES,MODE\_CBC’.This tells the program to enter cypher block chaining mode, this combined the ‘cipher\_decrypt’ method which gives the code enough instructions to decrypt the cypher. Finally, the code will instruct the program to save the output on the Desktop, and call the file ‘keystrokes\_decrypted.txt’. This provides both the decryption program and file that needs decrypting in the same folder, meaning it can be run using the terminal or cmd with the command ‘python’ followed by the name of the program.

**Text

Description automatically generated**

Figure 20: a snippet showing the decryption file.

**Graphical user interface, application

Description automatically generated**

Figure 21: A image showing a successfully encrypted keystroke file.

**Text

Description automatically generated with medium confidence**

Figure : An image showing a successfully decrypted keystroke file.

## **Disguising the Software**

There were several identified approaches which were planned to take place at this stage of the project disguising the software. Despite this, it was established that the utilisation of a PDF or HTML link was too time-consuming, and the research required to learn the skills would outweigh the benefits of the final product. Alternatively, a program called ‘Resource Hacker’ was opted for, defined as a bit of freeware that allows the user to change the icons of the programme, the first in the two-step process to disguise the program as a substitute program (Johnson, 2019). The process which was followed by ‘Resource Hacker’ was first, to save an image of a sales report icon from Google, which can replace the icon of the recently converted ‘exe’ program. Once the icon is downloaded, it needs to be converted to an ICO file and added as a resource within the ‘Resource Hacker’ GUI. This is an easy process and was quickly completed, allowing for the next stage to commence: altering the name of the file using the ‘unitrix’ method. This refers to a method named after the unitrix virus, given its name for its ability to trick victims into believing it wasn’t an executable (Bailey, 2021). This used the Right-to-left-Override character, which changes the way the name of the file looks to the user, misleading them to believe it’s a .jpg. This being done, the executable was finally ready to be tested in the disguised state. This was conducted by simply clicking and running the program, which avoided all anti-virus and successfully returned the information intended by the Keylogger software.

Graphical user interface, text, application, email

Description automatically generated

Figure 23: an image showing the resource hacker GUI.

Text

Description automatically generated

Figure 24: The icon used to mask the executable.

Graphical user interface, text, application, email

Description automatically generated

Figure 25: An image showing the executable properties after it had been masked, proving it’s still an executable.

Following on from the success of disguising the spyware, the method chosen to get it onto the target machine was via a phishing email. This email could be written and changed depending on who the target was which could be deciphered using social engineering. While the current icon of the executable indicates a sales report, this could be easily changed and adapted to cater to the needs of the user of the program. This is one of the main reasons that Resource Hacker was used, due to its ease of use and how it can be quickly adapted and changed in any situation. A phishing email could be easily devised using persuasive language to encourage the target to view the file, or perhaps pretending to be a supervisor of the target, making them more inclined to click the file in disguise. Once the image is clicked, the executable would then run making the machine vulnerable to the program.

# **Critical Evaluation**

## **Product Evaluation**

The project can be considered an overwhelming success, evident as in the designated window of time, I was able to learn Python to a high level, with a thorough understanding of the project created. I have furthered my existing knowledge and honed the skills learned throughout my university experience, for example, by enhancing my cryptography understanding and learning an entirely new coding language. Each main feature worked exceptionally well, despite having to adapt and refine the original plan in order to gain effective functionality. This can be seen when the keystrokes were initially intended to be within an XML file, however, it was consequently decided that a text file would be easier to work with due to a greater understanding of this type of document. In addition, another example of modifications to the original plan was the technique employed for document concealment. For example, it was initially planned to be a PDF, but following research, a more time-efficient method involving ‘Resource Hacker’ was alternatively used, whilst retaining the original plan’s effectiveness. I am proud of the wealth of research conducted to enable the comprehension of a new coding language in such a limited and demanding time period, through hours of trial and error faced throughout the project. I feel particularly accomplished with the features that enabled webcam and microphone recordings, as I recognised these to be difficult due to their high technicality and viewed them as potentially unobtainable at the very beginning of development. I would have felt fulfilled if I had tried and failed to implement them, let alone understand each component, and actually get them functioning, which explicitly highlights my knowledge and skill progression.

It must be acknowledged that the only feature that failed to be implemented was the passwords saved on a keychain. This is justifiable, due to the nature of the feature, I struggled to find educational material about it online. Although I dedicated a wealth of time and effort toward further research to address this failure, it was proving to be extremely intrusive and lacked evidence of it being done. This unfortunately led me to have to abandon it, but it must be noted this allowed my effort and research time to be committed to other areas of my project, including highly technical areas such as webcam and microphone recording feature implementation. Nevertheless, if I had a greater duration of time to complete this project, this is a feature I would aim to explore further, as it would make having this system to assist in penetration testing extremely useful. Another feature I would add if more time were available, is a GUI. This could perhaps upgrade the project to another level, adopting an application-style approach rather than having information fed back via email.

## **Project Evaluation**

The systematic approach I undertook throughout the project allowed for my focus to be sustained for long periods of time, whilst simultaneously maintaining a strong understanding of the project, resulting in its overall effectiveness. This was carried out by splitting the project up into manageable sub-sections, based on what I considered to be easiest to hardest, excluding the encryption and disguising the file. This commenced with following a wide variety of YouTube videos and researching via GitHub; Reddit; and other coding communities and forums, to develop the basic keylogger. This was considered a foundational element, which could be added to and further refined as I progressed in the code. I believe this technique was highly beneficial, as it aided me in focusing on one element at a time, before progressing further into more complex aspects of the project. Once a section of the project had been completed, I would test it by running the code within an IDE. Once the section’s success was established, I would document what I had done, and justify why I had done it. Research methods involved using informal sources, such as community forums or YouTube videos. These were chosen due to the use of ‘lay terms’ which I found were more specific to the needs of my project, simply explained in comparison to Python or coding books. These sources were also easier to access, and it was much easier to identify necessary information within a smaller more-tailored body of information.

Overall, the project and methods used to acquire this information can be considered a success, due to reaching my academic objectives previously stated. These being, to obtain a thorough understanding of Python, and develop a final product I am proud of, despite adaptations throughout. I concluded with a comprehensive understanding of this piece of spyware, specifically how it could be used in order to gain access to an unauthorised system, and its effectiveness when executed correctly. Nonetheless, if I were to undertake this project again, I would aim to improve my time management, to better allocate the workload over a longer period. It must be noted I consistently worked to the best of my ability, in spite of alternative assignments and differing workloads throughout the year. Another aspect I would choose to improve upon is the further implementation of additional features, including the originally intended passwords which are stored on a keychain. This however, it would take a lot more research, which is sparsely available due to its sensitive nature, as well as a GUI. I would also aim to spend a greater duration formatting the layout of the email which includes all the information, in order to increase its professional appearance. This being said, it does include all relevant information required to be a successful piece of spyware.

## **Evaluation of Tools**

### **Visual Studios Code**

VS code is one of the most popular graphical IDEs to generate code and create projects, it’s lightweight and fast, meaning I was able to run it on my personal machine with no errors or encountered problems. Furthermore, it’s also cross-platform, meaning it was easier to copy the code over to a VM, which runs Linux to test the code if it required testing in a different environment. In addition, there’s a vast extension marketplace, which has plugins to enhance the coding experience. Following on from this, it’s also Git integrated, which makes it simple to add and edit Git repositories. This permitted a seamless experience with VS code, meaning again no problems were encountered. I believe the project wouldn’t have gone as smoothly without the great IDE. The only drawback which should be acknowledged, is that when multiple scripts are open at one time, it can use many of system resources, simply because the options are so vast within the program. However, this is easily managed with some simple file management.

### **GitHub**

GitHub is a free resource which allows coders of all abilities to share projects they’ve completed and ask questions that they have to other more experienced coders of all abilities. I consistently used it to resolve errors; compare my work to what other developers conducting similar projects had done; as well as inspect other keyloggers, to consider how my features compare to the ones completed by more experienced coders. This allowed me to set realistic aims for my project, whilst gaining inspiration during the starting phases of the code. 

### **Reddit & Stack Overflow/ Coding Communities**

These are all community-driven resources, which without, I wouldn’t have learnt any of the language or have the ability to complete or understand the code. These communities display videos or questions relating to all things programming, with endless choices of errors or questions that people have posted, with many knowledgeable individuals who regularly help others with questions. Personally, these resources taught me more than any academic sources I found, because I found the informal style of the tutorial easier to follow and understand.

### **Resource Hacker**

This is a freeware that allows the user to edit Windows executables, enabling editing of the icon of the file, the menu, the dialogues and even the bitmaps. It includes a user-friendly GUI, that allows easy navigation and implementation of different resources. If it weren’t for this software giving me an opportunity to hide the .exe file, I would have had to commit the more essential time to develop and editing code, as well as learning the process of creating more scripts in Java, and how to embed the file into another document.

### **Libraries**

Throughout the project, many libraries were added and built to implement different features and functions in the software. One of the first to be added was the smtplib, which was used to enable the simple mail transfer protocol, which allowed me to let Python send and access mail autonomically. Furthermore, the network data and MAC address were grabbed using the socket and UUID libraries, which enables me to attain the important information, making this software a useful penetration testing assistant. Possibly one of the most crucial libraries imported at the ‘MIME’ libraries, which enables the creating and sending of email messages, which was utilised for every one of the attachments that were sent out via email. The core functions of a keylogger are to log and listen to keys, for this, the Listener library and the pynput.keyboard libraries were employed. These allowed me to monitor and capture keystrokes and write them in a data file, which would eventually be sent to the MIME library. Various other libraries were used to gather recordings or files from the target machine, such as cv2, numpy, pyaudio and OS, which allow interactions with the operating system. Finally, the Crypto.ciper.AES and secrets are included, both used in the encryption process. The secrets module was designed to generate a random key, however, it was decided that I would use a pre-made key, as I didn’t have the ability to create a program to decrypt a random key.

### **Problems Faced**

Luckily, there were limited problems encountered during the development of the project, the only constraint I faced was due to limited time and management of my resources. This was attributed to having to balance other projects and commitments alongside this. Despite this, as previously stated, this project always aimed to maintain priority. Another issue that couldn’t have been avoided was my original lack of knowledge of Python, which required a lot of time-consuming research. Thankfully, apart from minor technical difficulties, which resulted in Visual Studio Code having to be reinstalled, I confronted no other issues in the project that could not be managed by testing and research.

### **Product Appraisal**

Research on other Keyloggers found online, formed a baseline of objectives and goals that were originally established in the product design phase in the first deliverable. These were what I considered to be achievable given my knowledge before to commencing this project. From this, it could be argued that my project, not only met the original requirements but also adapted to meet the objectives, despite encountering adversities along the way. This is evident, for example, when changes in the disguising of the keylogger to a different method occurred. The fact that the prototype only improved with each generation, was extremely encouraging to see with my knowledge prior, and I believe that given more time I would have been able to exceed these goals and add additional features to improve the market, such as a functioning GUI.

# **Conclusion**

Keyloggers are not typically renowned pieces of Malware associated with penetration testing, more like an intrusive piece of Spyware, whereby individuals aren’t aware it's affecting them. However, I believe with the proposed features, it can truly be employed as a piece of Spyware to assist in Penetration testing. This is attributed to its intrusive features, which grab information off the machine and can also be customisable for user requirements. The natural progression of technology consistently locates new ways to improve systems, as well as the level of security and anti-viruses that protect these systems. Despite this, with the dedication to further research, this system can be adapted to become a more familiar system for Penetration Testers. Although this project didn’t meet the complete project requirements initially established, simply due to educational materials lacking ease of access; with a greater duration of time and commitment to understanding, a potentially superior system could be developed than originally planned. This could contribute to the launch of a new and effective go-to tool for penetration testers who obtain minimal system knowledge and are eager to discover access to a system.

# **References**

Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2017). *Agile Software Development Methods: Review and Analysis*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1709.08439.

Bailey, R. (2021). *Win32:Unitrix*. [online] How To Fix Guide. Available at: https://howtofix.guide/win32unitrix/ [Accessed 3 May 2023].

Bassil, Y. (2012). *A Simulation Model for the Waterfall Software Development Life Cycle*. [online] arXiv.org. Available at: https://arxiv.org/abs/1205.6904.

Foong, N.W. (2020). *How to Send an Email With Attachments in Python*. [online] Medium. Available at: https://betterprogramming.pub/how-to-send-an-email-with-attachments-in-python-abe3b957ecf3 [Accessed 24 Apr. 2023].

Heron, S. (2009). Advanced Encryption Standard (AES). *Network Security*, 2009(12), pp.8–12. doi:https://doi.org/10.1016/s1353-4858(10)70006-4.

Ibarrondo, A. and Viand, A. (2021). Pyfhel. *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. doi:https://doi.org/10.1145/3474366.3486923.

IBM (2022). *Cost of a Data Breach 2022*. [online] IBM. Available at: https://www.ibm.com/reports/data-breach.

Johnson, A. (2019). *Resource Hacker*. [online] www.angusj.com. Available at: http://www.angusj.com/resourcehacker/.

National Cyber Security Centre (2017). *Penetration Testing*. [online] Ncsc.gov.uk. Available at: https://www.ncsc.gov.uk/guidance/penetration-testing.

Python (2020). *The Python Standard Library — Python 3.8.1 documentation*. [online] Python.org. Available at: https://docs.python.org/3/library/.

Redscan (2019). *Tyes of penetration testing | black box, white box & grey box*. [online] Redscan. Available at: https://www.redscan.com/news/types-of-pen-testing-white-box-black-box-and-everything-in-between/.

Rhodes, B. and Goerzen, J. (2014). SMTP. *Foundations of Python Network Programming*, pp.241–258. doi:https://doi.org/10.1007/978-1-4302-5855-1\_13.

Runeson, P. (2006). A survey of unit testing practices. *IEEE Software*, 23(4), pp.22–29. doi:https://doi.org/10.1109/ms.2006.91.

Steve Whims (2022). *Using the Clipboard - Win32 apps*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/windows/win32/dataxchg/using-the-clipboard [Accessed 28 Apr. 2023].

Thomas, M. and Yorkshire, T. (2014). A Steganography-based Covert Keylogger. *Panagiotis Yialouris & Thomas Yorkshire International Journal of Computer Science and Security (IJCSS)*, [online] (8), p.177. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=47663ffbd92ea38b55ae1c4e26d373be6502e77f [Accessed 4 May 2023].

Vojinovic, I. (2019). *Data breach statistics that will make you think twice before filling out an online form*. [online] DataProt. Available at: https://dataprot.net/statistics/data-breach-statistics/.

Wynn, D.C. and Eckert, C.M. (2016). Perspectives on iteration in design and development. *Research in Engineering Design*, 28(2), pp.153–184. doi:https://doi.org/10.1007/s00163-016-0226-3.