**Almighty KeyLogger:**

**Architecture**

I implemented all the functionality with four files:

* **Functionality.dll** and **Runner.exe** contain the logic for the keylogger from the simple keylogging exercise.
* **MiniService.exe** contains the main functionality. It runs the keylogger on several desktops, stations, and sessions. It checks for the active session and runs the keylogger.
* **SugiotKeyLogger.exe** handles the installation/uninstallation of the service, starts the application, and is responsible for propagating data to the rest of the processes.
* (There is another file, **DropperTest.exe**, which I used to run some class code that was given.)

**List of security boundaries it successfully bypass**

My keylogger bypass all the boundaries.

The **MiniService** main thread always checks which session is logged in and its state (whether it is active), if there is a UI desktop, and if the keylogger is not currently running. It calls the function StartProcessInSession to run the keylogger on the current session. It uses WTSQueryUserToken, then DuplicateTokenEx, and finally CreateProcessAsUserW to create a process.

If the session is not logged in, it means that the session was created (I noticed that every time Windows switches a user, it creates a new session. WTSQueryUserToken will return an error). To handle this, I steal the Winlogon process token using the function OpenProcessToken (all the functionality is in the function TakeWinLogonToken). I noticed that when the session is in some state (0) and the console is connected, it means that the login screen UI is on.

Additionally, the **MiniService** control listens to session changes. It handles session console connect and session console disconnect events. If the console is disconnected, we kill the keylogger in that session. Otherwise, we create a new keylogger in the session.

I installed the keylogger with registry.