SA Project - Keylogger

Phoebe Tandjiria (z5444555)

University of New South Wales (UNSW)

COMP6841 Extended Security Engineering and Cyber Security

Code: https://github.com/ptandjiria/COMP6841_project/tree/main

Demo of user side vs attacker side: https://youtu.be/5TrRLOZ_I-Y

Abstract

Input devices such as keyboards and mice are susceptible to attacks because they provide information to computers. Keyloggers, whether software or hardware, are a growing issue because they can stealthily record sensitive data such as passwords and payment information. The goal of this report is to analyse how a keylogger can exploit a person's keystrokes by building a keylogger in Python which includes features such as recording keystrokes, and interpreting keystrokes to construct meaningful messages. The keylogged data is then emailed to the attacker. Another goal of this report is to understand how one can then protect their systems from such attacks, and highlight the threat keyloggers provide to user authentication, confidentiality and integrity. This report covers the entire process of creating a keylogger, including functionality, distribution, stealthing techniques and remote communication back to the attacker.

Table of Contents

Introduction

In today's increasingly digital world, cybersecurity is paramount as both individuals and organisations rely heavily on technology for storing sensitive information, communicating, and conducting business. Among the many threats to data security, keyloggers represent a particularly dangerous form of malware that can silently record users' keystrokes, capturing sensitive information such as passwords, credit card details, and personal messages. Understanding how keyloggers operate is crucial for developing defences against them.

This project focused on the technical construction and functionality of keyloggers to gain insight into their inner workings and to better understand the strategies used by attackers. By building a basic keylogger in Python, I explored several components of malware creation: keystroke logging, remote communication, and stealth. Throughout the process, I remained mindful of ethical considerations, using the project solely as a learning exercise to reinforce the importance of cybersecurity and to shed light on the preventive measures required to protect against such threats.

By analysing how keyloggers work and how their use can be prevented, I aim to use this knowledge responsibly and ethically to suggest various cybersecurity defences.

Existing keylogging technologies

Types of keyloggers

There are two types of keylogging technologies – hardware and software keyloggers. While hardware keyloggers are devices disguised in USB ports, computer cabling or keyboard circuit, software keyloggers are malware downloaded onto devices unknowingly. For this reason, software keyloggers are mainly used in cybersecurity attacks instead of their hardware equivalents, and they remain to be a major security threat because they do not require physical devices to gain access (Lenaerts-Bergmans, 2023). As such, I chose to focus my project on software keyloggers in particular.

From my research, I learned that software keyloggers come in various forms. The form-grabbing keylogger is deployed on a website rather than installed and aims to gain credentials from data inputted into fields. API keyloggers exist within applications and records events whenever a key is pressed while the application is running. JavaScript keyloggers are injected into websites and which record keystrokes entered by the website's users (Veracode, 2024).

How keyloggers are distributed

Apart from the injection of scripts into websites, these software keyloggers mainly are spread not because of technical hacking techniques, but due to forms of social engineering. For example, victims can click infected links or navigate to malicious websites, or open emails that contain the malware. Hackers will embed keyloggers into seemingly trustworthy websites or applications, and collect information in this way (Lenaerts-Bergmans, 2023). For this reason, this report will not only focus on the technical implementation of a keylogger, but will also extend on how keyloggers are spread and highlight how humans are the weakest link in any system.

Common stealth techniques

After installation, stealth techniques are used to evade detection of the keylogger by the user. These stealth techniques include running as a background service with a non-descript name, piggybacking off legitimate system processes (making them harder to detect with anti-virus software), or using scheduled tasks to activate the keylogger in a minimised or hidden state without user interaction . For my implementation, I chose to implement the keylogger as a background service with a non-descript name.

Implementation

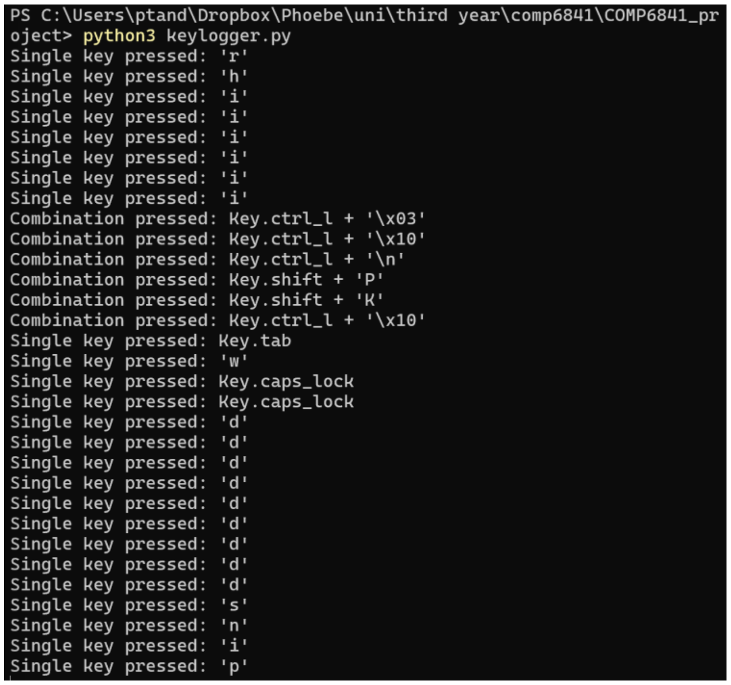
The use of Python

As mentioned before, the keylogger was to be implemented in Python. This was for a few practical reasons more so to do with the educational and security research component of this report. Python's syntax is simple and intuitive, and also easily tested for functionality without needing complex code. This is due to Python's extensive third-party libraries such as 'pynput' to record keystrokes, 'time' to record timestamps, and 'mss' to take screenshots at regular intervals. Python also makes it easier to focus on how the software works and how to develop better defenses, primarily because it is efficient, easy to use and readable. A Python keylogger is also able to be easily stealthed with libraries such as 'PyInstaller' or cs_Freeze', which compile the Python script as well as its dependencies into a standalone executable (.exe file), allowing it to run on systems that may not have Python installed (Geeks for Geeks, 2024). It is not the stealthiest choice compared to lower-level languages such as C or C++, but offers more convenience for learning purposes especially for this project.

The main keylogger program

Firstly, due to the intrusive nature of the keylogger, I added my program to exclusions listed under Virus & Threat Protection so that my Windows laptop would not treat my program as a threat or virus. In my main keylogger program, I chose to use the third-party library 'pynput' because it allows for control and monitoring of input devices including the keyboard. With 'pynput', I was able to listen to events to capture user input from the keyboard and print to the terminal/logged to a text file (keylogger.txt) the exact keys pressed when the program was running. These keys included all ASCII characters as well as the modifier keys (consisting of the Shift, Tab, Control, Caps Lock, Windows, and Alt keys). For ethical reasons and to not make this program too intrusive on my own computer, I created a mechanism to stop the running program – whenever the Escape key is pressed, the program will stop running. For a fully functional keylogger intended to keep running upon installation, this mechanism is not needed.

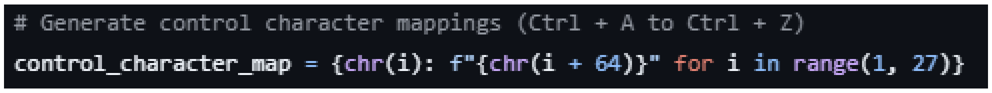
I wanted my program to not be limited to the logging of single key presses, and so I had to modify my program so that it would process double and triple key presses correctly. This was simply done by maintaining a list of modifier keys and checking if they are pressed. If pressed, the next key/s (key/s pressed in conjunction with the modifier key) must be processed also in the same press event. This successfully processed double and triple key presses together, but I noticed that when the control key was pressed, the key used in combination with it was logged in hexadecimal, as shown in Figure 1.



```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_pr
oject> python3 keylogger.py
Single key pressed: 'r'
Single key pressed: 'h'
Single key pressed: 'i'
Single key pressed: 'i'
Single key pressed: 'i'
Single key pressed: 'i'
Single key pressed: 'i'
Single key pressed: 'i'
Combination pressed: Key.ctrl_l + '\x03'
Combination pressed: Key.ctrl_l + '\x10'
Combination pressed: Key.ctrl_l + '\n'
Combination pressed: Key.shift + 'P'
Combination pressed: Key.shift + 'K'
Combination pressed: Key.ctrl_l + '\x10'
Single key pressed: Key.tab
Single key pressed: 'w'
Single key pressed: Key.caps_lock
Single key pressed: Key.caps_lock
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 'd'
Single key pressed: 's'
Single key pressed: 'n'
Single key pressed: 'i'
Single key pressed: 'p'
```

Figure 1: keys used in combination with Ctrl are shown in hexadecimal

Upon further research, I learned that characters pressed with the Ctrl key often translate into hexadecimal format because it is a convenient and efficient way for computers to represent and reference the character as binary data in its event handling and input systems. This matter was easily rectified with a simple mapping function which generates control characters mappings from their hexadecimal representations, as shown in Figure 2.

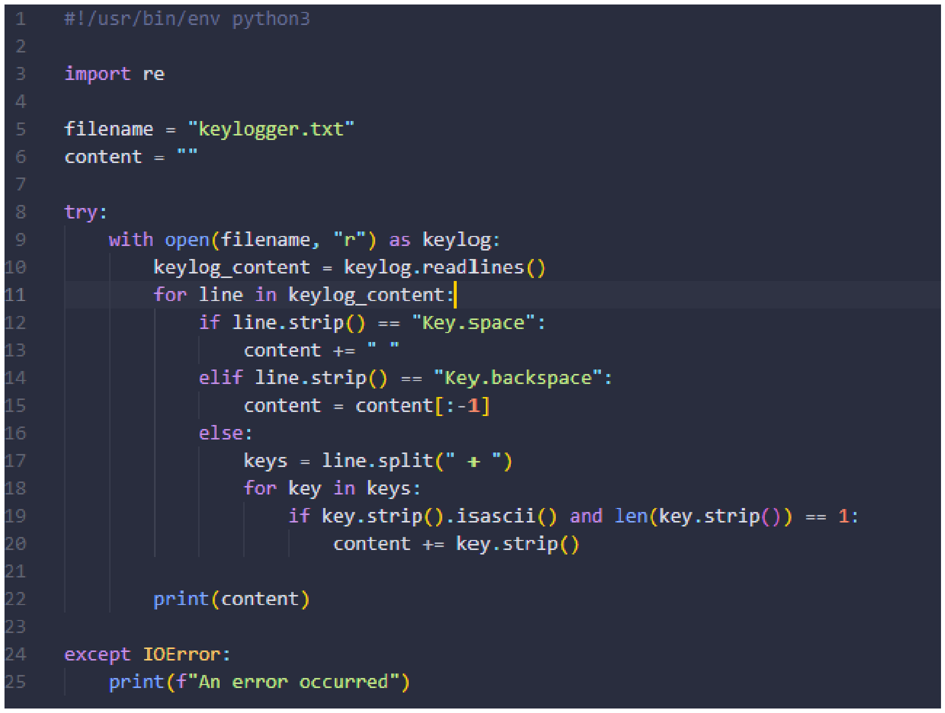


```python
# Generate control character mappings (Ctrl + A to Ctrl + Z)
control_character_map = {chr(i): f"{chr(i + 64)}" for i in range(1, 27)}
```

Figure 2: mapping function to convert hexadecimal into character format

Python script to interpret pressed keys

Having a log of simple keypresses was not enough – they would have to be concatenated together to form a meaningful message, phrase or sentence. I chose to write a simple python script to interpret my logged keypresses in keylogger.txt and print the message to the terminal. I ensured that the space and backspace keys were accounted for. This was a simple and easy task, and the code is shown in Figure 3.

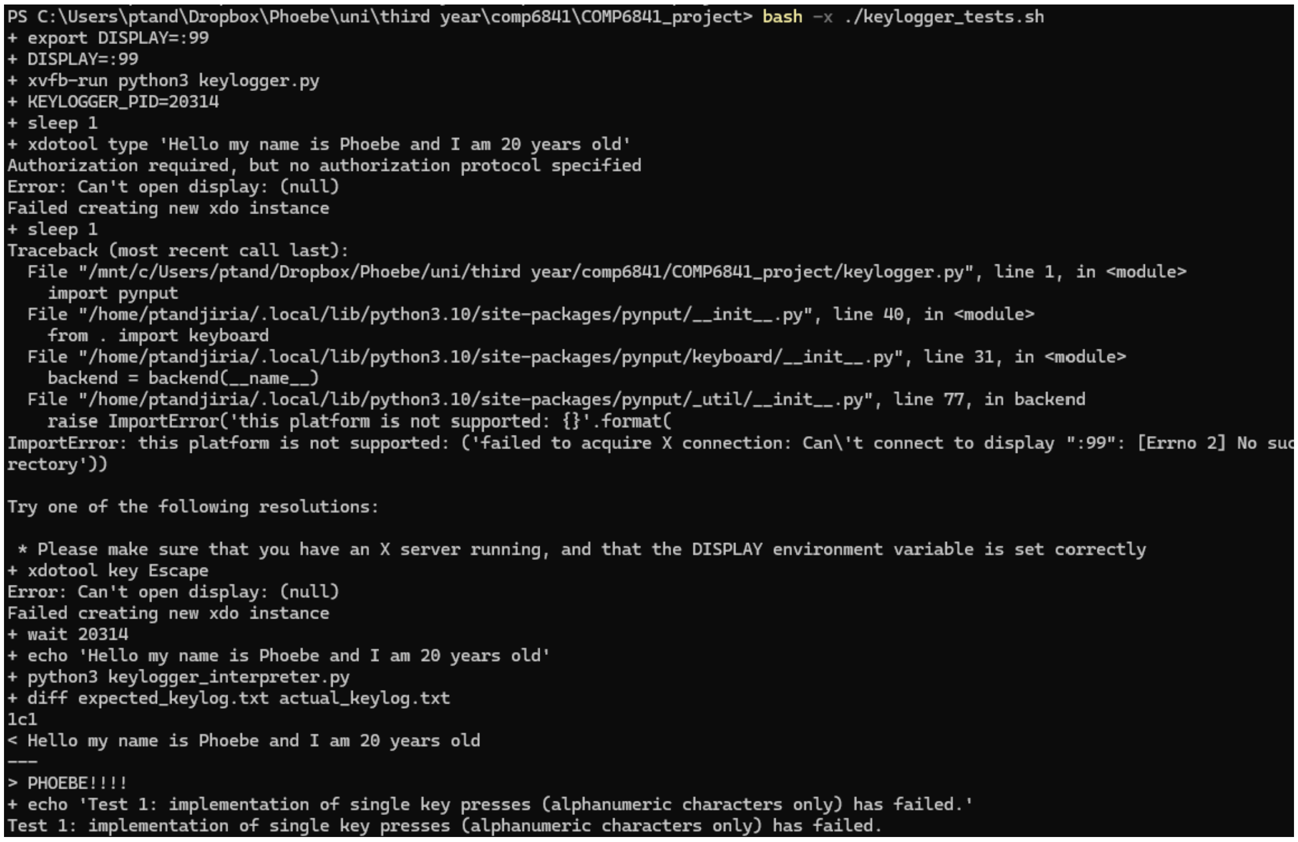


```python
#!/usr/bin/env python3

import re

filename = "keylogger.txt"
content = ""

try:
    with open(filename, "r") as keylog:
        keylog_content = keylog.readlines()
        for line in keylog_content:
            if line.strip() == "Key.space":
                content += " "
            elif line.strip() == "Key.backspace":
                content = content[:-1]
            else:
                keys = line.split(" + ")
                for key in keys:
                    if key.strip().isascii() and len(key.strip()) == 1:
                        content += key.strip()

        print(content)

except IOError:
    print(f"An error occurred")
```

Figure 3: keylogger_interpreter.py which interprets keypresses from keylogger.txt

Testing of the keylogger

I attempted writing a shell script to test my keylogger with various inputs, but I encountered the following errors to do with using 'pynput' and working in a non X-environment. I also found it difficult to simulate keypresses in shell. The shell script is accessible in keylogger_tests.sh.



```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> bash -x ./keylogger_tests.sh
+ export DISPLAY=:99
+ DISPLAY=:99
+ xvfb-run python3 keylogger.py
+ KEYLOGGER_PID=20314
+ sleep 1
+ xdotool type 'Hello my name is Phoebe and I am 20 years old'
Authorization required, but no authorization protocol specified
Error: Can't open display: (null)
Failed creating new xdo instance
+ sleep 1
Traceback (most recent call last):
  File "/mnt/c/Users/ptand/Dropbox/Phoebe/uni/third year/comp6841/COMP6841_project/keylogger.py", line 1, in <module>
    import pynput
  File "/home/ptandjiria/.local/lib/python3.10/site-packages/pynput/__init__.py", line 40, in <module>
    from . import keyboard
  File "/home/ptandjiria/.local/lib/python3.10/site-packages/pynput/keyboard/__init__.py", line 31, in <module>
    backend = backend(__name__)
  File "/home/ptandjiria/.local/lib/python3.10/site-packages/pynput/_util/__init__.py", line 77, in backend
    raise ImportError('this platform is not supported: {}'.format(
ImportError: this platform is not supported: ('failed to acquire X connection: Can\'t connect to display ":99": [Errno 2] No suc
rectory'))

Try one of the following resolutions:

 * Please make sure that you have an X server running, and that the DISPLAY environment variable is set correctly
+ xdotool key Escape
Error: Can't open display: (null)
Failed creating new xdo instance
+ wait 20314
+ echo 'Hello my name is Phoebe and I am 20 years old'
+ python3 keylogger_interpreter.py
+ diff expected_keylog.txt actual_keylog.txt
1c1
< Hello my name is Phoebe and I am 20 years old
---
> PHOEBE!!!!
+ echo 'Test 1: implementation of single key presses (alphanumeric characters only) has failed.'
Test 1: implementation of single key presses (alphanumeric characters only) has failed.
```

Figure 4: Error when attempting to use keylogger_tests.sh to test keylogger.py

As such, I chose to test my keylogger.py program manually, the results of which are shown in the table below.

Tests for keylogger.py

Test	Input	Motivation	Result	Supporting images
Test 1	Alphanumeric characters only + shift key	Ensures that single key presses work, as well as any shift combinations with alphanumeric characters.	Successful	Figure 5.1 in appendix
Test 2	Only ASCII characters with backspaces	Ensures that backspaces work as intended in the keylogger_interpreter.py file by removing the last character concatenated to the end of the 'meaningful message'.	Successful	Figures 5.2 and 5.3 in appendix
Test 3	Double key presses (shift key + non-alphanumeric characters)	Ensures that the shift key in combination with any non-alphanumeric characters will be processed correctly by keylogger_interpreter.py.	Successful	Figures 5.4 and 5.5 in appendix
Test 4	Double key presses (two modifiers at a time)	Ensures that any two modifiers pressed at the same time will be processed correctly by keylogger.py and outputted in the keylogger.txt file but will not affect the meaningful message created by keylogger_interpreter.py.	Successful	Figure 5.6 in appendix
Test 5	Triple key presses (three modifiers at a time)	Ensures that any three modifiers pressed at a time will be outputted by keylogger.py into keylogger.txt correctly but will not affect the message created by keylogger_interpreter.py.	Successful	Figure 5.7 in appendix

Injection and distribution of the keylogger

At this point, I had confidence in the functionality of the keylogger program as well as its interpreter program in constructing the correct message. The next and perhaps most important component of the keylogger was its injection and distribution to users. However, one of the project outcomes I had established for myself was to remain ethical throughout the project, and this included not distributing malware to non-consenting people. As such, this specific section of the report is largely hypothetical for ethical reasons and was not implemented by any means.

Some ideas which came to mind were inspired by existing keylogger technologies. This included injection into forms or websites and social engineering through phishing emails or convincing a user to click on malicious attachments/links. I was more interested in how social engineering and psychology could encourage the distribution of malware including keyloggers, so I chose to research this in depth to inform my own decision of how to better distribute the keylogger program.

Social engineering requires a victim that has less knowledge compared to the attacker. This allows for deception in order for the victim to reveal private information to unauthorized users, who can then gain access to a computer system or network (Hatfield, 2018).

There are variants of social engineering that could help distribute the keylogger to unknowing users. For example, impersonation as a trusted senior employee of a company’s IT department and asking for usernames and passwords due to a recent hack could fool employees into giving out their credentials (VICE Asia, 2019). This specifically takes advantage of the cognitive vulnerability of authority bias, in which people tend to have trust in authority figures. The mention of a recent hack into the network and system also creates a sense of urgency and time pressure, which can cause employees to act quickly without correctly identifying my identity.

Another variant of social engineering is using phishing emails to deceive the user into downloading the malware. This can be done by pretending to be a friend or co-worker, posing the email as a legitimate request for information, or posing the email as a security warning (Hatfield, 2018). Again, this approach targets the cognitive vulnerability of authority bias and can create a sense of urgency for the victim, disallowing them to correctly analyse the scam. Another version of this kind of attack is utilising pop-up windows which create a sense of urgency. This approach requires more sophistication since awareness of phishing attacks has grown generally as more people are educated in the workplace or by other means.

Out of the two variants of social engineering mentioned, the latter is most likely to be the most effective since it allows for wider distribution of the keylogger and works remotely. In the case of using pop-ups, I believe that a pop-up posing as a notification informing of a detected thread from a trusted operating system application (such as the Windows Defender Security application) could create a sense of urgency to download a supposed antivirus software which actually doubles as my keylogger program. An example of this is shown in Figure 6 in the appendix. In recent years, there has been such a scam which urges users not to turn off their computers and to urgently call a supposed tech support line, and this was used to convince PC users to share important payment information (Rimkiene, 2023).

Sending the log back to myself remotely (as the attacker)

Another important component of the keylogger was remote communication of the keylogs back to myself as the attacker. Some ideas I gained from my research included uploading the keylogs to a remote server (which involved having to configure a server to receive and store logs), writing logs to a file and uploading the file to a file sharing device (such as Dropbox, Google Drive, etc.) through their APIs, or socket programming a direct connection to my machine from the victim's machine (thus allowing for real-time data transmission (Seitz, 2014).

My first attempt at remote communication involved socket programming to establish a direct connection. I wrote a program named attacker_server.py which acted as a server socket to receive interpreted messages over a socket connection. This was to be used in conjunction with a client program inside keylogger.py, which means it would be open for connection while keylogger.py was running. Although this would allow for real-time data transmission, I realised that as an attacker, I would have to always leave my server socket open for connection to receive all logs.

As such, I chose to email myself the logs periodically every ten seconds. Although this wouldn't allow for exact real-time transmission, the periodic email-sending would allow near real-time transmission.

The implementation of emailing myself the logs included the importing of the 'smtplib' library (which allows for a connection to a mail server through Simple Mail Transfer Protocol). Since I aimed to attach keylogger.txt to the email, I also imported 'email.mime' to support this functionality. I then used these libraries to create a function 'send_email', which would send emails with keylogger.txt as an attachment. However, I wanted the log to be emailed every ten seconds. Since I had a listener which listened to keyboard events, I had to import the 'threading' library to support multiple threads occurring at a time within the program – this consisted of the listener thread and the email thread. This was functional, but there was a potential race condition in which the listener thread will write to keylogger.txt and the email thread will read keylogger.txt as well as clear it after the email is sent. This is elaborated upon later in this report.

Stealth techniques

I now had to consider how my Python program would run once installed as malware on a victim's computer. After some research of keylogger attacks over the course of history, I was inspired by the Grand Theft Auto V Modification Attack in 2015, where a mod for the video game was uploaded to a mod-sharing website. This mod consisted of a hidden keylogger which was receiving and sending information such as what users were typing, and it was later discovered that a file named Fade.exe was responsible for tracking such activity (Salas-Nino, Ritter, Hamdan, Wang, & Hou, 2023). As such, I chose to package my Python program as a standalone executable file (.exe) which would allow the program to run on all operating systems which may not have Python installed. The 'pyinstaller' Python library allowed for the packaging of the program into an .exe file, and allowed for options through configurations such as '--noconsole' (no visible console on startup) and '—onefile' (packaging of the program into one file).

The next goal was to have the executable run as a background service on startup of the PC. A known way to start an application on startup of the PC is to use Task Scheduler. After creating a new task, one can set the trigger to occur at startup, then add an action such as 'Start a program' while specifying the file location of the application. However, this step needs to be done on individual PCs after installation of the .exe file. Therefore, successful configuration of application startup of the keylogger requires social engineering once again. Piggybacking off of the pop-up phishing attack, perhaps users can be prompted to follow exact installation instructions of the 'antivirus' software, and this includes carefully configuring the 'antivirus' to always run on startup of the PC so that their computer is 'well protected against the attack' (in actuality, they are allowing the keylogger to run on startup instead).

Synthesis of all components of the keylogger program

At this point, the establishment of what is used on the user side and attacker side is needed, utilising all the components of the keylogger program previously mentioned.

On the attacker side, some social engineering is needed to create a phishing attack involving a pop-up pretending to come from a trusted Windows application such as Windows Defender Security System (Rimkiene, 2023). This popup would include a sense of urgency by informing the user of an attack and infection of malware, and that an installation of the latest antivirus software is needed.

Instructions would be provided to allow the 'antivirus' to run on startup of the PC. In actuality, this software is the keylogger.py program as an .exe file. The keylogger is therefore distributed and run covertly on victims' PCs.

On the user side, they unknowingly have the keylogger program always running, and their keystrokes are emailed in a keylogger.txt attachment to the attacker every ten seconds. The attacker can then make sense of the emailed keylogger.txt attachment using keylogger_interpreter.py, and the timestamps in the emails allow for near real-time detection of keystrokes.

A short demonstration of both sides of the attack is found on the cover page of this report.

Strengths and weaknesses of the approach

Strengths

The approach in the implementation of the keylogger program has a few strengths, mainly in that the functionality of the keylogger program is successful, the packaging of the Python program into a .exe file allows it to be executable as an application on PCs that do not have Python installed, and the logs are able to be remotely sent to the attacker via email. An additional Python interpreter script also allows the attacker to construct meaningful messages from individual keystrokes.

Weaknesses

Although the keylogger program was able to email the logs back to the attacker, thereby providing a nearly accurate timestamp of the keystrokes made, this is useless without context of where and why the keystrokes were made. Without any knowledge of what websites or applications the keystrokes were pressed, it would be futile to use these keystrokes as an attacker. One improvement which could be made to provide more context on keystrokes are screenshots of the PC screen at regular intervals. With the 'mss' Python library, screen images can be captured, and these images can then be sent in combination with the keystroke logs by email to the attacker every ten seconds. This allows the attacker to see the exact website or application the victim is on at the time of each keystroke (therefore allowing for interpretation of sensitive data such as usernames and passwords on specific applications).

Another problem arisen from the implementation of the keylogger is the race condition between the listener thread and the email thread in keylogger.py. Both threads run at the same time, but the listener thread writes keystrokes to keylogger.txt while the email thread reads keystrokes from the same file to email back to the attacker then completely clears keylogger.txt so that repetition of keystrokes does not occur. This creates a race condition. A suggestion to prevent the race condition is to introduce a lock from the 'threading' library. A lock object can be created using "lock = threading.Lock()" and critical code (mainly the code writing to the file as well as the code clearing the file) can be wrapped with the "lock.acquire" and "lock.release" methods. This ensures that only one thread at a time is able to access and modify the keylogger.txt file, therefore preventing conflicts between the two threads and preventing loss of keystrokes in the log.

Conclusion and recommendations

In conclusion, this report demonstrates the capabilities of a Python-based keylogger, from capturing keystrokes and interpreting user inputs to remotely sending data back to the attacker. Keylogging technology, whether implemented as hardware or software, poses a substantial risk to information security by compromising user authentication, confidentiality, and data integrity. Through careful planning, social engineering, and stealth techniques, attackers can effectively use keyloggers to obtain sensitive data without users' knowledge.

The following recommendations outline methods to protect against attacks and mitigate associated risks:

1. Enhanced user education: Organisations should invest in cybersecurity training, specifically focused on social engineering tactics like phishing and other deceptive practices that attackers commonly use to distribute keyloggers.
2. Use of anti-malware software: PC users should use reputable anti-malware tools that actively scan for suspicious processes and prevent unauthorised applications from running in the background. Regularly updating this software is crucial to ensure protection against new keylogging methods.
3. Awareness of processes running on the PC: PC users should be educated on the processes running on their PC so that they are able to identify unknown applications and kill the process. Seeing running applications is as easy as opening the Task Manager application and seeing what processes are running there.
4. Two-factor authentication: Adding an extra layer of security, such as 2FA, helps protect sensitive accounts, even if keylogging captures passwords. This measure can reduce the impact of keyloggers on authentication-based attacks.
5. Awareness of suspicious emails, links and websites: Educate users on identifying phishing emails, unexpected attachments, or unfamiliar links. Attackers often rely on unsuspecting individuals to click on these links to trigger the download and execution of keyloggers.

Reflection

Throughout this project, I've gained a deep understanding of both the technical and ethical aspects of keyloggers, distribution of such malware, and cybersecurity. Creating a functional keylogger required more than just programming skills; it required careful consideration of each step, from keystroke logging to the distribution and stealth aspects of malware. I gained insight into how attackers exploit seemingly small loopholes in system security and human vulnerabilities, emphasising the vital role of cybersecurity measures in preventing such threats.

The technical aspect of building a keylogger taught me a lot about input device handling. For instance, working with ‘pynput’ to capture keystrokes showed me how input can be intercepted directly by accessible software, which is both powerful and concerning when thinking of its potential misuse. By implementing support for various keypress combinations, such as multi-modifier keypresses, I was able to appreciate the complexity of handling raw input data and converting it into coherent information.

Another important lesson was understanding the significance of thread management. Integrating threading allowed for background processes such as handling a listener and emailing attachments, which is a critical concept not only for malware but for any real-world application where multiple operations need to run concurrently. It also introduced me to potential pitfalls like race conditions.

Additionally, exploring stealth and distribution techniques helped me understand how attackers hide malicious software in plain sight. This aspect of the project made it clear how easily a system could be compromised through social engineering tactics and how attackers rely on human vulnerabilities just as much as technical ones. For example, learning about methods like disguising malware as legitimate software and using phishing techniques has highlighted the critical importance of cybersecurity awareness among users, as technical defences alone aren’t always sufficient.

Ethically, working on this project has been an eye-opener. I came to understand the ethical responsibility that comes with cybersecurity knowledge. While understanding how keyloggers work is essential for defending against them, it’s also clear how this knowledge could be misused. As such, I made sure to approach this project with an educational focus, aiming to build awareness of the risks and effective prevention strategies rather than exploitation. This experience has furthered my appreciation for ethical considerations in cybersecurity, as well as highlighted the need for security professionals to act responsibly with such knowledge.

Overall, this project has provided me with a well-rounded view of cybersecurity—from the technical construction of malware to the psychological manipulation involved in social engineering, and finally, to the ethical responsibilities of those working in cybersecurity.

References

Geeks for Geeks. (2024, January 29). *Executable File Format | .exe Extension*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/exe-program-format/

Hatfield, J. M. (2018). Social engineering in cybersecurity: The evolution of a concept. *Computers & Security*, 102-113.

Lenaerts-Bergmans, B. (2023, February 2). *Keyloggers: How They Work and How to Detect Them*. Retrieved from Crowdstrike: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/keylogger/

Rimkiene, R. (2023, November 15). *How to remove Windows Defender security warning scam*. Retrieved from Cybernews: https://cybernews.com/malware/remove-windows-defender-security-warning/

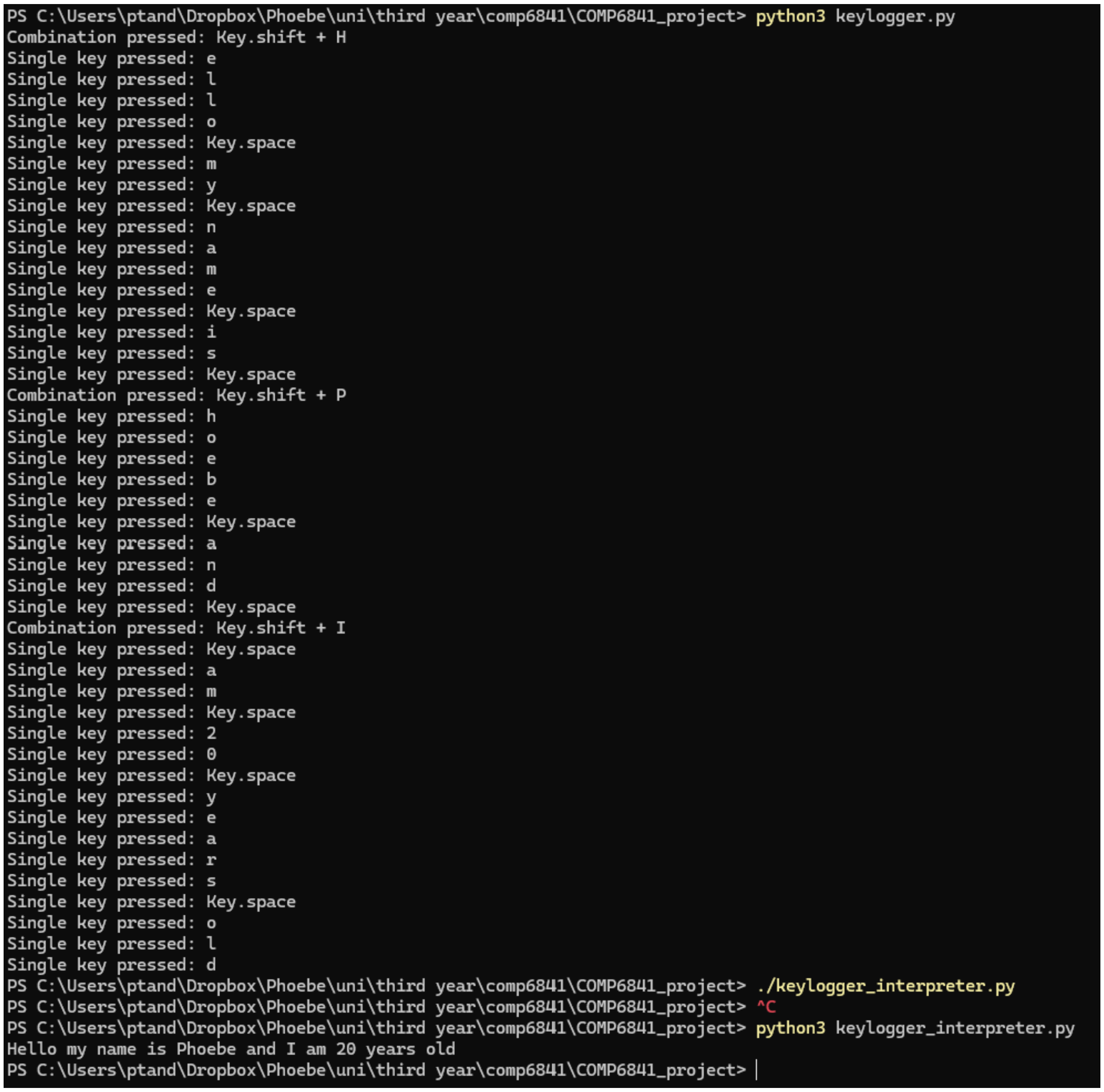
Salas-Nino, M., Ritter, G., Hamdan, D., Wang, T., & Hou, T. (2023). *The Evolution of Keylogger Technologies: A Survey from.* Texas: Texas State University.

Seitz, J. (2014). *Black Hat Python: Python Programming for Hackers and Pentester.* United States: No Starch Press.

Veracode. (2024). *Keyloggers: Detectors, PC Monitors, Keylogger Software, What Is a Keylogger*. Retrieved from Veracode: https://www.veracode.com/security/keylogger

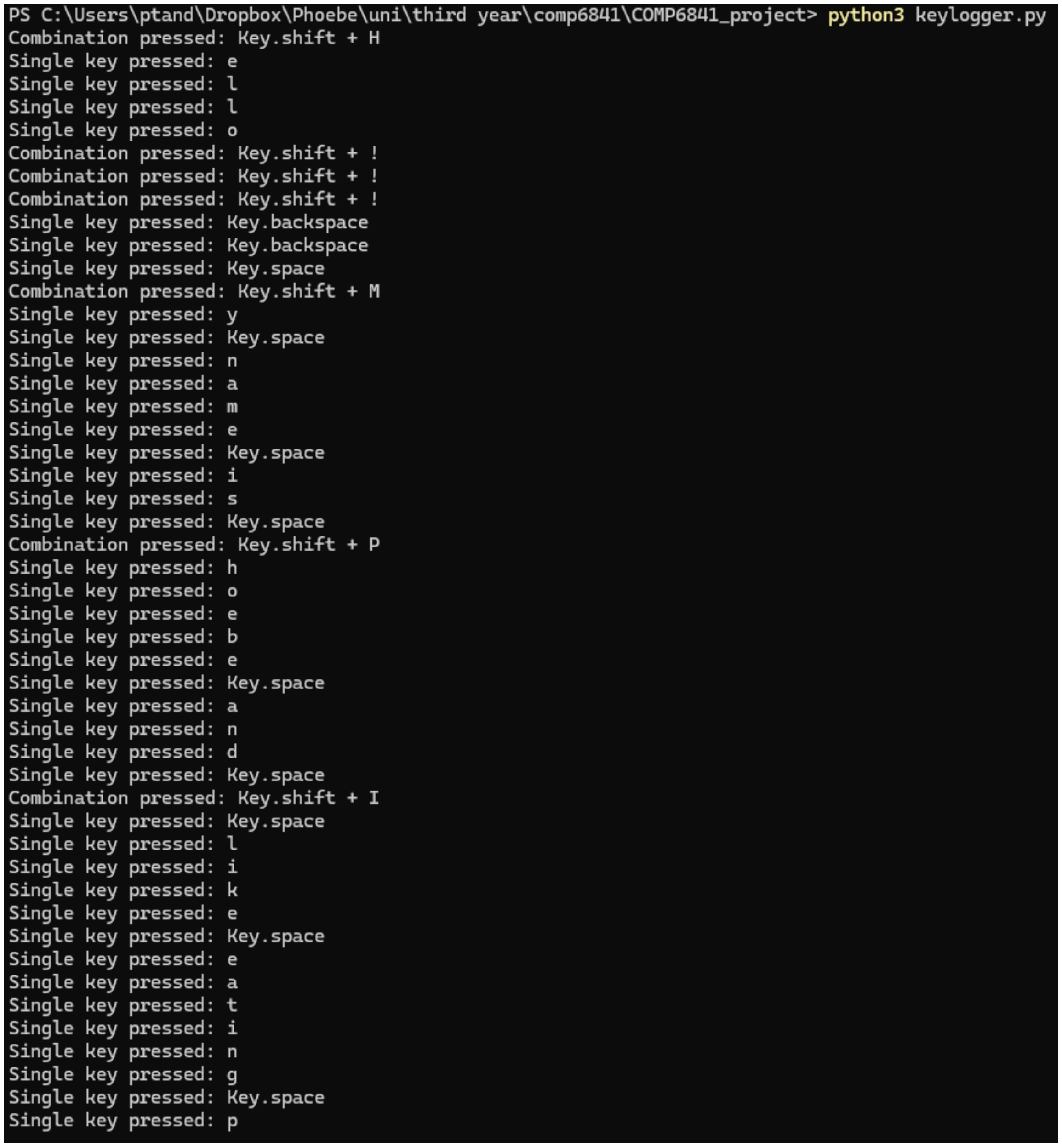
VICE Asia. (2019, September 7). *How the Legendary Hacker Kevin Mitnick Stole the Motorola Source Code*. Retrieved from VICE Asia: https://www.youtube.com/watch?v=fT8Jw-63MXM&ab_channel=VICEAsia

Appendix



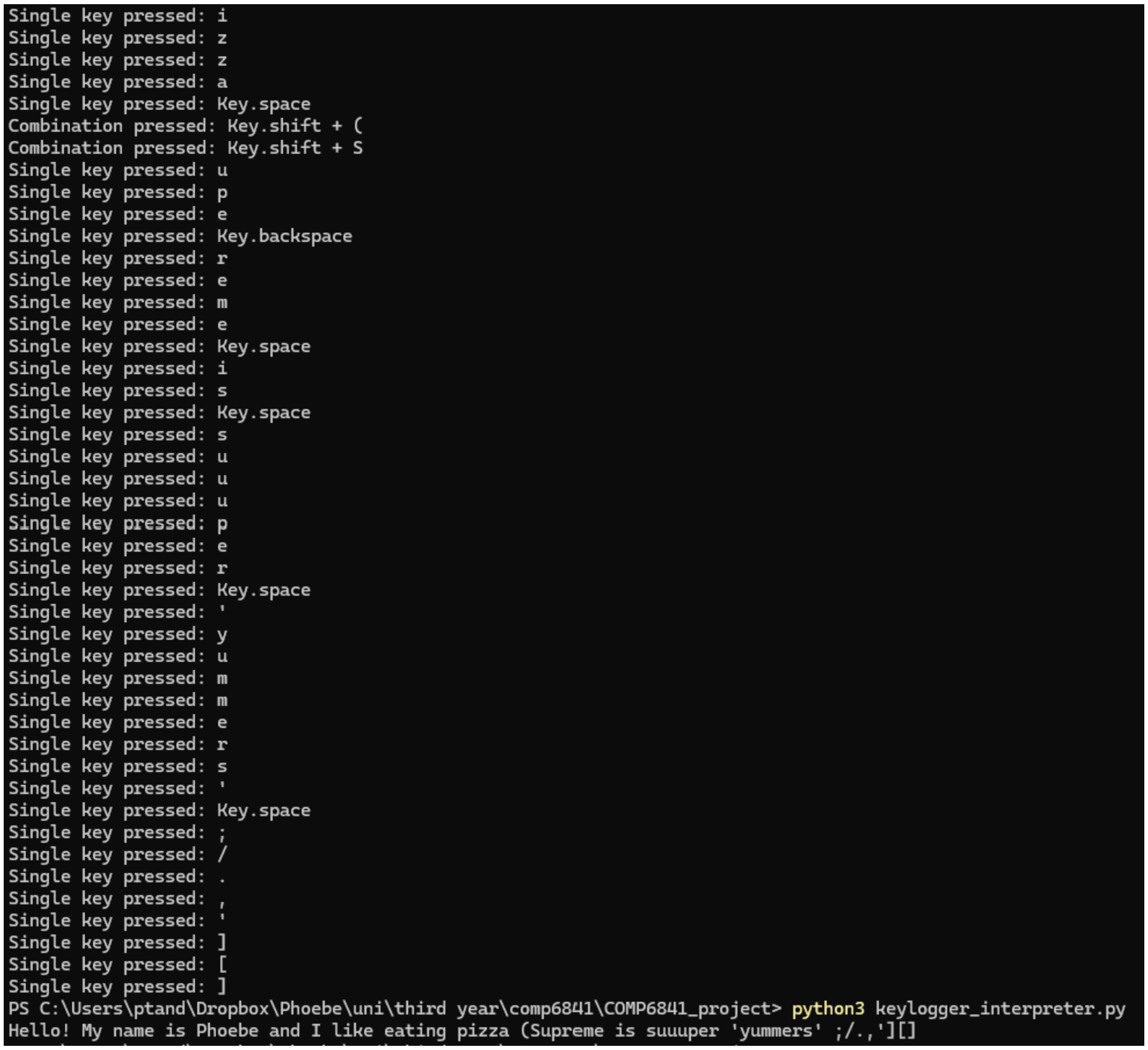
```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger.py
Combination pressed: Key.shift + H
Single key pressed: e
Single key pressed: l
Single key pressed: l
Single key pressed: o
Single key pressed: Key.space
Single key pressed: m
Single key pressed: y
Single key pressed: Key.space
Single key pressed: n
Single key pressed: a
Single key pressed: m
Single key pressed: e
Single key pressed: Key.space
Single key pressed: i
Single key pressed: s
Single key pressed: Key.space
Combination pressed: Key.shift + P
Single key pressed: h
Single key pressed: o
Single key pressed: e
Single key pressed: b
Single key pressed: e
Single key pressed: Key.space
Single key pressed: a
Single key pressed: n
Single key pressed: d
Single key pressed: Key.space
Combination pressed: Key.shift + I
Single key pressed: Key.space
Single key pressed: a
Single key pressed: m
Single key pressed: Key.space
Single key pressed: 2
Single key pressed: 0
Single key pressed: Key.space
Single key pressed: y
Single key pressed: e
Single key pressed: a
Single key pressed: r
Single key pressed: s
Single key pressed: Key.space
Single key pressed: o
Single key pressed: l
Single key pressed: d
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> ./keylogger_interpreter.py
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> ^C
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger_interpreter.py
Hello my name is Phoebe and I am 20 years old
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project>
```

Figure 5.1: Test 1 results



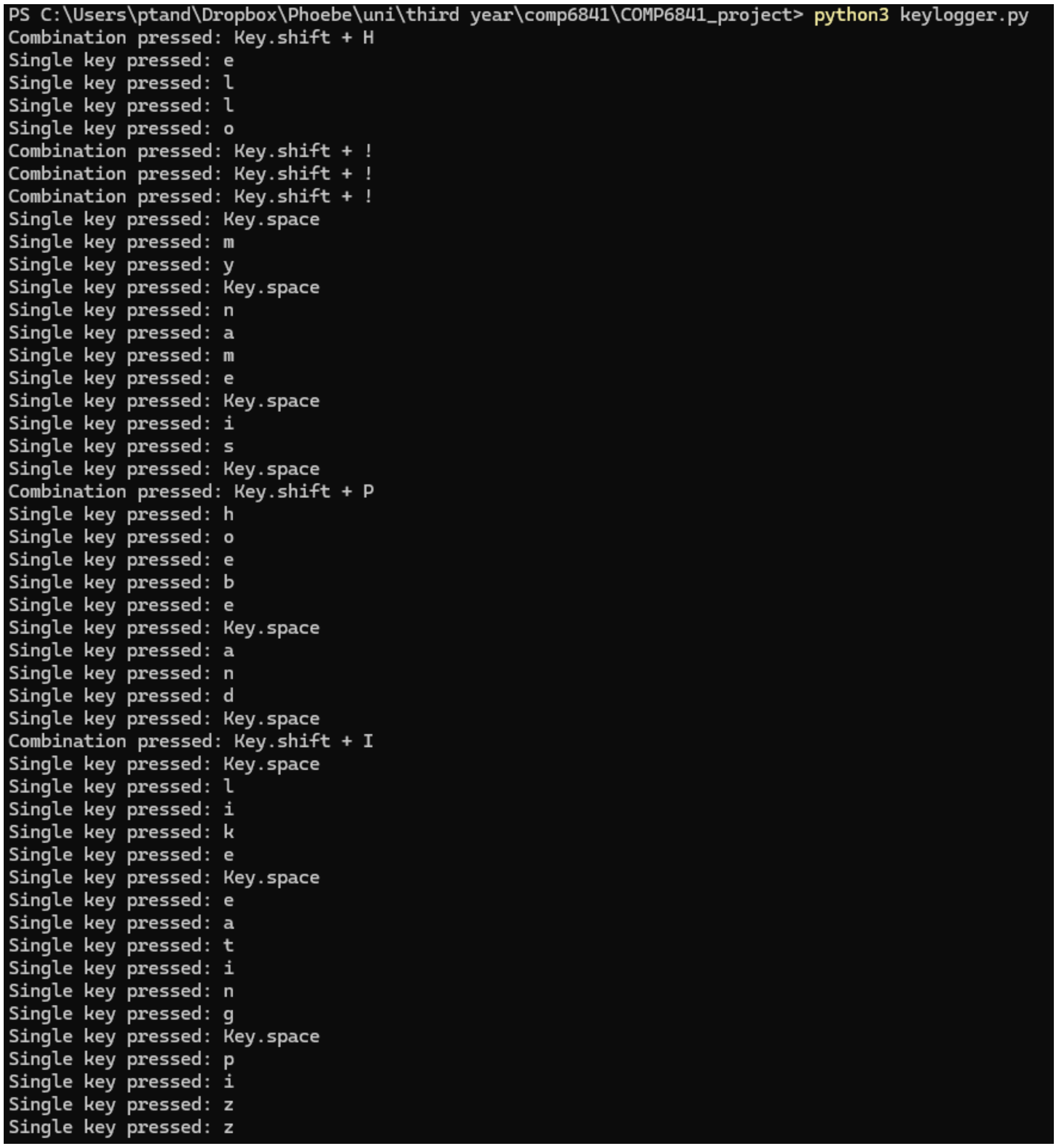
```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger.py
Combination pressed: Key.shift + H
Single key pressed: e
Single key pressed: l
Single key pressed: l
Single key pressed: o
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Single key pressed: Key.backspace
Single key pressed: Key.backspace
Single key pressed: Key.space
Combination pressed: Key.shift + M
Single key pressed: y
Single key pressed: Key.space
Single key pressed: n
Single key pressed: a
Single key pressed: m
Single key pressed: e
Single key pressed: Key.space
Single key pressed: i
Single key pressed: s
Single key pressed: Key.space
Combination pressed: Key.shift + P
Single key pressed: h
Single key pressed: o
Single key pressed: e
Single key pressed: b
Single key pressed: e
Single key pressed: Key.space
Single key pressed: a
Single key pressed: n
Single key pressed: d
Single key pressed: Key.space
Combination pressed: Key.shift + I
Single key pressed: Key.space
Single key pressed: l
Single key pressed: i
Single key pressed: k
Single key pressed: e
Single key pressed: Key.space
Single key pressed: e
Single key pressed: a
Single key pressed: t
Single key pressed: i
Single key pressed: n
Single key pressed: g
Single key pressed: Key.space
Single key pressed: p
```

Figure 5.2: Test 2 input



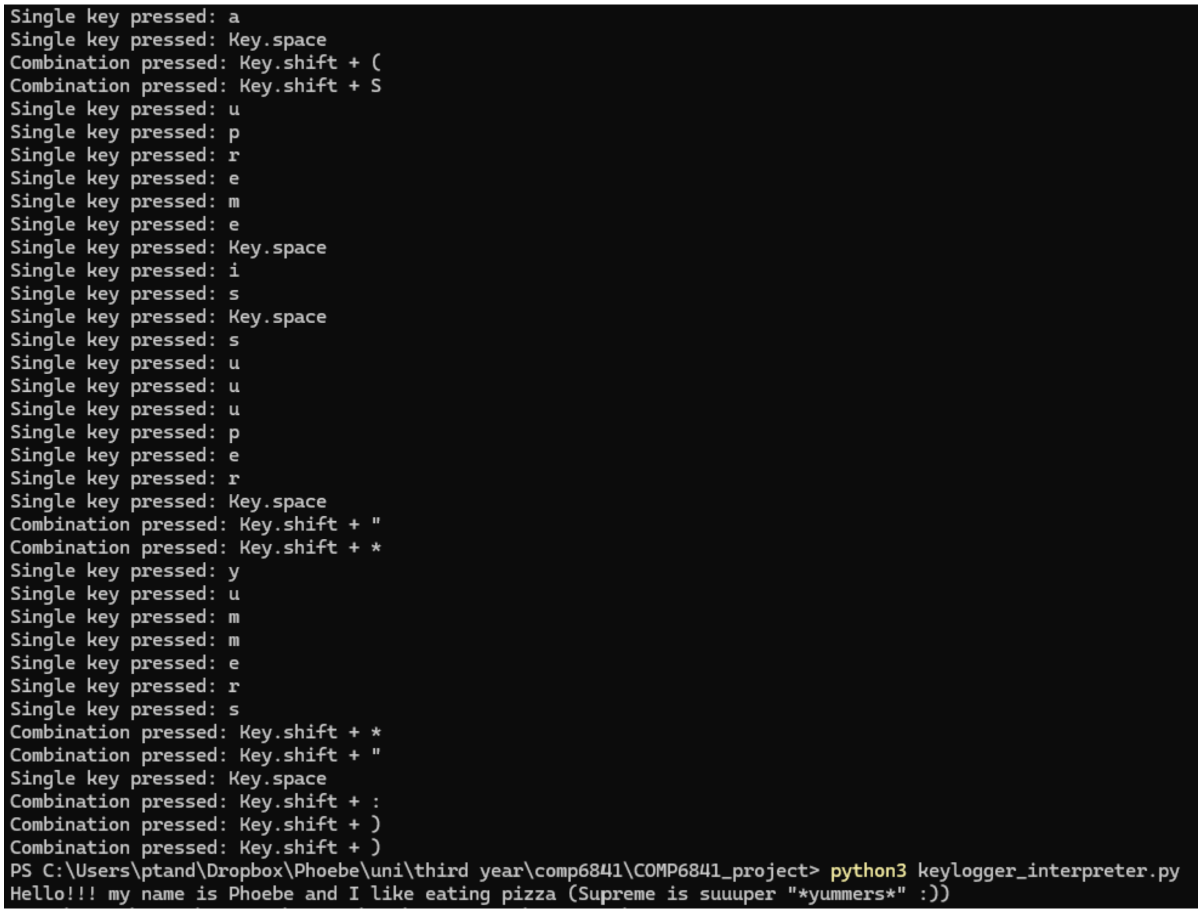
```
Single key pressed: i
Single key pressed: z
Single key pressed: z
Single key pressed: a
Single key pressed: Key.space
Combination pressed: Key.shift + (
Combination pressed: Key.shift + S
Single key pressed: u
Single key pressed: p
Single key pressed: e
Single key pressed: Key.backspace
Single key pressed: r
Single key pressed: e
Single key pressed: m
Single key pressed: e
Single key pressed: Key.space
Single key pressed: i
Single key pressed: s
Single key pressed: Key.space
Single key pressed: s
Single key pressed: u
Single key pressed: u
Single key pressed: u
Single key pressed: p
Single key pressed: e
Single key pressed: r
Single key pressed: Key.space
Single key pressed: '
Single key pressed: y
Single key pressed: u
Single key pressed: m
Single key pressed: m
Single key pressed: e
Single key pressed: r
Single key pressed: s
Single key pressed: '
Single key pressed: Key.space
Single key pressed: ;
Single key pressed: /
Single key pressed: .
Single key pressed: ,
Single key pressed: '
Single key pressed: ]
Single key pressed: [
Single key pressed: ]
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger_interpreter.py
Hello! My name is Phoebe and I like eating pizza (Supreme is suuuper 'yummers' ;/.,'][]
```

Figure 5.3: Test 2 results



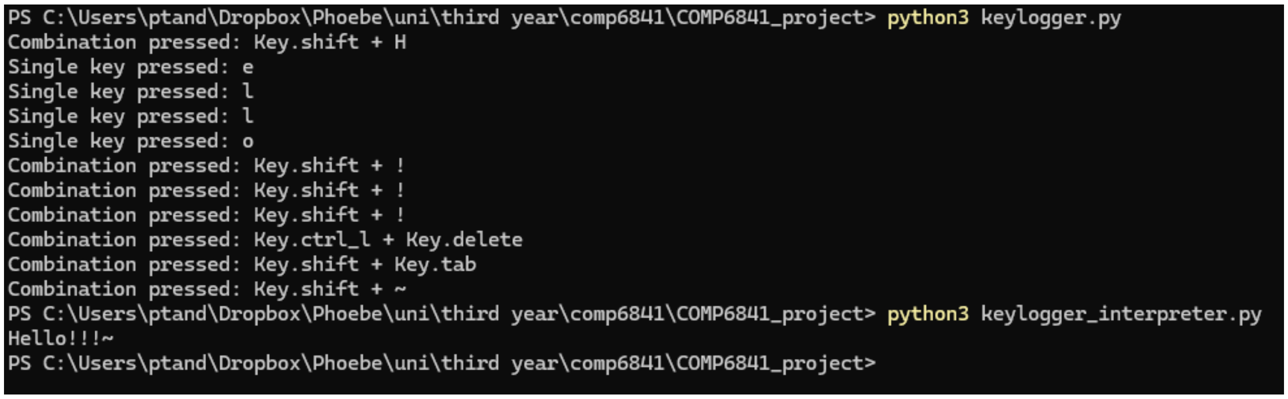
```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger.py
Combination pressed: Key.shift + H
Single key pressed: e
Single key pressed: l
Single key pressed: l
Single key pressed: o
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Single key pressed: Key.space
Single key pressed: m
Single key pressed: y
Single key pressed: Key.space
Single key pressed: n
Single key pressed: a
Single key pressed: m
Single key pressed: e
Single key pressed: Key.space
Single key pressed: i
Single key pressed: s
Single key pressed: Key.space
Combination pressed: Key.shift + P
Single key pressed: h
Single key pressed: o
Single key pressed: e
Single key pressed: b
Single key pressed: e
Single key pressed: Key.space
Single key pressed: a
Single key pressed: n
Single key pressed: d
Single key pressed: Key.space
Combination pressed: Key.shift + I
Single key pressed: Key.space
Single key pressed: l
Single key pressed: i
Single key pressed: k
Single key pressed: e
Single key pressed: Key.space
Single key pressed: e
Single key pressed: a
Single key pressed: t
Single key pressed: i
Single key pressed: n
Single key pressed: g
Single key pressed: Key.space
Single key pressed: p
Single key pressed: i
Single key pressed: z
Single key pressed: z
```

Figure 5.4: Test 3 input



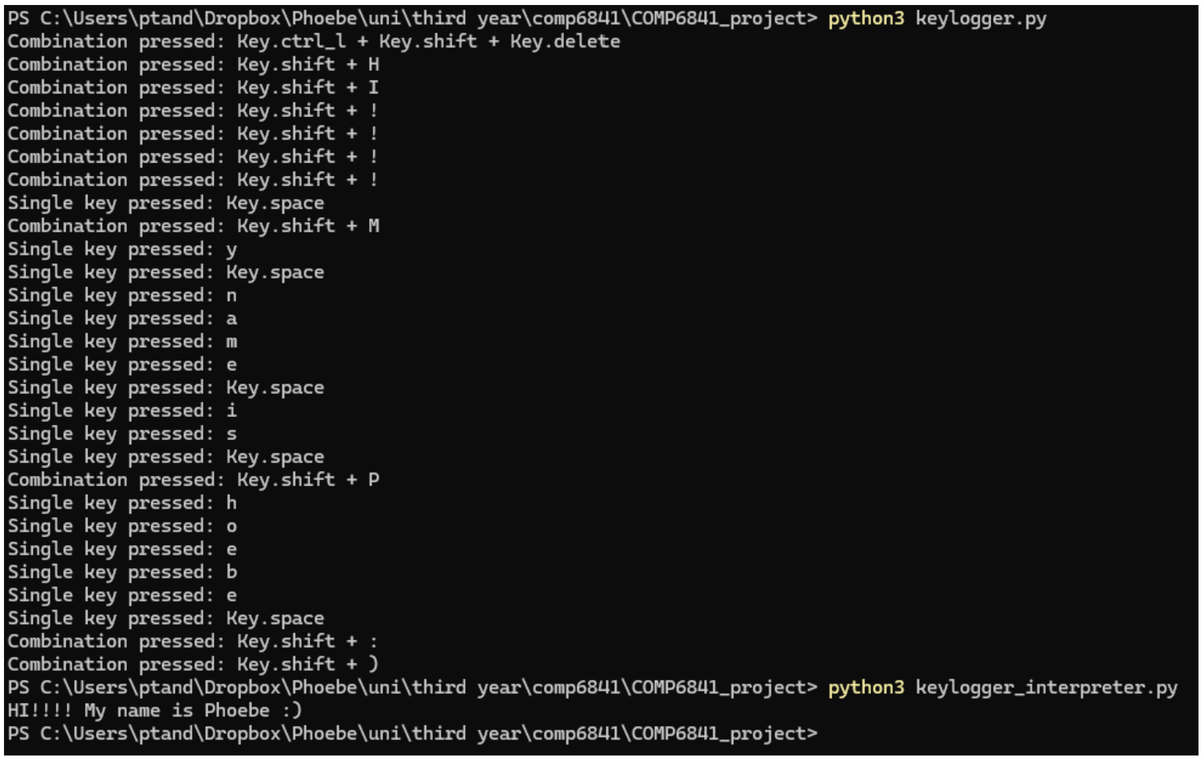
```
Single key pressed: a
Single key pressed: Key.space
Combination pressed: Key.shift + (
Combination pressed: Key.shift + S
Single key pressed: u
Single key pressed: p
Single key pressed: r
Single key pressed: e
Single key pressed: m
Single key pressed: e
Single key pressed: Key.space
Single key pressed: i
Single key pressed: s
Single key pressed: Key.space
Single key pressed: s
Single key pressed: u
Single key pressed: u
Single key pressed: u
Single key pressed: p
Single key pressed: e
Single key pressed: r
Single key pressed: Key.space
Combination pressed: Key.shift + "
Combination pressed: Key.shift + *
Single key pressed: y
Single key pressed: u
Single key pressed: m
Single key pressed: m
Single key pressed: e
Single key pressed: r
Single key pressed: s
Combination pressed: Key.shift + *
Combination pressed: Key.shift + "
Single key pressed: Key.space
Combination pressed: Key.shift + :
Combination pressed: Key.shift + )
Combination pressed: Key.shift + )
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger_interpreter.py
Hello!!! my name is Phoebe and I like eating pizza (Supreme is suuuper "*yummers*" :))
```

Figure 5.5: Test 3 results



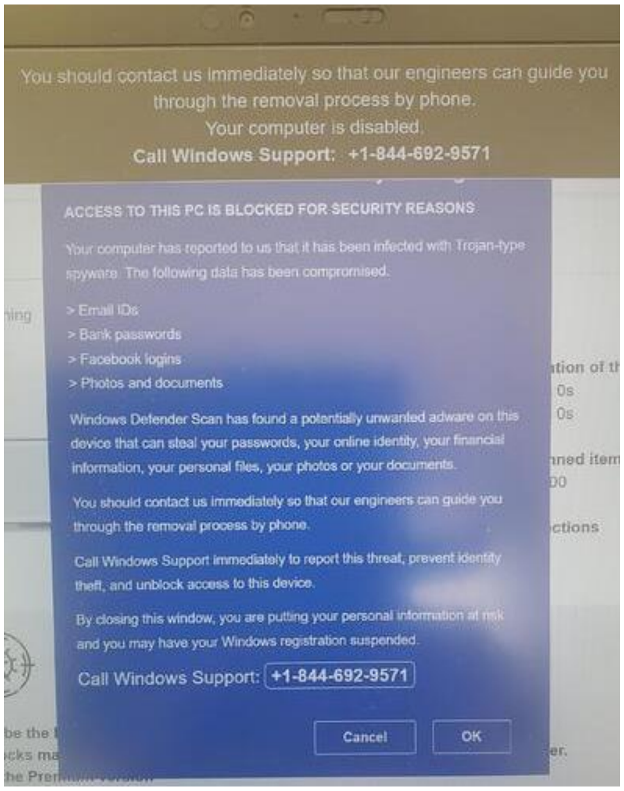
```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger.py
Combination pressed: Key.shift + H
Single key pressed: e
Single key pressed: l
Single key pressed: l
Single key pressed: o
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Combination pressed: Key.ctrl_l + Key.delete
Combination pressed: Key.shift + Key.tab
Combination pressed: Key.shift + ~
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger_interpreter.py
Hello!!!~
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project>
```

Figure 5.6: Test 4 results



```
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger.py
Combination pressed: Key.ctrl_l + Key.shift + Key.delete
Combination pressed: Key.shift + H
Combination pressed: Key.shift + I
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Combination pressed: Key.shift + !
Single key pressed: Key.space
Combination pressed: Key.shift + M
Single key pressed: y
Single key pressed: Key.space
Single key pressed: n
Single key pressed: a
Single key pressed: m
Single key pressed: e
Single key pressed: Key.space
Single key pressed: i
Single key pressed: s
Single key pressed: Key.space
Combination pressed: Key.shift + P
Single key pressed: h
Single key pressed: o
Single key pressed: e
Single key pressed: b
Single key pressed: e
Single key pressed: Key.space
Combination pressed: Key.shift + :
Combination pressed: Key.shift + )
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project> python3 keylogger_interpreter.py
HI!!!! My name is Phoebe :)
PS C:\Users\ptand\Dropbox\Phoebe\uni\third year\comp6841\COMP6841_project>
```

Figure 5.7: Test 5 results



You should contact us immediately so that our engineers can guide you through the removal process by phone.
Your computer is disabled.
Call Windows Support: +1-844-692-9571

ACCESS TO THIS PC IS BLOCKED FOR SECURITY REASONS

Your computer has reported to us that it has been infected with Trojan-type spyware. The following data has been compromised.

> Email IDs
> Bank passwords
> Facebook logins
> Photos and documents

Windows Defender Scan has found a potentially unwanted adware on this device that can steal your passwords, your online identity, your financial information, your personal files, your photos or your documents.

You should contact us immediately so that our engineers can guide you through the removal process by phone.

Call Windows Support immediately to report this threat, prevent identity theft, and unblock access to this device.

By closing this window, you are putting your personal information at risk and you may have your Windows registration suspended.

Call Windows Support: +1-844-692-9571

Figure 6: example of a popup scam