Project Report

Trey Smith, Branden Mazour, Liam Hilton-Green, Jared Parker

Capstone Project [Spring 2021]

The University of West Florida

4/30/21

CIS4592 Capstone Project

Dr. Owsnicki-Klewe

Overview of Submission Structure

The structure overview can be found in the README.md file at our GitHub repository.

Table of Contents

A. Executive Summary

Privacy Hub will be a desktop application that helps the user keep track of what data about them is currently being gathered. Our main focus will be on tracking the usage of any cameras or microphones connected to the user's computer. We will let the user know when they are being monitored and what is monitoring them. This will allow the user to be more aware and in control of their privacy.

B. Requirements

B.1 Security Requirements

A few potential security concerns rest in the general system usage per the user. PrivacyHub itself has proven to be a secure application that limits itself to read-only permissions. However, a threat actor may still misuse Privacy Hub on a compromised system. One potential misuse case was privilege escalation, in the event that Privacy Hub were to need administrative privileges. This was mitigated by the lack of a need for administrative privileges.

Another misuse case involves a threat actor leveraging Privacy Hub to view the victim system's peripherals and processes in active reconnaissance. In this case, Privacy Hub would not be responsible for this intrusion, and it would be the failure or oversight of the user in regard to some other aspect of their system allowing a threat actor to break into their system in the first place. Thus, the mitigation requirement for this rests with the user, in which it is advised that the user follows best practices for personal computer security (e.g. ensure that antivirus is up to date, enable Windows Defender, do not turn off the firewall, et cetera).

Lastly, one final misuse case would be a threat actor noticing Privacy Hub on the victim system and takes action to disable it in their process of hiding presence, an important aspect of hacking that threat actors follow to avoid detection in a system. Again, this would not be the fault of Privacy Hub -- it would be a flaw from another aspect of the system environment. Still, it should be noted that if a user notices a process disappear abruptly, it could be a sign of a threat actor hiding presence.

In addition to the realization of misuse cases as potential security flaws, it is important to note the history of vulnerabilities in the tools and application programming interfaces (API) used for Privacy Hub, and ensure there are no present vulnerabilities with the current tools and APIs applied to the project. These include: OpenProcess (processthreadsapi.h), Handle - Windows Sysinternals, Processes.GetProcesses Method (System.Diagnostics), Process.NET, and Windows.Devices.Background Namespace - Windows UWP applications. Out of these, only OpenProcess, Windows Sysinternals, and Process.NET have a history of vulnerabilities. Among these APIs with vulnerabilities, Process.NET is the one that has the most concern.

Since OpenProcess and Windows Sysinternals are up to date, there will be no problems with these two APIs, especially since their heyday of vulnerabilities was in the first decade of the 21st century. Regarding Process.NET, the most extreme vulnerability to recur is remote code execution, notably in regard to its failure to check the source markup of a file or failure to validate input properly. This has occurred multiple times in previous versions, two examples in more recent times include CVE-2019-1113 and CVE-2018-8540. Thus, a critical security requirement for Privacy Hub would be to ensure that the software is compatible with the latest version of these APIs and that the development team remains vigilant of any revelations of vulnerabilities that can be exploited through Privacy Hub with Process.NET. Ultimately, like the other APIs, it is up to the user to ensure that their system is kept up to date to prevent these vulnerabilities from occurring.

After extensive research and analysis, the conclusion is that the security burden and concerns rest on the user. Privacy Hub leaves a small footprint on the system with a lack of required privileges, it serves to add a layer of security by requiring a threat actor work slightly harder to conceal presence, and is well-secured with its usage and requirement of the latest APIs and tools utilized in the software.

Summary of security requirements: the user is largely responsible for keeping their system up to date to prevent the misuse cases from occurring and all tools and APIs integrated in the project should be kept up to date with the latest versions in compliance with ISO 15408.

B.2 Final Requirements

Privacy Hub lets the user select the USB devices they want to monitor. It then lists the processes currently using those devices. The program can either group the listing by device or group it by process.

Whenever a new process starts using a selected device, a Windows toast notification notifies the user. The user can save certain processes as “trusted processes” so they are not notified of them (e.g. Discord).

The program automatically refreshes its list of processes every so often. The user can manually refresh the list if they want to.

B.3 Comparison with Initial Requirements

Privacy Hub has met our initial requirements. Once we learned how to deal with handles and processes in Windows, meeting project requirements became straightforward.

Our main goal was to detect and notify of processes using the user’s webcam and microphone. Our program now displays and notifies users of these events. We have even enabled Windows push notifications for when an untrusted process begins using a device.

While there are more features we could add to the application, we have covered the main functionality as well as polished the program, which is what we were aiming to do.

C. Timeline

Given fifteen weeks from our first meeting until our final submission, our team aimed to have consistent progress each week in order to avoid a crunch period as our deadline grew closer.

C.1 Final Timeline

We managed to keep true to our tasks and consistently create something new for our project every week. This led to very few moments of crunch time for our team and made the development process very enjoyable. Our greatest moments of crunch time came in week 6 when a decision needed to be made about the OS we were working in. If we weren't able to connect the process handles to the devices, we would have jumped from Windows to Linux. Luckily we were able to make the connection and could continue in the Windows environment.

Week	**Accomplishment/Deadlines**
1	Introduction, Group Formation
2	Project idea & discussion Project Plan outline
3	Researching aspects of Windows system
4	Researching aspects of Windows system
5	Device Discovery, Creation of device object
6	Connection between processes and devices successful
7	Development of basic GUI elements
8	Refactoring of project
9	Select devices to monitor
10	Proper display of all user-focused information in GUI
11	Manual refresh
12	Automatic refresh

13	Full functionality of standard features in GUI
14	Trusted processes and notification systems
15	Refactoring of GUI

C.2 Comparison with Initial Timeline

In our initial timeline, our team was largely casting out our best guesses for where we saw ourselves with the project by then. We had not done as much designing for our project ahead of time as we should have. This led to our goals getting accomplished by the end of the project but in a greatly different order and pace than anticipated. The biggest oversight was the immense amount of effort it would take to connect our processes and devices together.

Week	**Accomplishment/Deadlines**
1	
2	
3	Researching aspects of Windows system
4	Continue Windows research. Create building blocks/proof of concept programs.
5	Finalize decision to develop for Windows vs Linux.
6	Develop basic version of peripheral detection.
7	Develop GUI for showing what peripherals have been detected.
8	Have a working demo. Peripheral can be detected
9	Polish GUI, work on methods for detecting peripheral usage.
10	Finalize peripheral detection usage system.
11	Be able to detect which processes are accessing which peripherals
12	Polish and bug fixes
13	Have a fully functional and polished demo.

14	
15	

D. Project Results Compared with Expectations

At one point, we were expecting to potentially switch to another project. This expectation came up when we were still learning how to work with processes in C#. We were able to push through this challenge and stick with the original project, so in terms of those expectations, we exceeded them.

We were not expecting to have any problems among the team, and we did not end up having any. All of the team members played a key role in the project and regularly communicated with each other. So in terms of teamwork, we met our expectations.

We also met the expectations of our project deadlines. We never missed any documentation or presentations, and we never fell behind our initial project timeline.

E. Project Process Review

We managed to follow the Scrum methodology well. Every week, we met two times - once on Monday and once on Friday. During those meetings, we talked about our progress, shortcomings, and plans for the project. These meetings helped us stay focused on the project and overcome any problems we faced. They were integral to the success of our project.

To keep track of user stories and tasks, we used a Trello board. We used it to assign tasks to team members at the start of each sprint. This board helped us keep our workflow organized. It played a large role in our project's completion.

F. Work to be Done

There are some improvements we will make to Privacy Hub if we continue the project. Firstly, we will introduce a settings menu to let users control things within the program, such as notifications. These settings will make the program more user-friendly.

Secondly, we will decrease the time it takes for the program to run. Currently, the program hangs whenever hitting a button on the GUI. This happens because threading is not set up properly. We will set up threading so that background tasks (namely process loading) are done simultaneously with the GUI loading. These changes will make the program more responsive.

As of now, our system can only detect a user's USB devices. We plan to expand this into bluetooth devices and any other form a user would need so the user has total coverage for their devices.