CSCI 499 Senior Project Defense Documentation

Project: Windows Operating System & Top 20 Apps Vulnerability Scanner

April 2025

Student Info

Name: Dylan A. Kelly

Major: Computer Science

Degree: Bachelor of Arts- Applied Computing (Cybersecurity Concentration)

Project Advisor: Dr. Sean Hayes

Expected Graduation: May 2025

I. Statement of Purpose

The use of out-of-date Operating Systems or applications subjects the user to a number of security concerns due to the vulnerabilities present in an out-of-date system or application. The Windows Operating System and Top 20 Applications Vulnerability Scanner provides users with a simple, easy-to use application that checks the user's computer to see if their OS can be updated. The application also determines which of the top 20 most popular applications are installed on the user's computer and determines if there is a more up-to-date version available for them to download. After scanning the user's computer, the application sends its findings to the user in an email in order to remind them to update their computer or installed applications.

II. Research and Background

Background: Due to my degree being in cybersecurity, I wanted my senior project to reflect my academic focus. I felt as though creating a vulnerability scanner was an excellent crossroads between my study in cybersecurity concepts and programming. Regarding programming, the languages I have had the most experience with and am most familiar with are Java and C++. Relevant programming skills that I acquired prior to breaking ground on my project were from my Survey of Scripting Languages, Systems Analysis and Software Design, and Procedural Programming. I am not the most proficient programmer, and I quickly determined that my project would require some very specific functionality. While my project could have been coded in either Java or C++, I decided to learn Python in order to better address the functionality requirements for my project. Despite my lack of experience with programming in Python, I knew that it was a very user-friendly language, meaning it would be relatively easy to learn as I went. I also knew that Python had a multitude of libraries and was extremely versatile, both of which would allow me to more easily structure and adapt my program. For those reasons, I opted to learn Python and use it to program my senior project. My experience with vulnerability scanners is not as extensive. Therefore, I spent some time researching vulnerability scanners in order to get an accurate understanding of what would be needed to create one.

Research: My nature as a novice in Python programming meant that I needed to familiarize myself with the Python language. I also needed to put a specific emphasis on researching and familiarizing myself with the libraries that would allow me to web scrape and send emails, as both were vital to my project's functionality. I also needed a reference point for how my program should function and what that functionality would look like when applied. For this purpose, I used the program Grabber, which is a vulnerability scanner that was created using Python. Grabber was the initial inspiration for my project and I used it as a reference for what was possible for my project. The things that I had to research and learn about for the completion of my project were:

Python:

- Retrieving system data (using the platform library)
- Web scraping (using the BeautifulSoup library)
- Emailing (using the smtp library)
- Application GUI (using the tkinter library)
- Threading (using the threading library)

Website:

- How to locate relevant html tags
- How to retrieve the information within specific html tags

Computer:

- Determining application versions using PowerShell commands

III. Project Language(s), Software, and Hardware

Languages:

- Python
- HTML (I did not create any HTML code for my project. I pulled information from the HTML pages on CVE Details.)

Software:

- The platform, subprocess, json, requests, BeautifulSoup, smtplib, MIMEText, MIMEMultipart, MIMEBase, encoders, tkinter, sys, and threading libraries.

- Visual Studio Code

Hardware:

- Windows 10 Pro PC
- Windows 10 Pro laptop

IV. Project Requirements

Project Requirements: https://github.com/DylanAKelly/CSU-Senior-Project/blob/master/docs/SeniorProjectRequirementsDocument-DylanKelly.md

V. Project Implementation Description and Explanation

Source Code: https://github.com/DylanAKelly/CSU-Senior-Project/blob/master/src/VerOSTest.py

My project is a client-side vulnerability scanner that detects what Windows operating system is running on the user's device, as well as the installed version (if applicable) of the top 20 most downloaded apps on Windows computers. The program uses web scraping to pull vulnerability data from CVE Details, and then emails that information to the user based on the email address they provide. Once the program is launched, the user is met with the program's graphical user interface **(Figure 1)**. Once the graphical user interface window has appeared, the user is presented with three different buttons. The first button, labeled "Begin Scan", is light green in color and begins the vulnerability scan. The second button, labeled "Stop Program", is red in color and ends the program immediately once pressed. The third and final button, labeled

“Change Email”, is light blue in color and is positioned near the bottom right corner of the graphical user interface window. It is also positioned next to a textbox labeled “New Email”, where users can enter their email or update their email in order to receive the results of the vulnerability scan in their inbox once it is completed. Once users start the program for the first time, they should enter the email address that they would like the results of the vulnerability scan to be emailed to. After the user has entered their email and clicked the “Change Email” button, a text file called “VulSumUserEmail.txt” will appear on their computer. This text file stores the email address that the user would like the vulnerability summary to be sent to **(Figure 2)**. The preferred email can be updated at any time by running the program, entering a new email address into the “New Email” field, and pressing the “Change Email” button.

If the user forgets to add their email when using the program for the first time or deletes the contents of the “VulSumUserEmail.txt” file, they will encounter a popup explaining the error and what they should do **(Figure 3a)**. If the user attempts to update their email by inputting a new email into the “New Email” field, but presses the “Change Email” button without entering any text, they will encounter a popup explaining that they did not enter an email into the field, and that they should now do so at this time **(Figure 3b)**. If the user inputs an invalid email address or input, they will be met with a popup that informs them of the issue and suggests what steps they should take to correct it **(Figure 3c)**. When the user successfully changes their email, they will encounter a popup informing them that their email was changed, and the popup will display their new email **(Figure 3d)**.

After the user presses the “Begin Scan” button, the vulnerability scan will start running, which is conveyed to the user by a progress bar beginning to move **(Figure 4)**. Once the user enters a valid email and the vulnerability scan completes, they will be shown a popup that informs them the scan completed successfully and that they can check their inbox to see the results of the scan and close or stop the program **(Figure 5)**. Once the user logs into their email, they will see an email from top20appsvulnsum@gmail.com that consists of three text files **(Figure 6)**. These text files contain information on the user's current Windows operating system (operatingsystem.txt), what versions, if any, of the top 20 applications they have installed (appversions.txt), and the vulnerabilities present in those versions of the top 20 applications (vulnerabilities.txt) **(Figures 7-9)**.

Figure 1. Graphical User Interface

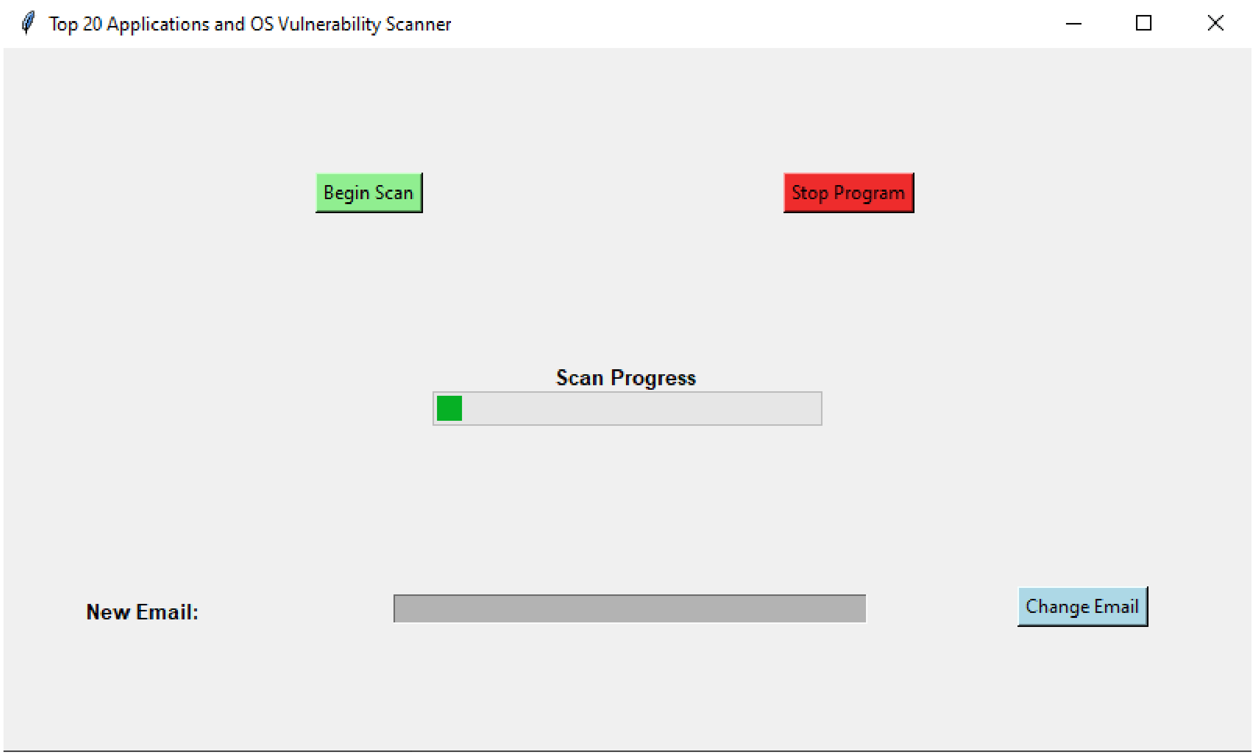
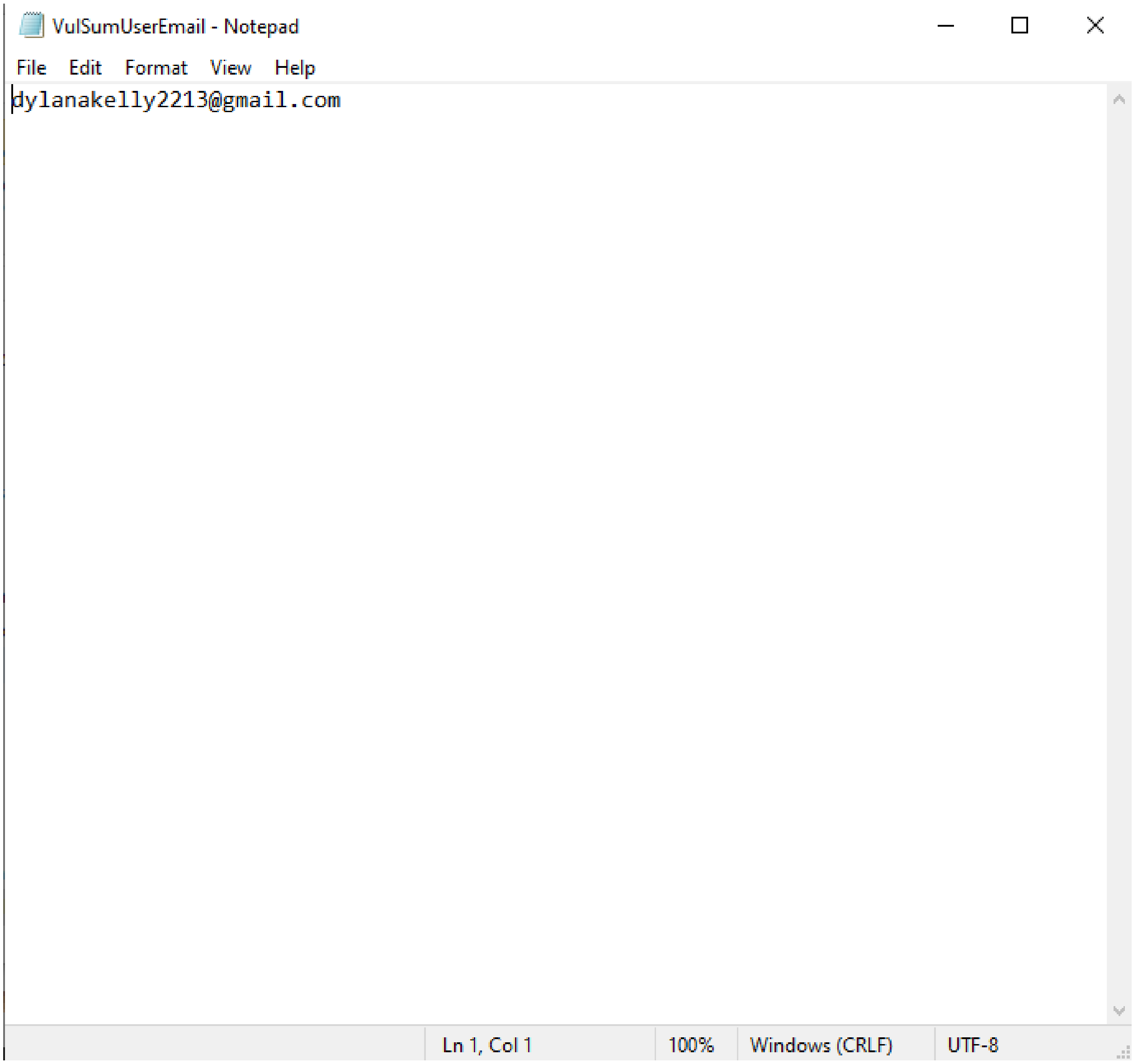
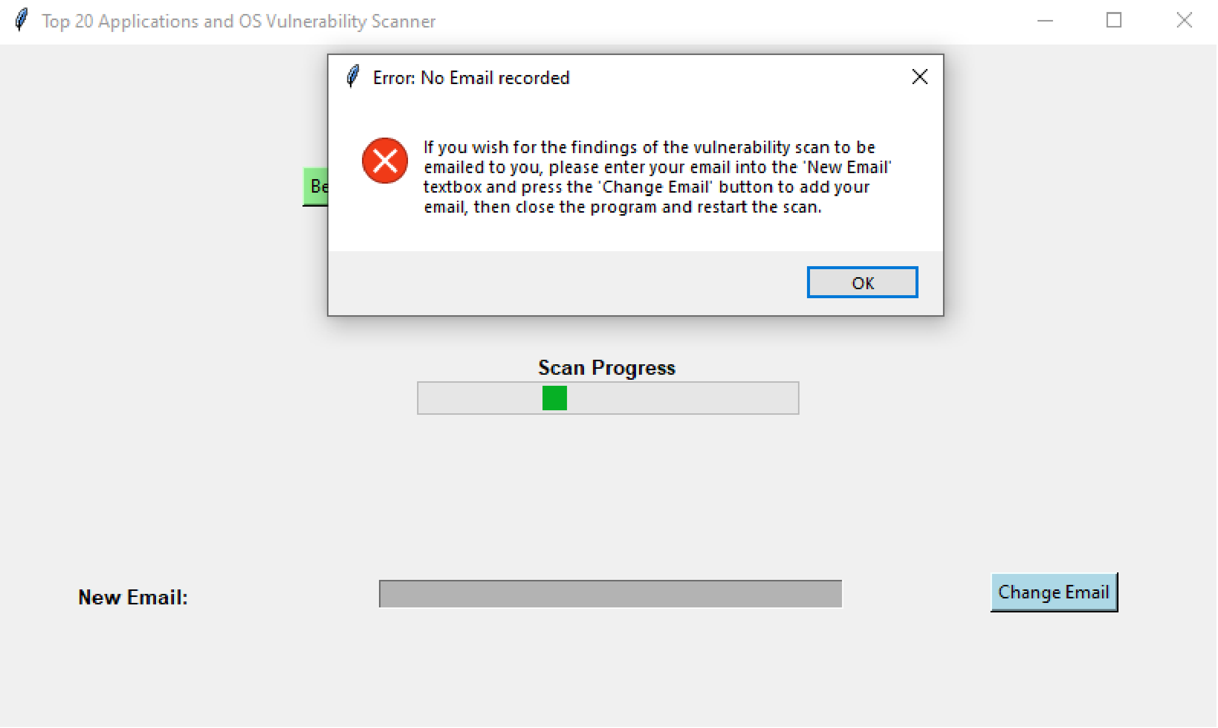


Figure 2. Stored email



```
dylanakelly2213@gmail.com
```

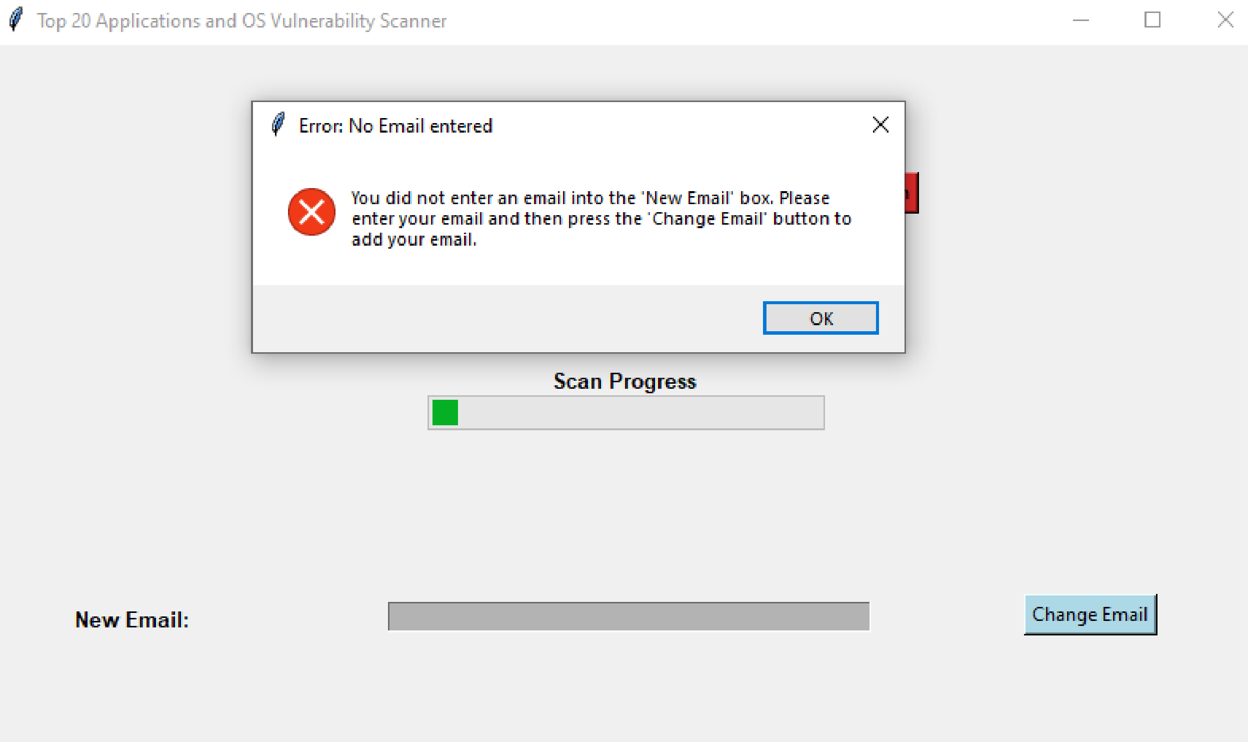
Figure 3a. Error prompting user to enter an email into the "New Email" textbox



Error: No Email recorded

If you wish for the findings of the vulnerability scan to be emailed to you, please enter your email into the 'New Email' textbox and press the 'Change Email' button to add your email, then close the program and restart the scan.

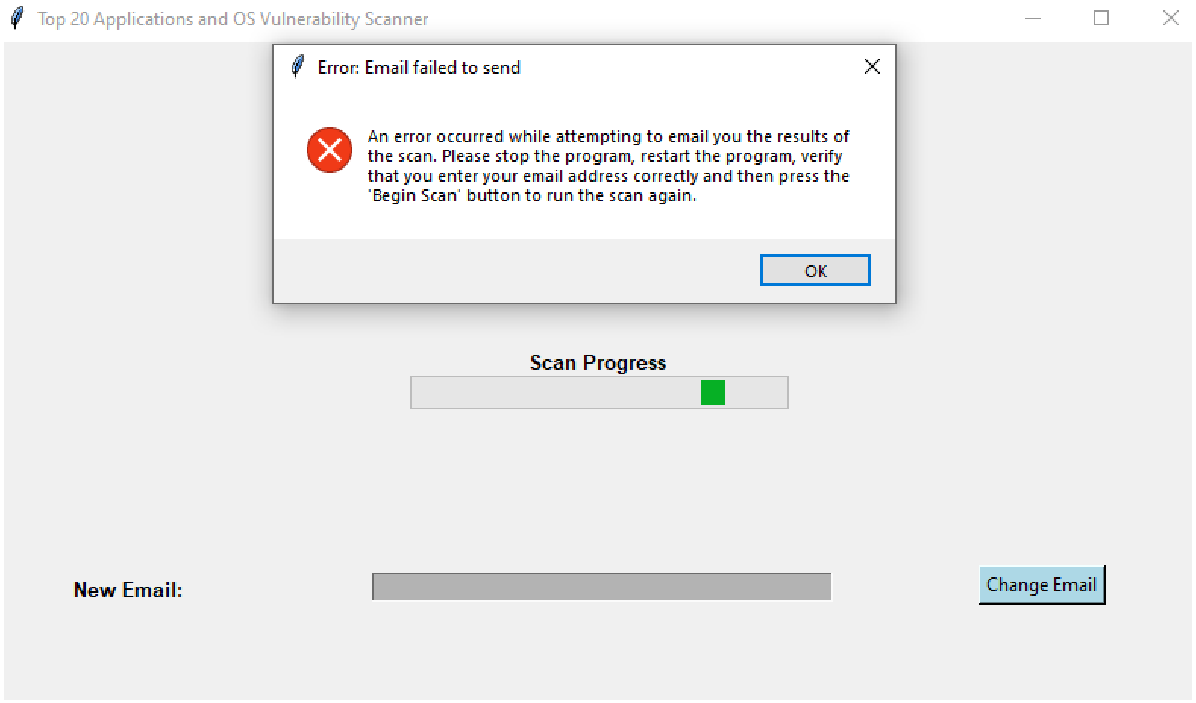
Figure 3b. Error informing the user they did not enter anything into the "New Email" textbox



Error: No Email entered

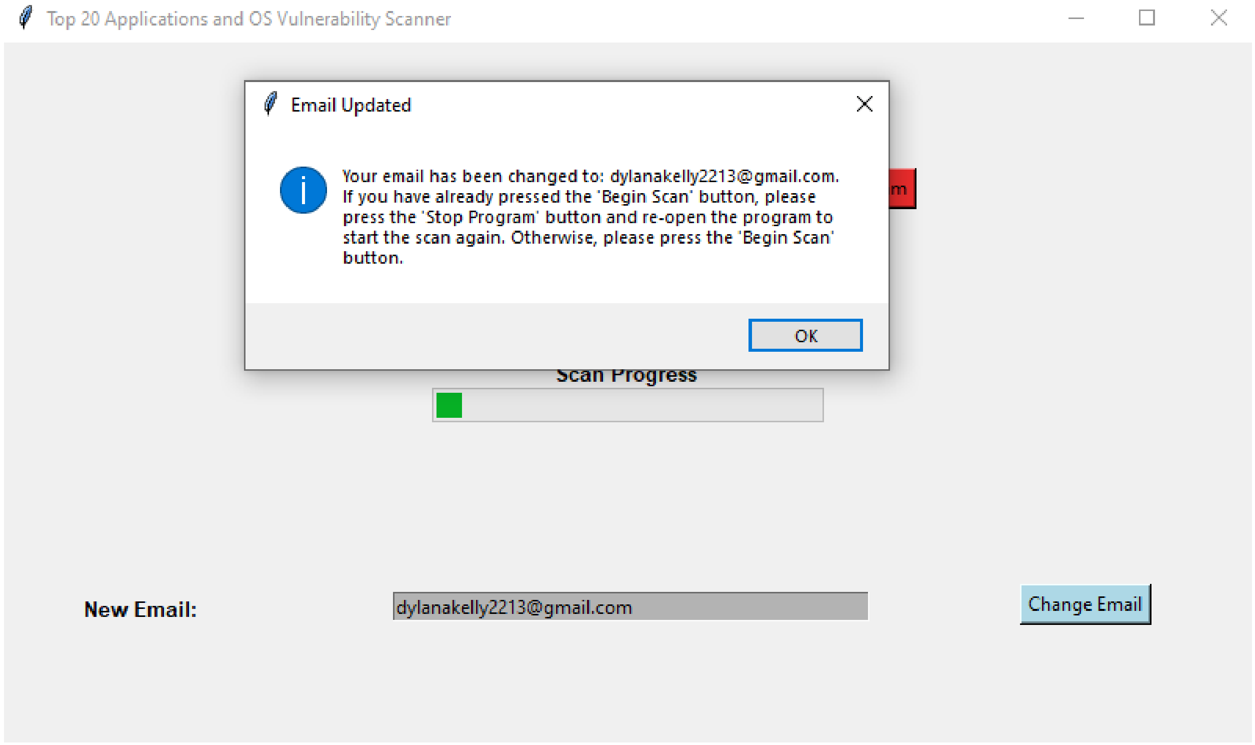
You did not enter an email into the 'New Email' box. Please enter your email and then press the 'Change Email' button to add your email.

Figure 3c. Error informing user that they entered an invalid email address


Top 20 Applications and OS Vulnerability Scanner
Error: Email failed to send
OK
Scan Progress
New Email:
Change Email

An error occurred while attempting to email you the results of the scan. Please stop the program, restart the program, verify that you enter your email address correctly and then press the 'Begin Scan' button to run the scan again.

Figure 3d. Popup informing the user that their email was successfully changed



Your email has been changed to: dylanakelly2213@gmail.com. If you have already pressed the 'Begin Scan' button, please press the 'Stop Program' button and re-open the program to start the scan again. Otherwise, please press the 'Begin Scan' button.

New Email: dylanakelly2213@gmail.com

Figure 4. Progress bar moving, indicating that the vulnerability scan is running

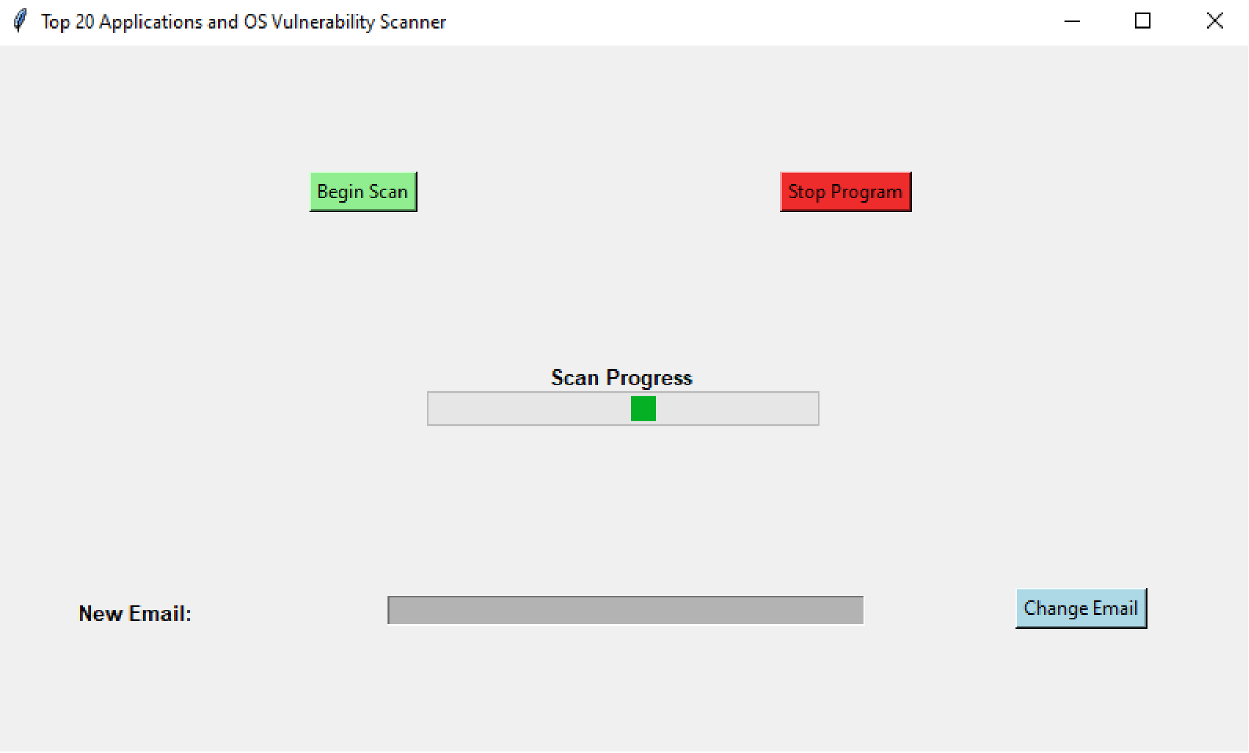
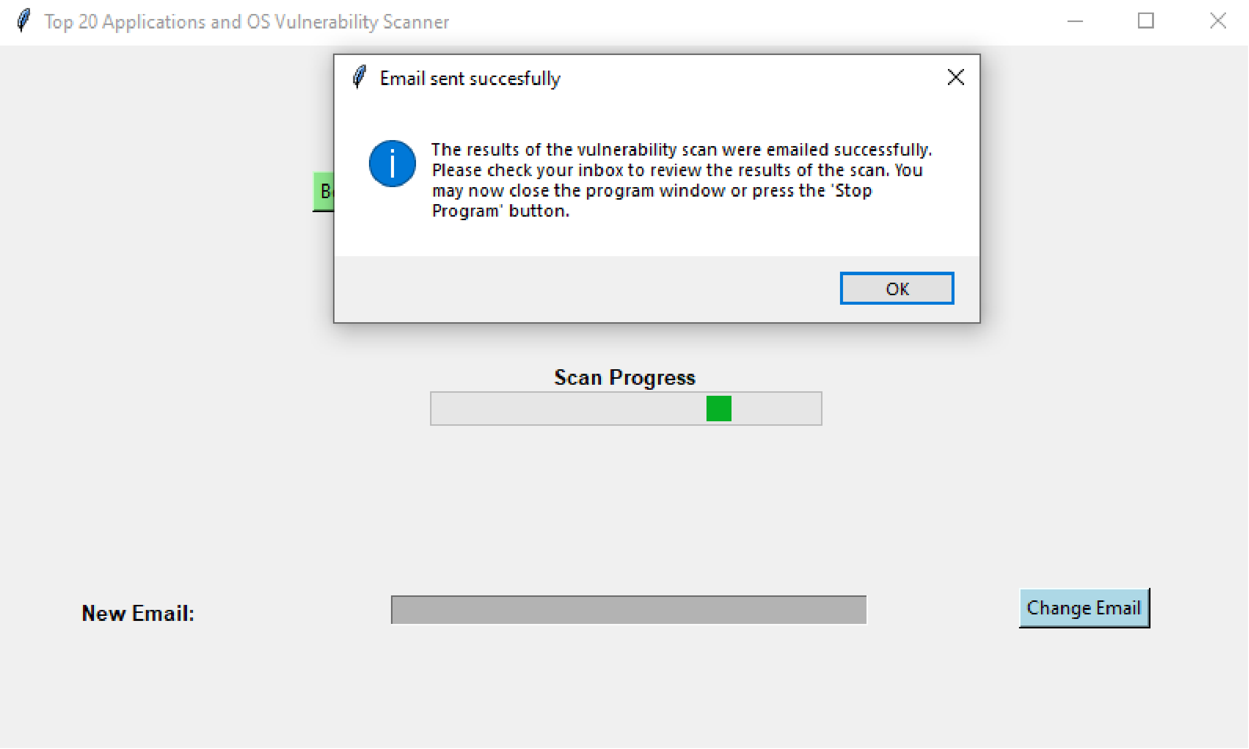


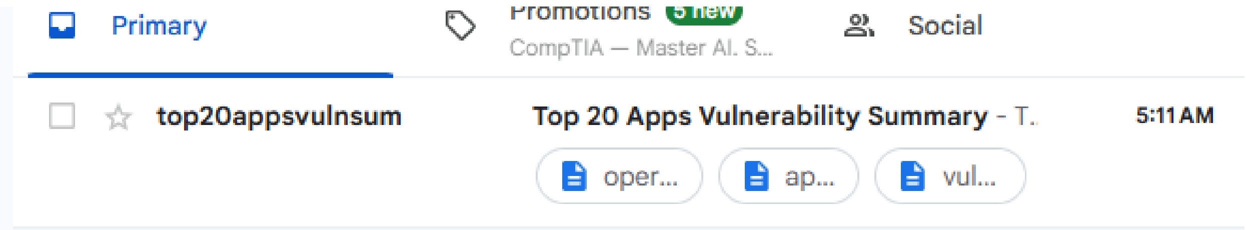
Figure 5.Popup explaining that the scan is complete and the results are in the user's inbox



Email sent succesfully

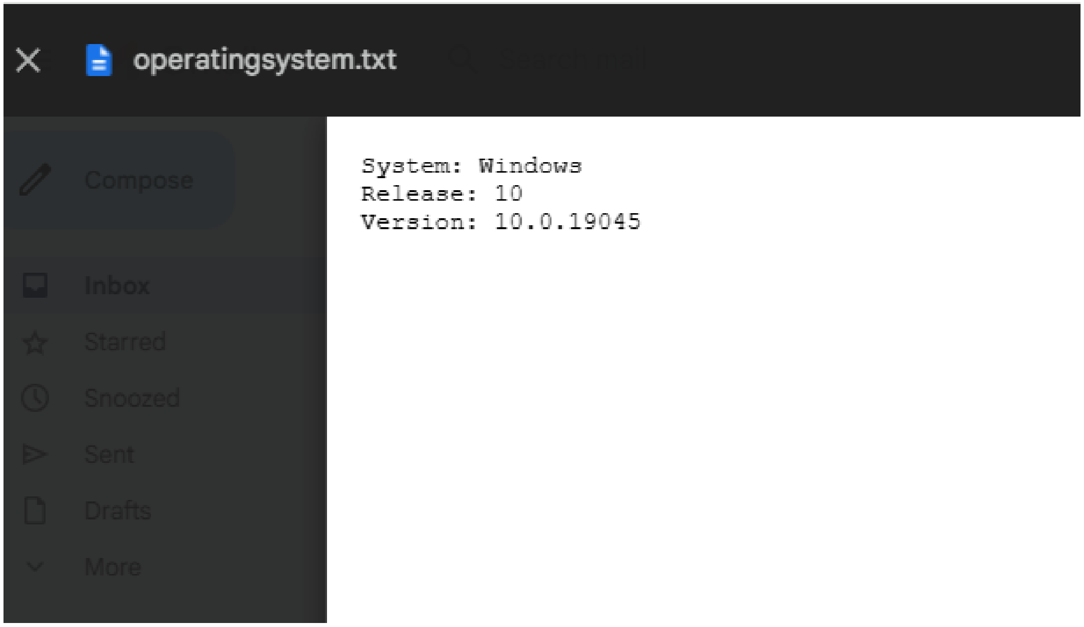
The results of the vulnerability scan were emailed successfully. Please check your inbox to review the results of the scan. You may now close the program window or press the 'Stop Program' button.

Figure 6.User has received an email fromtop20appsvulnsum@gmail.com containing scan results



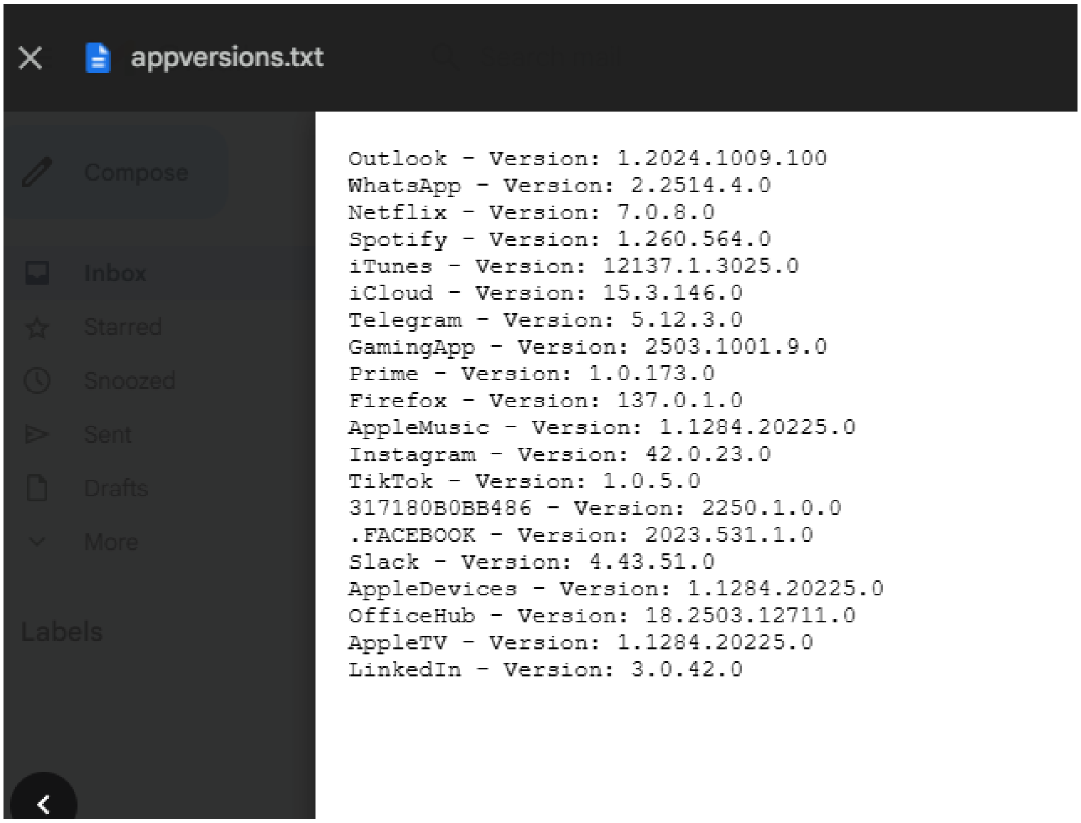
top20appsvulnsum — Top 20 Apps Vulnerability Summary - T. — 5:11 AM

Figure 7. Text file containing information on the user's Windows Operating System



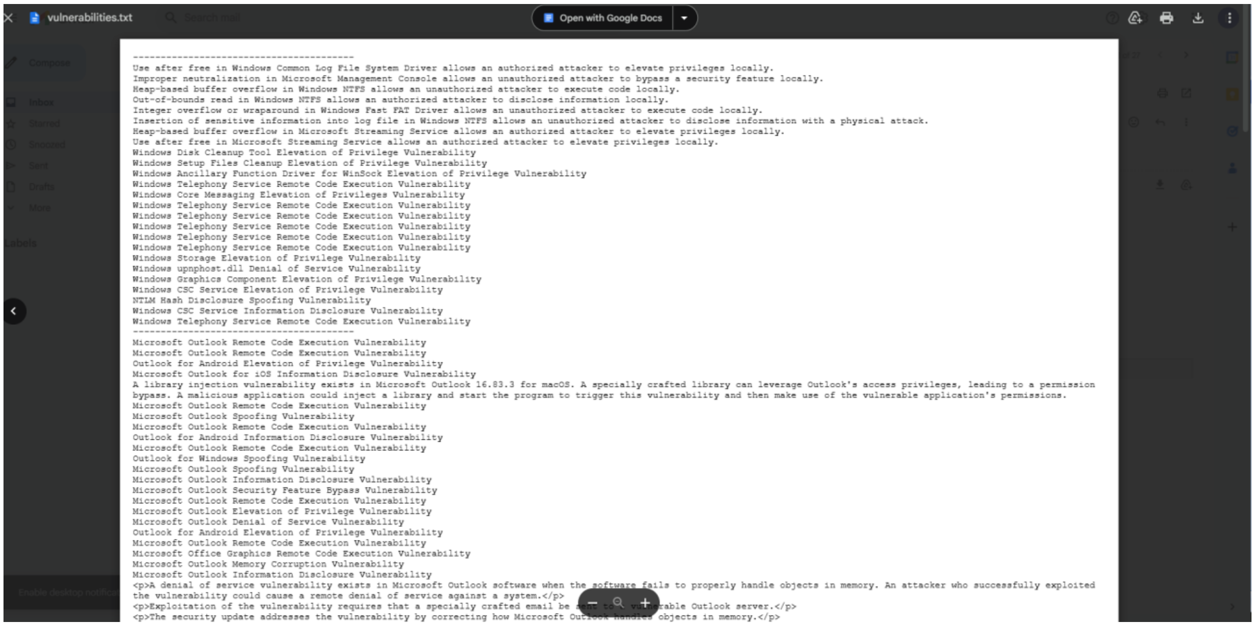
```
System: Windows
Release: 10
Version: 10.0.19045
```

Figure 8. Text file containing information on the installed Top 20 apps



```
Outlook - Version: 1.2024.1009.100
WhatsApp - Version: 2.2514.4.0
Netflix - Version: 7.0.8.0
Spotify - Version: 1.260.564.0
iTunes - Version: 12137.1.3025.0
iCloud - Version: 15.3.146.0
Telegram - Version: 5.12.3.0
GamingApp - Version: 2503.1001.9.0
Prime - Version: 1.0.173.0
Firefox - Version: 137.0.1.0
AppleMusic - Version: 1.1284.20225.0
Instagram - Version: 42.0.23.0
TikTok - Version: 1.0.5.0
317180B0BB486 - Version: 2250.1.0.0
.FACEBOOK - Version: 2023.531.1.0
Slack - Version: 4.43.51.0
AppleDevices - Version: 1.1284.20225.0
OfficeHub - Version: 18.2503.12711.0
AppleTV - Version: 1.1284.20225.0
LinkedIn - Version: 3.0.42.0
```

Figure 9. Text file containing information on the potential vulnerabilities



```
----------------------------------------
Use after free in Windows Common Log File System Driver allows an authorized attacker to elevate privileges locally.
Improper neutralization in Microsoft Management Console allows an unauthorized attacker to bypass a security feature locally.
Heap-based buffer overflow in Windows NTFS allows an unauthorized attacker to execute code locally.
Out-of-bounds read in Windows NTFS allows an authorized attacker to disclose information locally.
Integer overflow or wraparound in Windows Fast FAT Driver allows an unauthorized attacker to execute code locally.
Insertion of sensitive information into log file in Windows NTFS allows an unauthorized attacker to disclose information with a physical attack.
Heap-based buffer overflow in Microsoft Streaming Service allows an authorized attacker to elevate privileges locally.
Use after free in Microsoft Streaming Service allows an authorized attacker to elevate privileges locally.
Windows Disk Cleanup Tool Elevation of Privilege Vulnerability
Windows Setup Files Cleanup Elevation of Privilege Vulnerability
Windows Ancillary Function Driver for WinSock Elevation of Privilege Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
Windows Core Messaging Elevation of Privileges Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
Windows Storage Elevation of Privilege Vulnerability
Windows upnphost.dll Denial of Service Vulnerability
Windows Graphics Component Elevation of Privilege Vulnerability
Windows CSC Service Elevation of Privilege Vulnerability
NTLM Hash Disclosure Spoofing Vulnerability
Windows CSC Service Information Disclosure Vulnerability
Windows Telephony Service Remote Code Execution Vulnerability
----------------------------------------
Microsoft Outlook Remote Code Execution Vulnerability
Microsoft Outlook Remote Code Execution Vulnerability
Outlook for Android Elevation of Privilege Vulnerability
Microsoft Outlook for iOS Information Disclosure Vulnerability
A library injection vulnerability exists in Microsoft Outlook 16.83.3 for macOS. A specially crafted library can leverage Outlook's access privileges, leading to a permission
bypass. A malicious application could inject a library and start the program to trigger this vulnerability and then make use of the vulnerable application's permissions.
Microsoft Outlook Remote Code Execution Vulnerability
Microsoft Outlook Spoofing Vulnerability
Microsoft Outlook Remote Code Execution Vulnerability
Outlook for Android Information Disclosure Vulnerability
Microsoft Outlook Remote Code Execution Vulnerability
Outlook for Windows Spoofing Vulnerability
Microsoft Outlook Spoofing Vulnerability
Microsoft Outlook Information Disclosure Vulnerability
Microsoft Outlook Security Feature Bypass Vulnerability
Microsoft Outlook Remote Code Execution Vulnerability
Microsoft Outlook Elevation of Privilege Vulnerability
Microsoft Outlook Denial of Service Vulnerability
Outlook for Android Elevation of Privilege Vulnerability
Microsoft Outlook Remote Code Execution Vulnerability
Microsoft Office Graphics Remote Code Execution Vulnerability
Microsoft Outlook Memory Corruption Vulnerability
Microsoft Outlook Information Disclosure Vulnerability
<p>A denial of service vulnerability exists in Microsoft Outlook software when the software fails to properly handle objects in memory. An attacker who successfully exploited
the vulnerability could cause a remote denial of service against a system.</p>
<p>Exploitation of the vulnerability requires that a specially crafted email be sent to a vulnerable Outlook server.</p>
<p>The security update addresses the vulnerability by correcting how Microsoft Outlook handles objects in memory.</p>
```

VI. Test Plan

Test Plan: https://github.com/DylanAKelly/CSU-Senior-Project/blob/master/docs/TestPlan-DylanKelly.md

VII. Test Plan Results

Results Overview: The test plan for my program was broken down into two parts, manual testing and user acceptance testing. I decided to break testing down into these two categories for a few reasons. First, I felt that manual testing would prove to be more efficient due to the nature of my program and the nature of the tests that I knew I would need to perform. Second, due to my program being designed to help users who might not be as technologically inclined as individuals in computer science fields, I wanted to ensure that my program would be easy to understand and use. As such, I determined that the best way to verify my program was easy to understand and use was to have some users test it and provide feedback in the form of a Google Form survey. Finally, I determined that creating automated test cases would not be time-efficient for this particular project.

Results: Of my twenty-two test cases, seventeen of them were successful based on the criteria I set when writing said test cases. Out of the fourteen manual tests, nine of them were successful. All eight of the user acceptance tests can be considered successful, though the results do show room for improvement.

Manual Tests:

Manual Test 1 – Program successfully displays the current version of the installed Windows Operating System. This information is written to a text file called “oepratingsystem.txt” and is emailed to the user after the scan completes.

Manual Test 2 – Program successfully displays the current version of the installed top 20 application. This information is written to a text file called “appversions.txt” and is emailed to the user after the scan completes.

Manual Test 5 – Program successfully accesses the website CVE Details and determines what vulnerabilities are present on the Windows Operating System version.

Manual Test 6 – Program successfully accesses the website CVE Details and determines what vulnerabilities are present on the installed top 20 application version.

Manual Test 8 – Program successfully sends an email to the user containing their Windows OS version, the versions of installed top 20 apps, and the vulnerabilities present in their version of the Windows Operating System and installed top 20 apps.

Manual Test 9 – Program successfully retrieves Windows Operating System vulnerability information from CVE Details and writes that information to a text file called “vulnerabilities.txt”, which is emailed to the user after the scan completes.

Manual Test 11 – Program is written in such a way that its structure is consistent and can be easily maintained and improved upon.

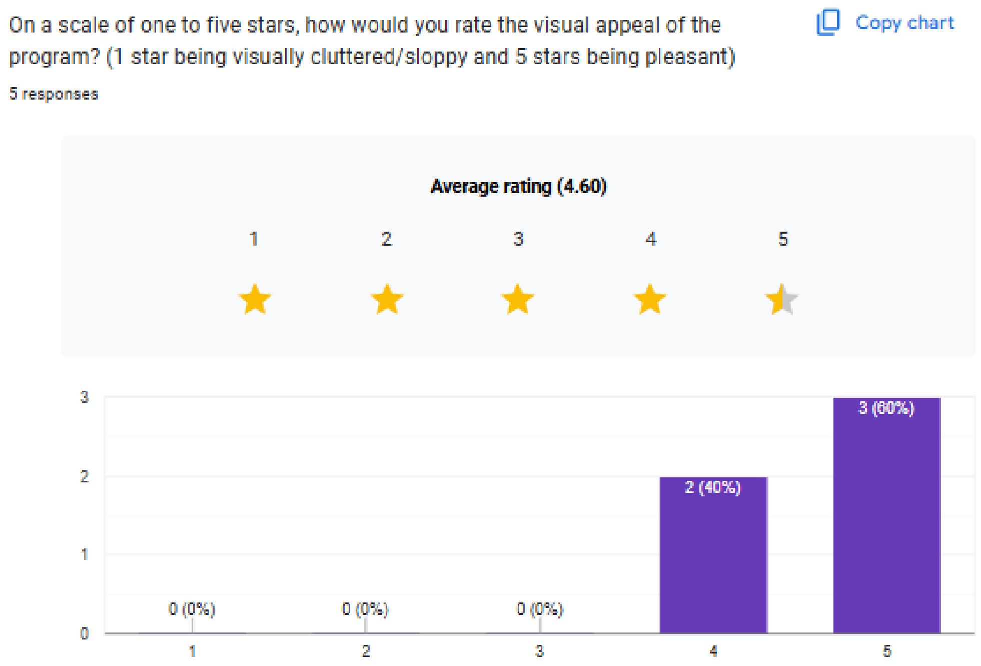
Manual Test 12 – Program does not request or access any other information stored on the user’s computer outside of the user’s email address (which the user must enter themselves) and the installed Windows OS version and the versions of any of the top 20 applications installed on the computer.

Manual Test 14 – Program complies with industry ethical standards. More information on this test case can be found in my Ethics Self-Assessment document in my GitHub Repository. The link to this document can be found below

Ethics Self-Assessment: https://github.com/DylanAKelly/CSU-Senior-Project/blob/master/docs/EthicsSelfAssessment.docx

User Acceptance Tests: The user acceptance tests were conducted using five people. Four of my five survey participants were less technologically knowledgeable. This is because I wanted to make sure that my program could be used and understood by people who might not know too much about computers. The fifth participant was fairly knowledgeable about computers. Each person was instructed to download my program and run it. Once they had experimented with using my program, I sent them a link to a Google Form that consisted of eight questions. These questions were designed to gauge each user's satisfaction with my program and its capabilities and features. Each question required an average of four (out of five) stars in order for the test case to be considered passed. The results of those surveys are listed below.

Question 1: How would you rate the visual appeal of the program?

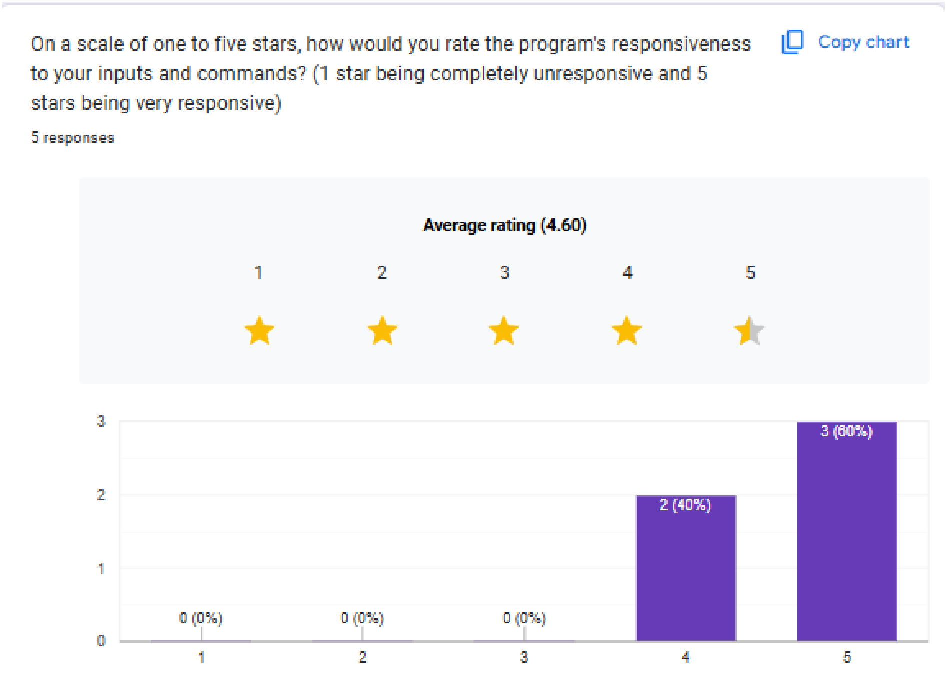


On a scale of one to five stars, how would you rate the visual appeal of the program? (1 star being visually cluttered/sloppy and 5 stars being pleasant)

5 responses

Average rating (4.60)

Question 2: How would you rate the program’s responsiveness to your commands?

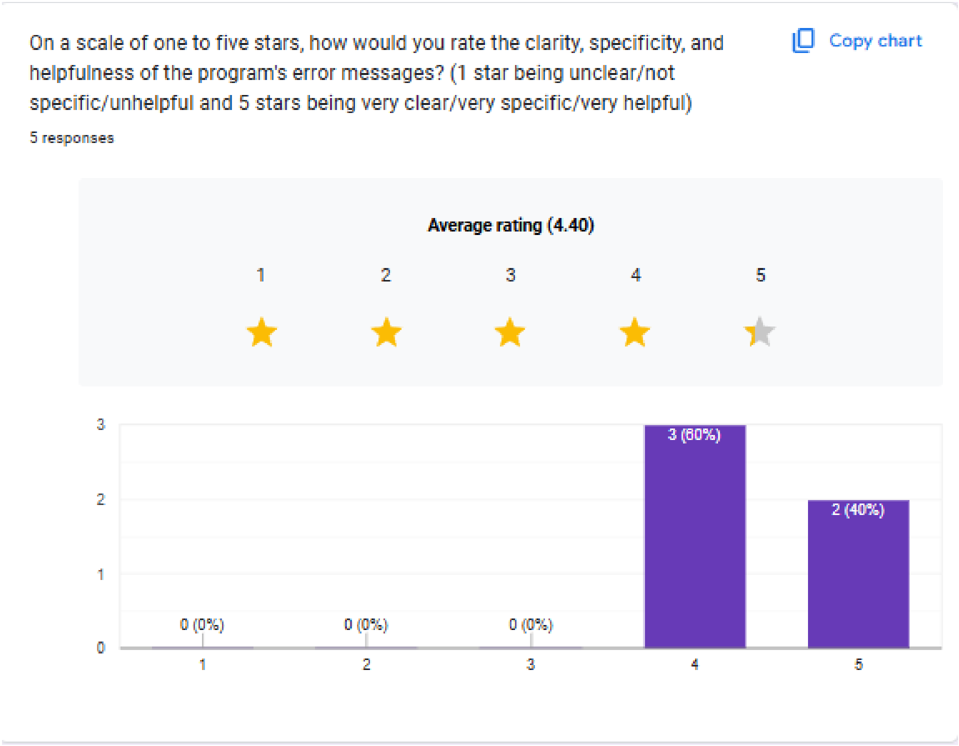


On a scale of one to five stars, how would you rate the program's responsiveness to your inputs and commands? (1 star being completely unresponsive and 5 stars being very responsive)

5 responses

Average rating (4.60)

Question 3: How would you rate the clarity, specificity, and helpfulness of the error messages?

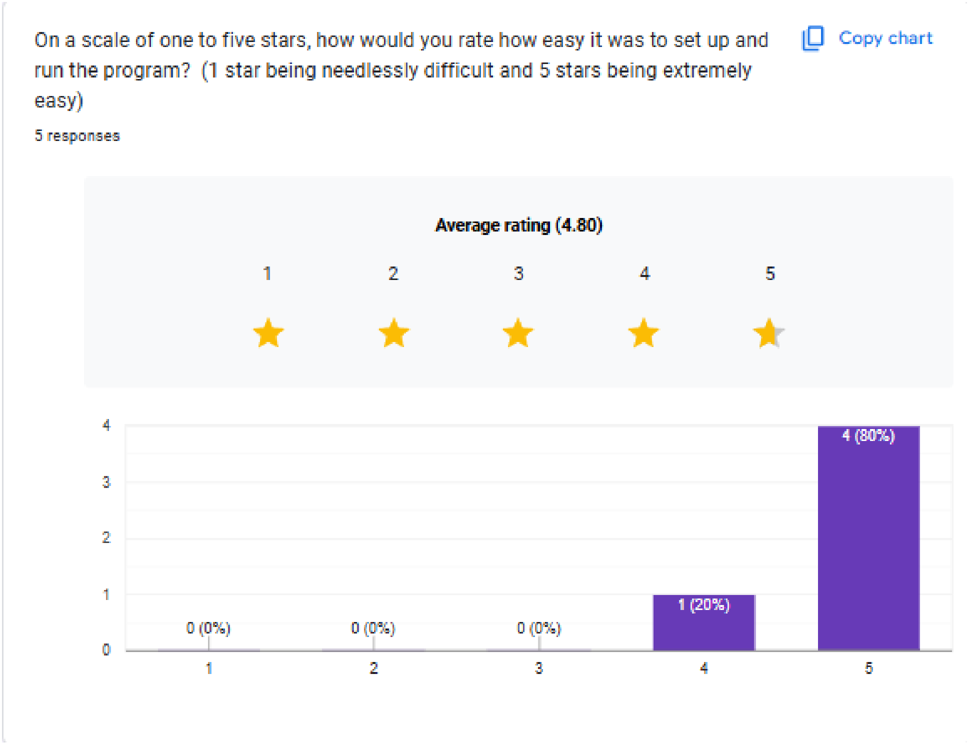


On a scale of one to five stars, how would you rate the clarity, specificity, and helpfulness of the program's error messages? (1 star being unclear/not specific/unhelpful and 5 stars being very clear/very specific/very helpful)

5 responses

Average rating (4.40)

Question 4: How would you rate how easy the program is to set up and run?

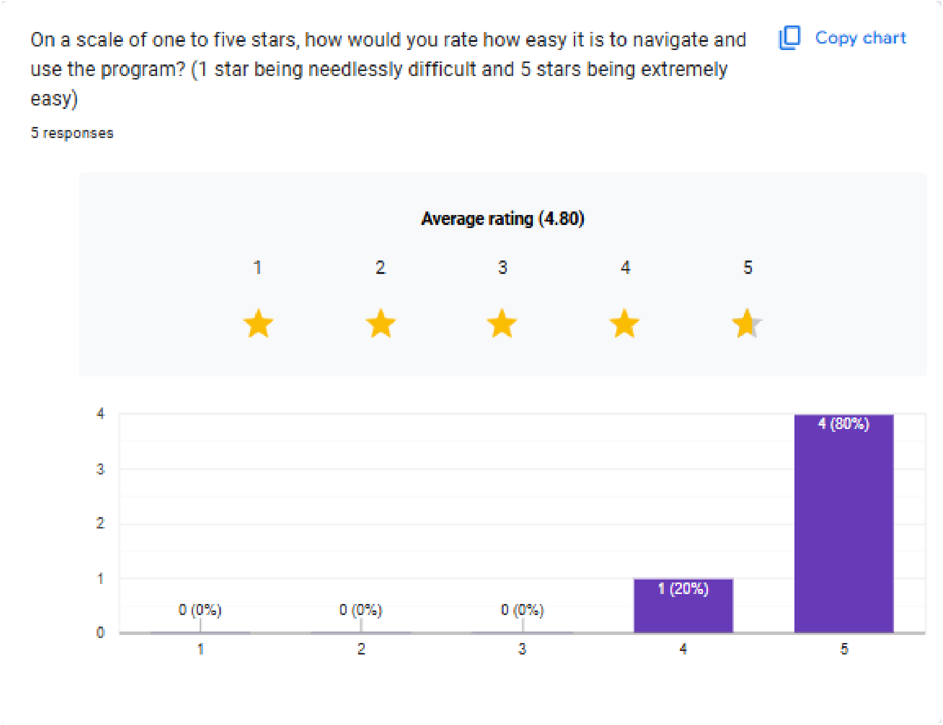


On a scale of one to five stars, how would you rate how easy it was to set up and run the program? (1 star being needlessly difficult and 5 stars being extremely easy)

5 responses

Average rating (4.80)

Question 5: How would you rate how easy it is to navigate and use the program?

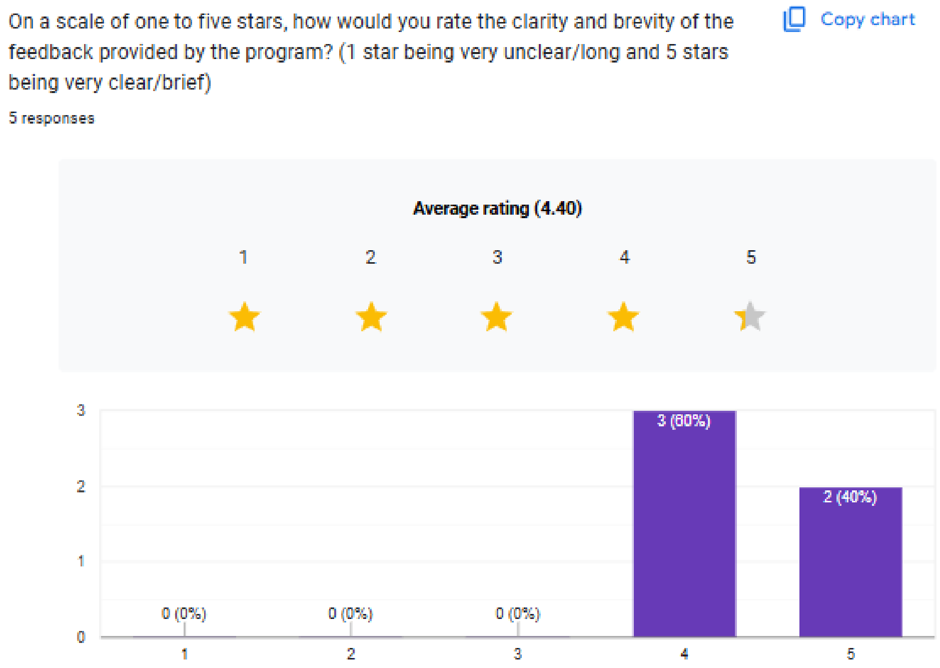


On a scale of one to five stars, how would you rate how easy it is to navigate and use the program? (1 star being needlessly difficult and 5 stars being extremely easy)

5 responses

Average rating (4.80)

Question 6: How would you rate the clarity & brevity of the program's feedback?

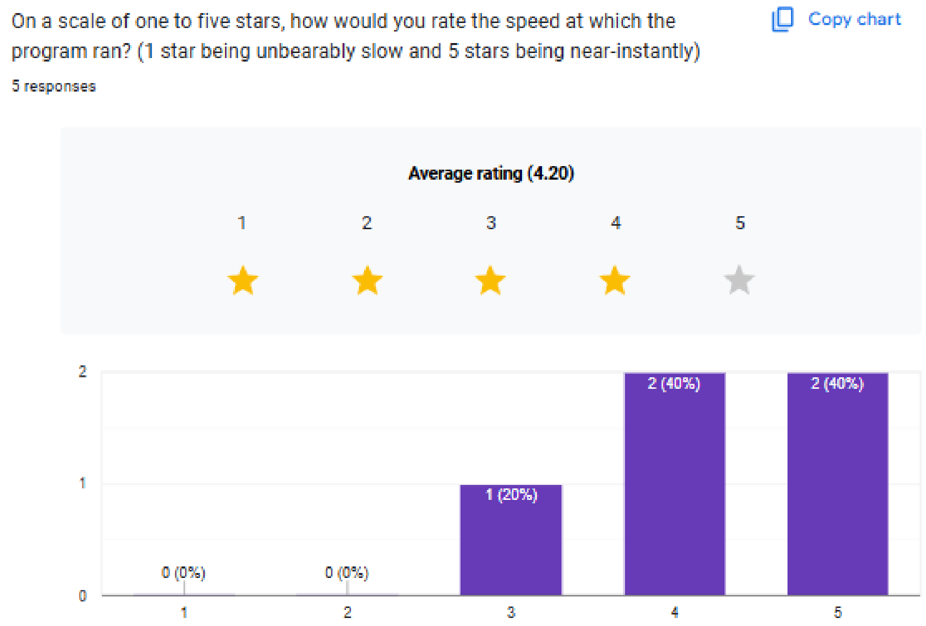


On a scale of one to five stars, how would you rate the clarity and brevity of the feedback provided by the program? (1 star being very unclear/long and 5 stars being very clear/brief)

5 responses

Average rating (4.40)

Question 7: How would you rate the speed at which the program ran?

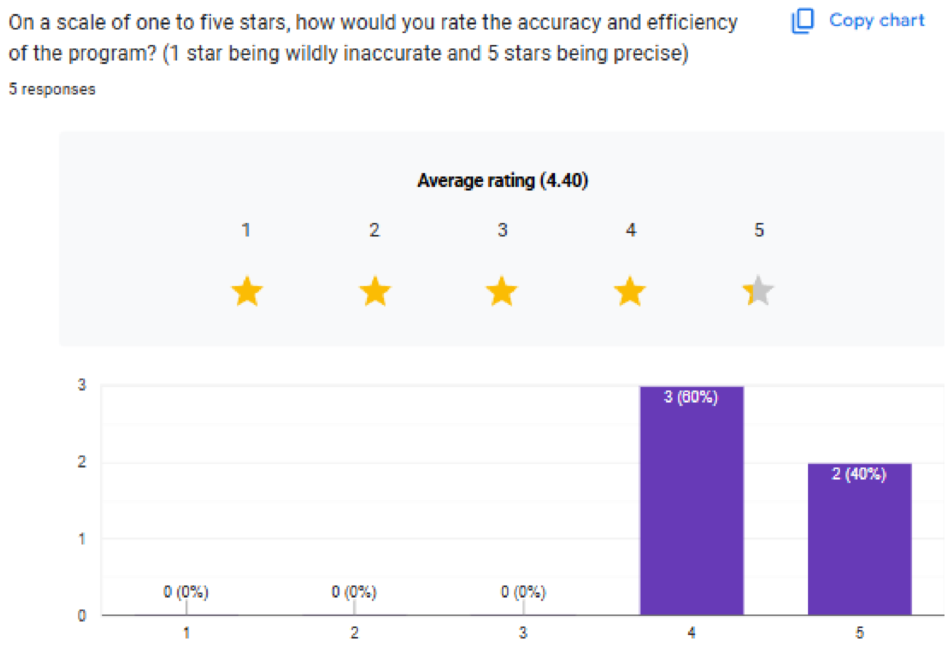


On a scale of one to five stars, how would you rate the speed at which the program ran? (1 star being unbearably slow and 5 stars being near-instantly)

5 responses

Average rating (4.20)

Question 8: How would you rate the accuracy and efficiency of the program?



On a scale of one to five stars, how would you rate the accuracy and efficiency of the program? (1 star being wildly inaccurate and 5 stars being precise)

5 responses

Average rating (4.40)

Survey Takeaways: Generally speaking, the users who I had test the program and take the survey were happy with how it looked, felt, and ran. Survey question #7, “How would you rate the speed at which the program ran?” is the question with the lowest average. As one might expect, the survey participant that was the most experienced with computers was the one who rated question seven with three stars. This did not surprise me, as my program currently takes roughly two to three minutes to run. This will be addressed in greater detail in the “Future Improvements” section. While these survey scores are satisfactory and suggest that my program is acceptable for users, there are considerations, changes, and improvements that could be made which might result in an increase to those scores were I to survey the users again.

VIII. Challenges Overcome

While designing and creating my program, there were a number of challenges that arose and had to be dealt with. Most of the issues I faced during this process related to learning the functionality of different Python libraries and figuring out how to incorporate those libraries into my program. Some of the more daunting challenges faced and how they were handled were:

Learning Python

While I did already have some experience with Python, it was surface-level knowledge. I knew going in that using Python to code my vulnerability scanner would require me to relearn most of the basics. I also knew that I would need to familiarize myself with Python to the degree that I would be familiar with and understand some of the more unique aspects of the language. This was not particularly difficult; however, it did take a considerable amount of time.

Using BeautifulSoup to web scrape vulnerability information from the CVE Details website

Due to the fact that I lacked experience with both Python and web scraping, figuring out how to use the BeautifulSoup library was extremely important. It took a few days of watching videos, reading articles about BeautifulSoup & web scraping, and referencing the Grabber source code before I began to understand how web scraping in Python worked. Once I had started to understand the 'how' of web scraping, I ran into another challenge. I needed to look at each of the relevant CVE Details html pages and determine which HTML tags I needed to pull information from. This took less time than figuring out how to web scrape, as I simply needed to look for the vulnerability descriptions in the HTML page and record what tags contained that information. I also had to try and determine if CVE Details changed the format or structure of their HTML pages. If they did, I would have needed to figure out a different way to gather the relevant vulnerability info. Fortunately, CVE Details does not seem to change the structure of their HTML often, if they do at all.

Figuring out how to send the user an email containing the vulnerability scan results

Figuring out how to send the user an email was also a somewhat difficult process. I had to learn how to use the smtplib, Multipurpose Internet Mail Extension (MIME) text, multipart, & base, and encoders libraries. This took a couple of days, as I needed to figure out what parts of each library was needed in order to send the user an email, and I needed to figure out how to add the text files as attachments to that email. I also struggled with logging into the program's email using the code I had written, as it was not accepting the password for that account. I had to do some troubleshooting to resolve this and eventually got it to work by using a passkey.

Creating and maintaining the Graphical User Interface

As previously mentioned, I had fairly limited experience with Python. This meant that I had not created a Graphical User Interface in Python before, so I had to learn how to do that from scratch. This also took a couple of days, primarily due to the features that I wanted the GUI to have. There were multiple different ways that I could have constructed the GUI, but I opted to go with the tkinter library for a few reasons. First, the tkinter library is easy to use, so I did not run into too much trouble while figuring out how to create the GUI. Second, the library is rather extensive, meaning it has most features that would be needed for a simple GUI. Finally, the library is usable on the vast majority of operating systems, so there was no issue in getting it to work on Windows devices. After I had got the basics of the GUI set up, I ran into some issues. The two most notable issues were the progress bar not moving while the program was running and the "Stop Program" button causing the program to not respond. I solved the former by creating a thread using the threading library, which enabled the progress bar to visually move while the program and scan were running. The latter issue was fixed by creating a function that stopped the progress bar.

IX. Future Enhancements

While reviewing my test case results and completing my documentation, and even before, I was aware of areas in my program that could be improved upon. In some instances, these improvements are rather significant. I am proud of the program I have made, but there are many improvements that could be made. While I could list potential improvements for a while, I will discuss a few of the most impactful improvements that I have thought of so far.

Implement a database

I believe that implementing a database to store information would be the single most impactful improvement I could make. As mentioned in my ethics self-assessment document, implementing a database element to my program would make the user's data much safer, as it could be encrypted at rest. This would also allow me to store vulnerability information and the links to the CVE Details pages much easier. I could also store the program's email account and password in the database, which would make it much more secure.

Make the use of PowerShell more efficient

In order to determine the version (if any) of the top 20 installed apps present on the user's computer, I have my program run a PowerShell command. The issue is the PowerShell command gets the app versions one at a time. This inefficiency is the reason that my program takes a few minutes to run to completion. Adjusting the PowerShell script so that it checks for the version of all top 20 apps at once instead of one at a time would increase the program's speed considerably.

Splitting the program’s functionality

Currently, my entire program runs once you press the “Begin Scan” button. I would like to change the structure of my code so that there is a new button that, when pressed, gathers the information from CVE Details but does not start the scan. Then, the “Begin Scan” button would only start the scan. Splitting the functionality would allow users to choose whether they perform the vulnerability scan with the information that the program had on file from the previous scan or pull the most recent information from the CVE Details website and then scan.

Updating the manner of retrieval of information from CVE Details

My program currently stores the links to the CVE Details page for the OS and each application in an array. This means that I must manually update those links when a new version of the OS or application comes out. An improvement to this system would be storing the links to the CVE details page prior to the version vulnerability page, which is a page that lists the previous versions of the OS or applications and the number of vulnerabilities present. If the program were to store the links to that page, the program could be made to check the HTML tags of that page for the newest version of the OS or application, which would be at the top of the page. Then, the program could navigate to that page and pull the vulnerability information from it, like it does currently. This would ensure that the scanner is always using up-to-date information and would prevent the need for manual changes to the links.

Having the program automatically run on computer startup

While not the most important improvement to make, it would be nice to have the program run automatically when the user starts their computer. This would help remind them to scan their system and to update their OS and applications.

These improvements would address the manual test cases that failed and result in a more efficient and effective program.

X. Defense Presentation Slides

Link to Slides:

https://docs.google.com/presentation/d/1SNbN43asdN72oBF_x2mgfVH_P0MEwoMNuCQ3WEXmYO4/edit?usp=sharing