**PROJECT REPORT**

**BYTE SENTRY**

*Submitted for the Partial Fulfilment of the Requirements*

*for the Award of the Degree of*

**BSc. Cyber Forensic**

Done by

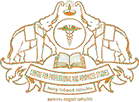
**GOKUL KRISHNA M**

**210021042645**

Under the guidance of

**Ms. TEENA MATHEW**

**STAS, EDAPPALLY**



**CENTRE FOR PROFESSIONAL AND ADVANCED STUDIES**

**SCHOOL OF TECHNOLOGY & APPLIED SCIENCES**

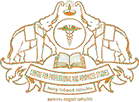
**EDAPPALLY, KERALA**

**2021-24**

**CENTRE FOR PROFESSIONAL AND ADVANCED STUDIES**

**SCHOOL OF TECHNOLOGY & APPLIED SCIENCES**

**EDAPPALLY, KERALA**



**CERTIFICATE**

This is to certify that the project entitled “**BYTE SENTRY”** is a bonafide work done by **GOKUL KRISHNA M** submitted in the partial fulfilment of the requirement of the degree of B.sc Cyber forensic of school of Technology and Applied Sciences, Edappally during the period 2021-2024.

Mrs. Swapna Ms. Teena Mathew

(Head of the Department) (Internal Guide)

Internal Examiner External Examiner

Date:

**ACKNOWLEDGEMENT**

Dedicating this project report to the Almighty God whose abundant grace and mercies enabled its successful completion, we would like to express our profound gratitude to all the people who had inspired and motivated us to make this report a success.

I would express our gratitude to the Management of School of Technology & Applied Sciences for providing with all the required facilities without which the report would not have been possible. I express my heartful gratitude to our Principal **Thirumeni K R, School of Technology and Applied Science** for his warm support with regard to our work.

I would express deep sense of gratitude to **Mrs. Swapna M** (Head of the department computer science, School of Technology and Applied Sciences, Edappally) for her valuable help and guidance.

I am deeply indebted to my guide **Ms. Teena Mathew** for providing me with valuable advice and guidance.

I also extend my sincere thanks to all other members of the faculty of the department of Cyber forensics for their assistance and encouragement.

Last, but not least I would like to thank my friends for their co-operations and encouragement.

**DECLARATION**

I **GOKUL KRISHNA M**, hereby declare that the mini project report “**BYTE SENTRY**” is submitted in partial fulfilment of the requirements for the fifth semester of Bachelor of Science and it is a report of the original work done by me in the department of Cyber Forensics at School of Technology and Applied Sciences, Edappally. in MG university.

Place: Edappally **GOKUL KRISHNA M**

Date:

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SL N0** | **TITLE** | **PAGE** |
| **1.** | **ABSTRACT** | **8** |
| **2.** | **INTRODUCTION** | **9** |
|  | **2.1 About the project** |  |
| **3.** | **SYSTEM STUDY AND ANALYSIS** | **11** |
|  | **3.1 Existing system** |  |
|  | **3.1.1 Disadvantage** |  |
|  | **3.2 Proposed system** |  |
|  | **3.2.1 Advantages** |  |
| **4.** | **SYSTEM CONFIGURATION** | **13** |
|  | **4.1 Hardware configuration** |  |
|  | **4.2 Software configuration** |  |
|  | **4.2.1 Backend** |  |
| **5.** | **HASH CODE** | **14** |
|  | **5.1 SHA 256** |  |

|  |  |  |
| --- | --- | --- |
|  | **5.1.1 Characteristics of the SHA-256**  **Algorithm** |  |
|  | **5.1.2 Steps in SHA-256 Algorithm** |  |
| **6.** | **SYSYTEM TESTING AND IMPLEMENTATION** | **19** |
|  | **6.1 SYSTEM TESTING** |  |
|  | **6.1.1 Unit Testing** |  |
|  | **6.1.2 Integration Testing** |  |
|  | **6.1.3 Validation Testing** |  |
|  | **6.1.4 Output Testing** |  |
|  | **6.1.5 User Acceptance Testing (UAT):** |  |
| **7.** | **IMPLEMENTATION** | **22** |
|  | **7.1 Deployment** |  |
|  | **7.2 Dependencies** |  |
|  | **7.3 Configuration** |  |
|  | **7.4 Security** |  |

|  |  |  |
| --- | --- | --- |
|  | **7.5 App password** |  |
| **8.** | **BIBLIOGRAPHY** | **25** |
| **9.** | **CONCLUSION** | **27** |
| **10.** | **FURTHER SCOPE OF THE PROJECT** | **28** |
| **11.** | **APPENDIX** | **30** |
|  | **11.1 Screenshots** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**1.ABSTRACT**

The File Monitoring System is a Python-based application designed to provide users with real-time vigilance over their files. It operates through a user-friendly web interface where users input their email addresses and passwords, which are used for notification purposes. Upon uploading files, individual threads are created to continuously compute and compare cryptographic hashes of their content. If any unauthorized modification is detected, an immediate email notification is dispatched to the user, specifying the affected file.

This application serves as a proactive security measure, enhancing data integrity and confidentiality by promptly alerting users to any alterations. Its primary aim is to offer users a watchful eye on their crucial files. By delivering real-time alerts for unauthorized changes, it empowers users to take immediate action, whether that involves investigating the cause or restoring the original file. This proactive approach towards file security ensures that data remains intact and confidential, even in the face of potential threats.

One of the key strengths of the File Monitoring System lies in its ability to provide real-time monitoring, enabling swift responses and interventions in the event of any suspicious activities. The user-friendly interface further enhances its effectiveness, allowing users to easily select and monitor files. Additionally, the inclusion of email notifications ensures that users are promptly informed, even when they are not actively using the application. The system also supports simultaneous monitoring of multiple files, making it suitable for a wide range of use cases. Furthermore, the application employs cryptographic hashing, a reliable method for detecting even the slightest modifications, ensuring robust data integrity..

**2.INTRODUCTION**

The provided Python script creates a graphical user interface (GUI) application for file monitoring and email notification. The application leverages the Tkinter library for GUI elements and integrates custom styling using the 'customtkinter' module. Users can interact with the application to select files for monitoring, input their email credentials, and initiate the monitoring process.

Upon launching the application, users are greeted with a visually appealing interface. A background image, loaded with the help of the Pillow library, provides an aesthetic backdrop. The GUI includes labeled entry fields for the user's email and app password, as well as a button to select files for monitoring. A listbox displays the selected files, while a "Start Monitoring" button is initially disabled until files are chosen.

The core functionality of the application revolves around two key processes: file hashing and email notifications. When files are selected, their hashes are computed using the hashlib library, ensuring data integrity. In case of file modification, an email is generated using the smtplib library and dispatched to the user's provided email address, notifying them of the alteration.

A multi-threaded approach is employed to enable concurrent monitoring of multiple files. This ensures efficiency and responsiveness in the monitoring process. Additionally, the application accounts for potential security measures by prompting users to enable two-factor authentication.

Overall, this application serves as a proactive security measure, alerting users to any unauthorized changes in their selected files via real-time email notifications. By combining GUI elements, multi-threading, and email integration, it provides a user-friendly and effective solution for file monitoring.

**2.1 About the project**

This Python application creates a robust file monitoring system with integrated email notifications. It utilizes the Tkinter library to create a graphical user interface (GUI) for user interaction. Through this interface, users can easily select files for monitoring. The system handles multiple files concurrently by using separate threads for each uploaded file.

Upon file submission, the application saves them in a specified directory. This provides a dedicated space for tracking any alterations made to the files. SHA-256 cryptographic hashing is used to generate unique fingerprints for each file. These hashes serve as a reference for comparison.

Whenever a modification is detected, the system promptly sends an email notification to the user who originally uploaded the file. This notification contains essential details about the nature of the modification, such as whether the file has been altered. The smtplib module facilitates seamless communication between the application and the user's email service.

To ensure smooth operation, the application takes precautions to create the upload directory if it does not already exist. This mitigates potential errors or disruptions in the monitoring process.

In summary, this file monitoring system provides an efficient and user-friendly solution for tracking modifications in uploaded files. Its combination of Tkinter for GUI development, multi-threading for concurrent monitoring, and email notifications for timely alerts makes it a powerful tool for users seeking to stay informed about any changes to their files.

**3.SYSTEM STUDY AND ANALYSIS**

**3.1 Existing system**

* Traditional file integrity monitoring methods rely on manual processes and periodic checks.
* These methods involve comparing file hashes or checksums manually, which is time-consuming and prone to human error.
* Furthermore, the lack of real-time detection capabilities hinders immediate response to file modifications.

**3.1.1 Disadvantage**

1. Manual Monitoring: Requires manual effort to compare file hashes, making it tedious and error prone.
2. Lack of Real-time Detection: Inability to detect file modifications immediately, leading to delayed response.
3. Limited Scalability: Traditional systems struggle to handle large-scale file monitoring and management.
4. Reliance on Weak Hash Algorithms: Many traditional systems rely on the MD5 algorithm for file hashing, which poses security risks.

**3.2 Proposed system**

* The proposed system is a Python script that automates file monitoring and notification.
* It provides a user-friendly interface for selecting the file to monitor and utilizes the SHA-256 hashing algorithm to track file changes.
* When a modification occurs, the system sends an email notification to the user, ensuring timely awareness of any file modification.

**3.2.1 Advantages**

1. Strong Hash Algorithms: File Integrity Sentinel utilizes robust hash algorithms like SHA-256, ensuring secure and reliable file integrity verification.
2. Email Alerts and Notifications: The tool sends email alerts to specified recipients, enabling quick response to file integrity issues.
3. Scalability: File Integrity Sentinel is designed to handle large-scale file monitoring, making it suitable for organizations with diverse file environments.

**4.** **SYSTEM CONFIGURATION**

* 1. **Hardware configuration**
  + Processor (CPU): Intel Core i7-11500
  + Memory (RAM): 12 GB DDR4
  + Storage: 1TB NVMe class 35 SSD
  + Display: 23.8” FHD Non-Touch Anti-Glare
  + Keyboard: Dell KB216 Wired Multimedia USB Keyboard
  + Mouse: Dell MS116 Optical Mouse
  1. **Software configuration**
* Operating System: Windows 10 or 11
* Python: Version 3.7
* Integrated Development Environments (IDEs): PyCharm, Visual Studio
  + 1. **Backend**

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Python supports multiple programming paradigms, including Procedural, Object Oriented and Functional programming language. Python design philosophy emphasizes code readability with the use of significant.

**5.HASH CODE**

A close-up of a lock

Description automatically generatedHashing is the process of scrambling raw information to the extent that it cannot reproduce it back to its original form. It takes a piece of information and passes it through a function that performs mathematical operations on the plaintext. This function is called the hash function, and the output is called the hash value/digest.

As seen from the above image, the hash function is responsible for converting the plaintext to its respective hash digest. They are designed to be irreversible, which means your digest should not provide you with the original plaintext by any means necessary. Hash functions also provide the same output value if the input remains unchanged, irrespective of the number of iterations.

This property is crucial for various applications like data integrity verification. Even a slight modification in the input, such as changing a single character, will result in a completely different hash value. This is a fundamental characteristic called the "avalanche effect."

Even a small change in the input will produce a significantly different hash code. Even if you alter a tiny portion of the original input, the resulting hash value will be radically different. This property ensures that hash functions are highly sensitive to changes in the input data.

**5.1 SHA 256**

SHA 256 is a part of the SHA 2 family of algorithms, where SHA stands for Secure Hash Algorithm. Published in 2001, it was a joint effort between the NSA and NIST to introduce a successor to the SHA 1 family, which was slowly losing strength against brute force attacks.

The significance of the 256 in the name stands for the final hash digest value, i.e. irrespective of the size of plaintext/cleartext, the hash value will always be 256 bits.

The other algorithms in the SHA family are more or less similar to SHA 256. Now, look into knowing a little more about their guidelines.

Embark on a transformative journey through our Cyber security Bootcamp, where you'll delve deep into the intricacies of cutting-edge technologies like the SHA-256 algorithm. Uncover the cryptographic principles that make this algorithm the cornerstone of blockchain security, all while honing your skills in defending against cyber threats.

**5.1.1Characteristics of the SHA-256 Algorithm**

A graphic of a paper with a hashtag

Description automatically generated

* Message Length: The length of the cleartext should be less than 264 bits. The size needs to be in the comparison area to keep the digest as random as possible.
* Digest Length: The length of the hash digest should be 256 bits in SHA 256 algorithm, 512 bits in SHA-512, and so on. Bigger digests usually suggest significantly more calculations at the cost of speed and space.
* Irreversible: By design, all hash functions such as the SHA 256 are irreversible. You should neither get a plaintext when you have the digest beforehand nor should the digest provide its original value when you pass it through the hash function again.

**5.1.2Steps in SHA-256 Algorithm**

You can divide the complete process into five different segments, as mentioned below:

* **Padding Bits**

A diagram of a diagram

Description automatically generated with medium confidenceIt adds some extra bits to the message, such that the length is exactly 64 bits short of a multiple of 512. During the addition, the first bit should be one, and the rest of it should be filled with zeroes.

* **Padding Length**

A diagram of a padding bits

Description automatically generatedYou can add 64 bits of data now to make the final plaintext a multiple of 512. You can calculate these 64 bits of characters by applying the modulus to your original cleartext without the padding.

* **Compression Functions**

The entire message gets broken down into multiple blocks of 512 bits each. It puts each block through 64 rounds of operation, with the output of each block serving as the input for the following block. The entire process is as follows:

A diagram of a computer program

Description automatically generated

While the value of K[i] in all those rounds is pre-initialized, W[i] is another input that is calculated individually for each block, depending on the number of iterations being processed at the moment.

* **Output**

With each iteration, the final output of the block serves as the input for the next block. The entire cycle keeps repeating until you reach the last 512-bit block, and you then consider its output the final hash digest. This digest will be of the length 256-bit, as per the name of this algorithm.

**5.2 SHA FAMILY**

The Secure Hash Algorithm (SHA) family is a set of cryptographic hash functions that play a crucial role in ensuring data integrity and security in various applications across the digital landscape. These algorithms are designed to take an input (or message) of arbitrary length and produce a fixed-size hash value, typically represented as a string of hexadecimal characters. This hash value, also known as the digest, is unique to the input data, meaning that even a small change in the input will result in a significantly different hash.

The SHA family was developed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in the United States. It includes several variants, each denoted by a number, indicating the specific version. The most widely used members of the SHA family include SHA-1, SHA-256, SHA-384, and SHA-512, which produce hash values of 160, 256, 384, and 512 bits respectively.

SHA-1, once widely used for secure communications and data integrity, has become vulnerable to collision attacks, where two different inputs can produce the same hash value. This weakness led to its deprecation in favor of more secure algorithms.SHA-256, SHA-384, and SHA-512, collectively known as the SHA-2 family, are considered much more secure and remain widely used today. They offer significantly higher levels of cryptographic strength and are the preferred choice for tasks like generating digital signatures, securing SSL/TLS connections, and ensuring the integrity of data in various applications.

In recent years, SHA-3 has emerged as an alternative to SHA-2. It was selected through a public competition organized by NIST, and it uses a fundamentally different approach, known as Keccak, compared to the algorithms in the SHA-2 family. SHA-3 provides similar security guarantees to SHA-2 but employs a different internal structure, making it a valuable addition to the cryptographic toolkit.

**6. SYSTEM TESTING AND IMPLEMENTATION**

**6.1 SYSTEM TESTING**

System testing is a comprehensive and end-to-end testing phase that evaluates the entire software system as a whole. Its primary objective is to ensure that the system functions as intended and meets its defined requirements. System testing checks not only the functionality but also the performance, security, scalability, and usability of the application. It encompasses various scenarios, including typical and edge cases, and often involves testing in a production-like environment. It assesses whether the application performs reliably and consistently while meeting user expectations. The basic types of testing are:

➢ Unit testing

➢Integration testing

➢ Validation testing

➢ Output testing

➢ User acceptance testing

**6.1.1 Unit Testing**

Unit testing is a fundamental practice in software development that involves validating the accuracy of discrete units of code, such as functions, methods, or classes. These units are self-contained components responsible for specific tasks. The primary aim of unit testing is to confirm that each unit performs its designated function correctly, generating the anticipated result for a particular input. In the context of this codebase, unit testing would entail crafting test cases tailored to individual functions or methods. For instance, tests can be designed to scrutinize the behavior of functions like 'sendEmail', 'getHash', and 'startMonitoring'.

**6.1.2 Integration Testing**

Integration testing is a critical phase in software testing that assesses the interaction and cooperation between multiple units or modules within a program. Unlike unit testing, which focuses on individual components in isolation, integration testing evaluates how these components collaborate when integrated together. In the context of this codebase, integration testing would involve examining the seamless interaction between different functions, such as 'getHash', 'sendEmail', and 'startMonitoring'. It ensures that these units harmonize effectively, with no unexpected conflicts or miscommunications.

**6.1.3 Validation Testing**

Validation testing is a dynamic testing process that assesses whether the software fulfills the specified requirements and meets the needs of the end-users. It aims to confirm that the software delivers the expected functionality, features, and performance as outlined in the requirements documentation. In the context of this codebase, validation testing would entail verifying that the application accurately monitors file modifications, sends email notifications, and handles user inputs for email and password credentials. This testing phase assures that the software aligns with the intended purpose and user expectations.

**6.1.4 Output Testing**

Output testing, also known as results-based testing, concentrates on validating the output or results generated by the software in response to specific inputs or stimuli. It involves comparing the actual output with the expected output to ensure they match. In the context of this codebase, output testing would involve confirming that functions like 'getHash' produce the correct hash values for various input files. Similarly, it would validate that 'sendEmail' functions as expected by successfully sending email notifications upon detecting file modifications. Output testing serves as a crucial step in affirming the accuracy and reliability of the application's generated results.

**6.1.5 User Acceptance Testing (UAT):**

User Acceptance Testing, often abbreviated as UAT, is a crucial phase in the software testing process that involves end-users or stakeholders evaluating the software's functionality to ensure it meets their specific requirements and expectations. UAT aims to validate that the software system aligns with the business objectives and is ready for real-world use. In the context of this codebase, UAT would involve actual end-users or designated stakeholders testing the application to confirm that it effectively monitors file modifications, sends email notifications, and handles user inputs for email and password credentials. It serves as a final validation step before the software is deployed.

.

**7. IMPLEMENTATION**

**7.1 Deployment**

This Python script creates a customized GUI application for file monitoring using tkinter, featuring an aesthetically enhanced interface with a background image. Users input their email and an application-specific password for authentication. They can select files for monitoring, with selected paths displayed in a list. Upon clicking "Start Monitoring," a background thread continuously checks file integrity using SHA256 hashes. If a modification is detected, an email notification is sent using the provided credentials. Error messages are displayed for issues like not selecting any files. A "Close" button allows users to exit the application. This system has been tested in a local environment, but specific deployment details beyond this environment are not provided.

**7.2 Dependencies**

1. hashlib: This library is used for cryptographic hashing operations, specifically to compute the SHA256 hash of files. The hashes are crucial for verifying file integrity and detecting any modifications.

2. smtplib: This library is employed to facilitate the sending of email notifications. It allows the program to connect to an SMTP (Simple Mail Transfer Protocol) server, send constructed email messages, and handle the email communication.

3. email.message.EmailMessage: This module is used to create and format email messages. In this code, it's utilized to construct notification emails that are sent when file modifications are detected.

4. tkinter: This is the standard Python interface for creating graphical user interfaces (GUIs). It provides the necessary elements like windows, buttons, labels, etc., to create the user interface for the application.

5. tkinter.messagebox: This module is a part of the tkinter library and is used to display various types of message boxes, including informational, warning, and error messages. In this code, it's used to show an error message if no files are selected.

6. tkinter.Frame, tkinter.Listbox, tkinter.Scrollbar: These tkinter classes are used to create different elements of the GUI, including frames for organizing widgets, a listbox for displaying selected files, and a scrollbar to navigate through the listbox entries.

7. tkinter.filedialog.askopenfilenames: This function is used to open a file dialog box that allows the user to select one or more files. It returns the file paths of the selected files.

8. PIL (Pillow): This Python Imaging Library (PIL) fork is utilized to handle images. In this code, it is used to open and display a background image for the GUI.

9. customtkinter: This appears to be a custom module or library, but its specific functionality and purpose are not detailed in the provided code. It's likely used for additional customization of the tkinter elements.

**7.3 Configuration**

* Email Address and Password: The email address and password for sending notifications are configured in the sendEmail function.
* SMTP Server and Port: The SMTP server and port are set for Gmail in the same functionScalability.

**7.4 Security**

• File Upload Security: Implement checks to validate uploaded files to prevent malicious uploads.

• Authentication and Authorization: Implement user authentication and authorization mechanisms to ensure only authorized users can access the system.

**7.5 App password**

Creating an app password for your email account allows you to securely connect third-party applications (like email clients, calendar apps, etc.) to your email account. This is useful if you have two-factor authentication (2FA) enabled on your email account, as some applications may not support 2FA directly.

**steps to create an app password**

1. Sign in to your Google Account:

- Go to [https://myaccount.google.com/](https://myaccount.google.com/).

- Log in with your Gmail email address and password.

2. Security Settings:

- Click on "Security" in the left-hand menu.

3. App Passwords:

- Scroll down to the "Signing in to Google" section and find the "App passwords" option. Click on it.

4. Generate App Password:

- If you have 2-Step Verification enabled, you might need to enter your password again.

- Select the app and device you want to generate the app password for.

5. Generate:

- Click on the "Select app" dropdown and choose the app for which you're generating the password.

- Select your device (if applicable).

6. Generate Password:

- Click "Generate".

- You'll receive a 16-digit passcode. This is the app password.

7. Use the App Password:

- Copy this app password and use it to sign in to your email application or service.

**8. BIBLIOGRAPHY**

**1.File Integrity Monitoring:**

• Title: "File System Forensic Analysis"

Author: Brian Carrier

Publisher: Addison-Wesley

• Title: "File System Forensic Analysis CD-ROM"

Author: Brian Carrier

Publisher: Addison-Wesley

**2.SHA-256 Hash Algorithm:**

• Title: "Cryptography and Network Security: Principles and Practice"

Authors: William Stallings

Publisher: Pearson

•Title: "Practical Cryptography"

Authors: Niels Ferguson, Bruce Schneier

Publisher: Wiley

**3.Python Programming:**

•Title: "Automate the Boring Stuff with Python: Practical Programming for Total

Beginners”

Author: Al Sweigart

Publisher: No Starch Press

•Title: "Python Crash Course"

Author: Eric Matthes

Publisher: No Starch Press

•Title: "Effective Python: 90 Specific Ways to Write Better Python"

Author: Brett Slatkin

Publisher: Addison-Wesley

**4. GUI Programming with Tkinter:**

•Title: "Python GUI Programming with Tkinter"

Author: Alan D. Moore

Publisher: Packt Publishing

•Title: "Tkinter GUI Application Development Blueprints"

Author: Bhaskar Chaudhary

Publisher: Packt Publishing

**WEBSITE REFERRED:**

• <https://www.tripwire.com/state-of-security/file-integrity-monitoring>

• <https://www.manageengine.com/products/eventlog/file-integrity-monitoring.html>

• <https://www.movable-type.co.uk/scripts/sha256.html>

• <https://www.grafiati.com/en/literature-selections/sha-256/>

• [https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application- using-flask-in-python-3](https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-%20%20%20%20using-flask-in-python-3)

**9. CONCLUSION**

"Byte Sentry" is a dynamic file monitoring application designed to provide users with vigilant oversight of their critical files. Through its intuitive user interface, the application streamlines the process of selecting and monitoring files, ensuring an accessible and user-friendly experience. Employing multi-threading, "Byte Sentry" optimizes efficiency by concurrently monitoring multiple files, allowing users to keep a watchful eye on a diverse range of files without compromising performance.

One of the application's standout features is its seamless integration of email notifications. In the event of any file modifications, users receive prompt alerts, enabling them to take immediate action. By leveraging Gmail's SMTP services, the application enhances the reliability of email delivery, ensuring that notifications reach users in a timely and consistent manner. This robust notification system adds an extra layer of convenience and security to the file monitoring process.

"Byte Sentry" caters to a broad spectrum of professional and personal scenarios. Professionals working with sensitive documents benefit from the assurance that critical files are actively monitored, providing an additional safeguard against unauthorized changes. In personal contexts, users can rely on this tool to protect important documents and track any alterations. This application is a powerful demonstration of Python's adaptability, showcasing how it can be leveraged with specialized libraries to create practical, efficient, and dependable software solutions for real-world challenges.

**10. FURTHER SCOPE OF THE PROJECT**

1.User Authentication and Authorization:

• Implement user accounts and authentication to ensure secure access to the application. This could involve username/password authentication or integration with OAuth providers like Google or GitHub.

2.File Type Filters:

• Allow users to specify specific file types (e.g., .txt, .pdf, .docx) they want to monitor. This can be achieved by extending the form to include a list of accepted file extensions.

3.Scheduled Monitoring:

• Provide the option for users to schedule specific times for file monitoring. This could be useful for scenarios where users only want to monitor files during specific hours of the day.

4.Support for Multiple Email Providers:

• Extend the email notification system to support multiple email providers beyond just Gmail, giving users more flexibility in how they receive notifications.

5.Integration with Cloud Storage:

• Allow users to monitor files stored in cloud platforms like Dropbox, Google Drive, or OneDrive. This could involve integrating with their APIs.

6.Logging and Auditing:

• Implement a logging system to record all monitoring activities. This can be useful for audit trails and troubleshooting purposes.

7.User Preferences and Settings:

• Provide users with the ability to customize settings such as notification frequency, notification format, and monitoring preferences.

8.Mobile Application:

• Develop a mobile application or responsive design to make the application accessible

on various devices.

**11. APPENDIX**

**11.1 Screenshots**

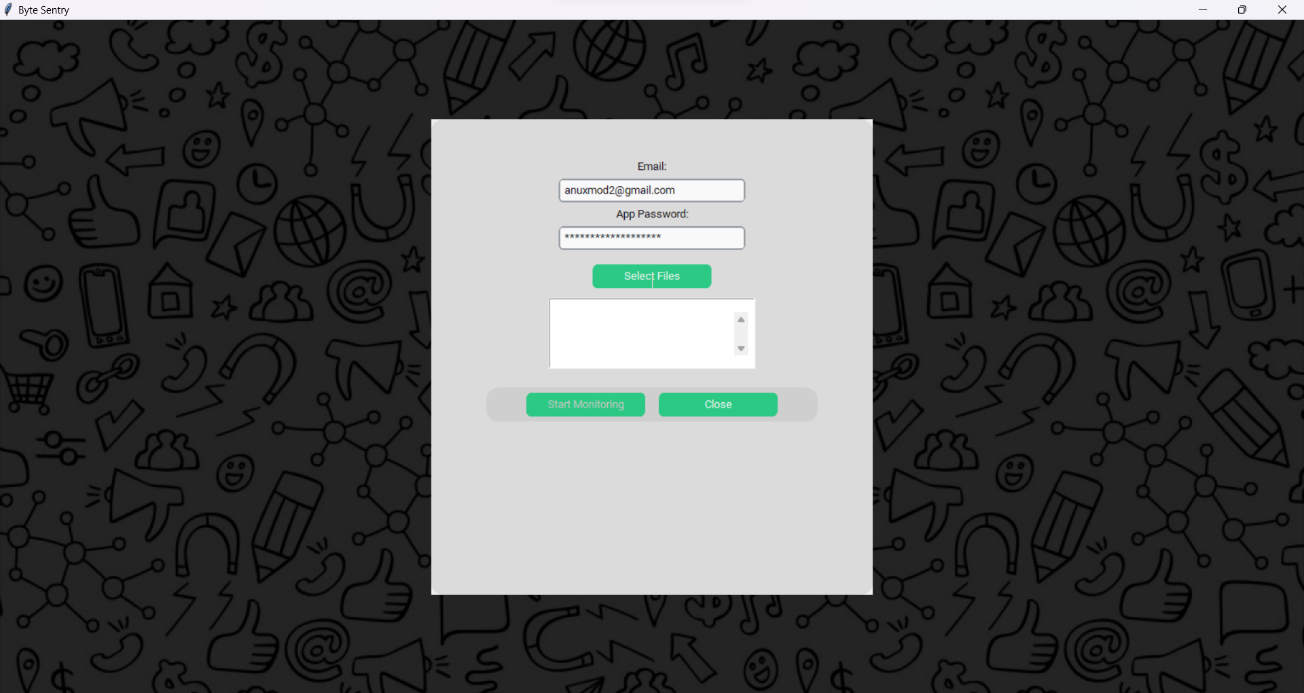
1. **Running the code in cmd**

A screenshot of a computer

Description automatically generated

1. **A screenshot of a computer

   Description automatically generatedGUI Interface**
2. **Entering Email & App password in GUI**

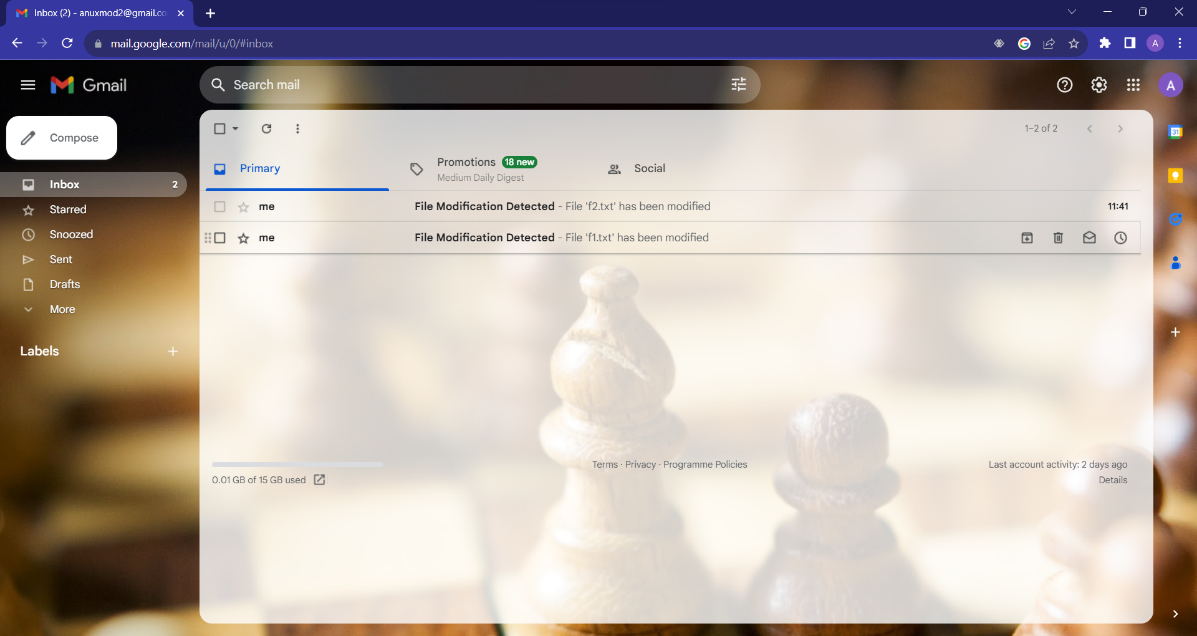


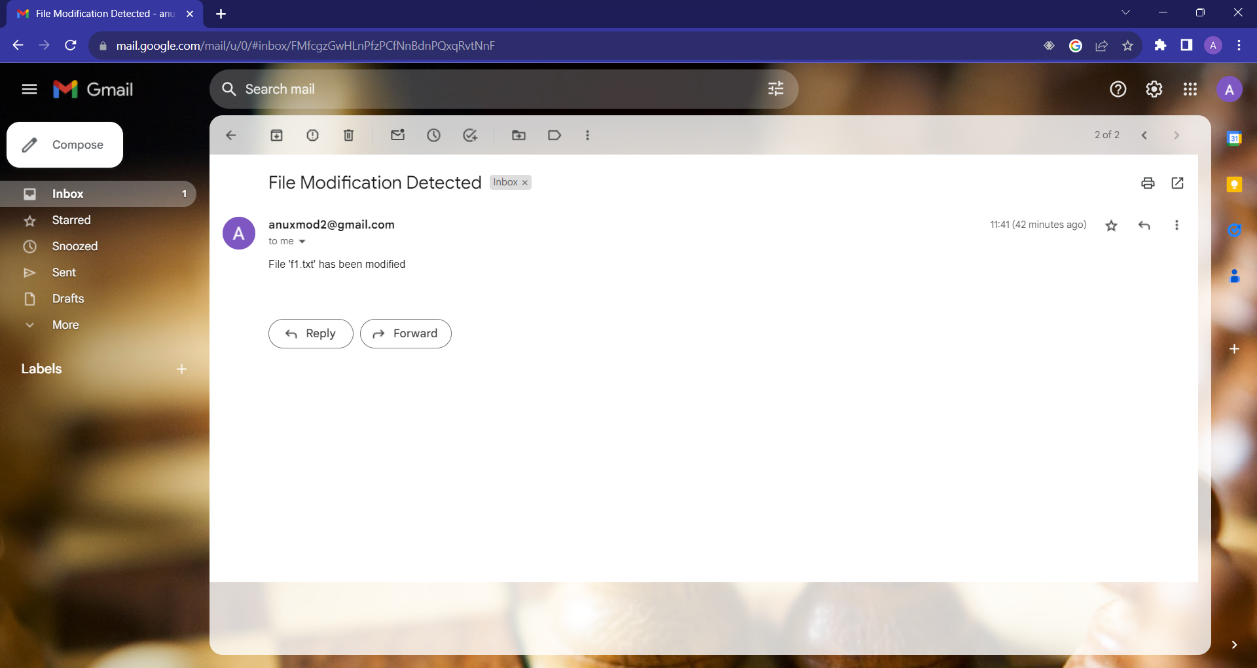
1. **Uploading files for monitoring**

**A screenshot of a computer

Description automatically generated**

1. **Email notification after detecting modification.**

****

****