	Student
Name:	**Mikołaj**
Surname:	**Kabała**
ID:	**193380**
Project Name and number:	**Zapis do pliku wciśniętych klawiszy dla Windows (keylogger)**
Submission Date:	**20.12.2024**

1. Cel projektu

1.1 Główny cel projektu

Stworzenie keyloggera dla Windowsa 10/11, który będzie zapisywał Make Cody klawiatury do pliku tekstowego.

1.2 Narzędzia i sposób realizacji

Do realizacji projektu użyłem przykładowego kbfiltra udostępnionego przez Microsoft.

Cały projekt przeprowadziłem przy pomocy takich narzędzi jak:

- VirtualBox - tworzenie wirtualnej maszyny, na której został zastosowany sterownik
- WinDbg - do debugowania wirtualnej maszyny
- Visual Studio - z zintegrowanym WDK i SDK

1.3 Kbfiltr

Filtr oferowany przez Microsoft działa między sterownikiem KbdClass i i8042prt. Posiada on funkcję *ServiceCallBack()*, która jest odpowiedzialna za dostarczenie pakietu przycisków (które wywołały przerwanie) do systemu Windows.

W powyższej funkcji mamy bezpośredni dostęp do wciśniętych klawiszy i możemy przeprowadzić na nich różne operacje np. modyfikowanie, usunięcie czy jak w przypadku tego projektu zapisać je do pliku.

W projekcie skupiłem się na wgraniu sterownika dla klawiatury PS/2, dlatego keylogger nie przechwytuje przerwań np. z klawiatury podłączonej Bluetoothem.

1.4 Zapis make kodów do pliku

Do zapisu make kodów do pliku tekstowego użyłem funkcji z biblioteki <Wdm.h>, takich jak:

- ZwCreate
- ZwWrite

Jednakże należy pamiętać o Interrupt Request Level (IRQL). Jest to mechanizm do zarządzania priorytetami obsługi przerwań. Działający na konkretnym poziomie IRQL ma ograniczony zestaw funkcji, a odwołanie się z wyższego do niższego poziomu wywołuje fatal error.

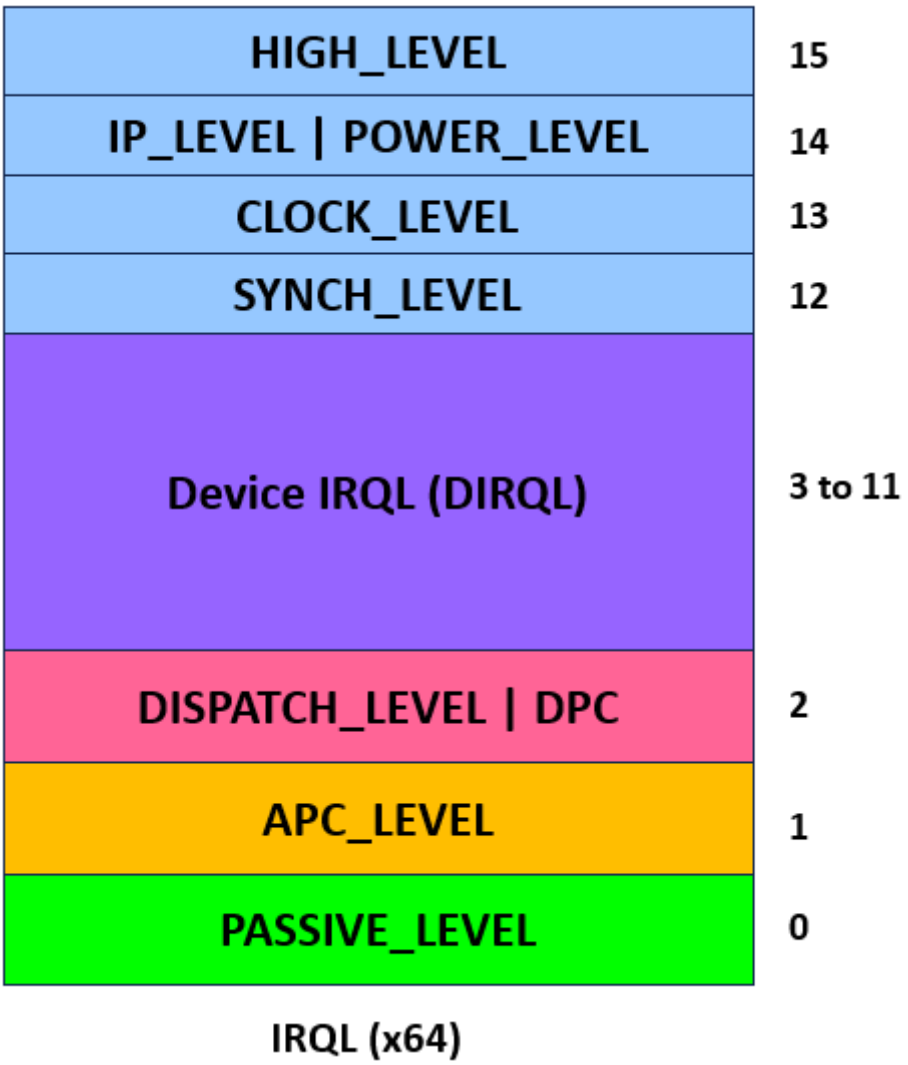


Tabela poziomów IRQL

Jako że przerwania z klawiatury występuą na poziomie 2, a powyższe funkcje z biblioteki <*Wdm.h*> są wykonywane na poziomie 0, to musiałem użyć wątków roboczych.

Przy każdym *ServiceCallBack()* jest zatem tworzony Work Item, któremu przekazuję wciśnięty makecode. Po jego stworzeniu dodajemy go do głównej kolejki, która wykona się na PASSIVE_LEVEL (IRQL 1).

Do funkcji Work Item należy przekazać make code'a wciśniętego przycisku. Funkcja ta jest odpowiedzialna za trzy rzeczy:

- otworzenie pliku
- zapis makecode’a
- zwolnienie Work Item z kolejki

2. Przebieg i realizacja projektu

2.1 Visual Studio, SDK, WDK

Pierwszym krokiem jest konfiguracja Visual Studio. Celem jest stworzenie środowiska, które umożliwi generowanie i testowanie nowych sterowników.

Poniżej krótki instruktarz:

1. Instalacja Visual Studio.

   W instalatorze w pojedynczych składnikach należy dodać poniższe komponenty:

   - MSVC v143 - VS 2022 C++ ARM64/ARM64EC Spectre-mitigated libs (Latest)
   - MSVC v143 - VS 2022 C++ x64/x86 Spectre-mitigated libs (Latest)
   - C++ ATL for latest v143 build tools with Spectre Mitigations (ARM64/ARM64EC)
   - C++ ATL for latest v143 build tools with Spectre Mitigations (x86 & x64)
   - C++ MFC for latest v143 build tools with Spectre Mitigations (ARM64/ARM64EC)
   - C++ MFC for latest v143 build tools with Spectre Mitigations (x86 & x64)
   - Windows Driver Kit

2. Instalacja SDK

   Windows SDK, czy Software Development Kit to zestaw narzędzi, bibliotek i dokumentacji, które umożliwiają tworzenie aplikacji dla systemu Windows.

   W celu pobrania SDK należy pobrać instalator ze strony Microsoft, a następnie go odpalić.

3. Instalacja WDK

   WDK, czyli Windows Driver Kit to zestaw narzędzi programistycznych, służący do tworzenia, testowania i debugowania sterowników dla systemu Windows.

   W celu pobranie WDK należy (podobnie jak powyżej z SDK) pobrać instalator i zainstalować go na swoim komputerze.

Po wykonaniu tych trzech kroków Visual Studio powinien dawać opcję tworzenia szablonów dla sterowników, w tym KMDF, którego użyjemy do stworzenia filtru sterownika dla klawiatury W celu sprawdzenia wersji albo poprawnej instalacji SDK należy wejść w **Ustawienia->Aplikacje->Zainstalowane aplikacje**, a następnie wpisać w wyszukiwarkę SDK. Następnie powinna nam się wyświetlić wersja pobranego SDK.

2.2 Maszyna wirtualna

Utworzone sterowniki powinniśmy testować na wirtualnych maszynach. Pisanie sterowników działających na poziomie jądra systemu wiąże się z dużym ryzykiem. W bardzo łatwy sposób (szczególnie przy zbyt małej wiedzy) możemy doprowadzić do poważnych awarii systemu. Wykorzystanie wirtualnej maszyny pozwala nam na bezpieczeństwo, szybkie przywracanie środowiska testowego, czy łatwe debugowanie.

Do projektu użyłem obrazu Windowsa 11, pobranego ze strony Microsoftu. Przy użyciu VirtualBoxa stworzyłem nową maszynę za pomocą pobranego pliku .iso. W ten sposób stworzyłem środowisko testowe dla moich sterowników.

2.3 WinDbg

WinDbg to narzędzie do debugowania dostarczane przez Microsoft. Jest ono kluczowe przy testowaniu sterowników na maszynach wirtualnych. Dzięki debugowaniu na poziomie jądra możemy analizować, co dokładnie powoduje błąd.

W tym celu należy zainstalować WinDbg na swoim urządzeniu.

2.4 Konfiguracja wirtualnej maszyny z WinDbg

Jeżeli posiadamy już wirtualną maszynę i WinDbg to musimy ustawić debugowanie w trybie jądra. W tym celu należy wykonać poniższe kroki:

1. W ustawieniach wirtualnej maszyny należy aktywować Serial Port. Nie jest istotne, który dokładnie (ważne, żeby zapamiętać, który się wybrało). Osobiście wybrałem COM1 i dalsze kroki opisują konfigurację przy użyciu tego właśnie portu.

2. Należy następnie odpalić wirtualną maszynę i w wierszu poleceń wpisać następujące komendy:

   bcdedit /debug on

   bcdedit /dbgsettings serial debugport:n baudrate:115200

   Powyższe komendy są odpowiedzialne za włączenie debugowania na poziomie systemu operacyjnego i wysyłaniu informacji debugowania przez port szeregowy, co umożliwi połączenie z WinDbg.

3. Następnie konfigurujemy WinDbg. Przechodzimy do **Pliki->Attach to kernel->COM**. W Baud Rate ustawiamy: 115200 (taką samą wartość jaką podaliśmy w komendzie w wirtualnej maszynie, krok 2), a w porcie wpisujemy nazwę wybranego portu szeregowego (dla COM1): \\.\pipe\com_1. Po tych ustawieniach zatwierdzamy przyciskiem OK.

Po wykonaniu powyższych komend połączenie między wirtualną maszyną powinno zostać utworzone. Możemy to zweryfikować, np uruchamiając ponownie wirtualną maszynę.

2.5 Pobranie przykładowego filtru od Microsoftu

Z githuba pobrałem przykładowy projekt, który zawiera przykład filtra do sterownika klawiatury. Sterownik ten działa pomiędzy KbdClass, ai8042prt. Przechodzą przez niego wszystkie pakiety z wciśniętymi klawiszami i są zwracane do KbdClass.

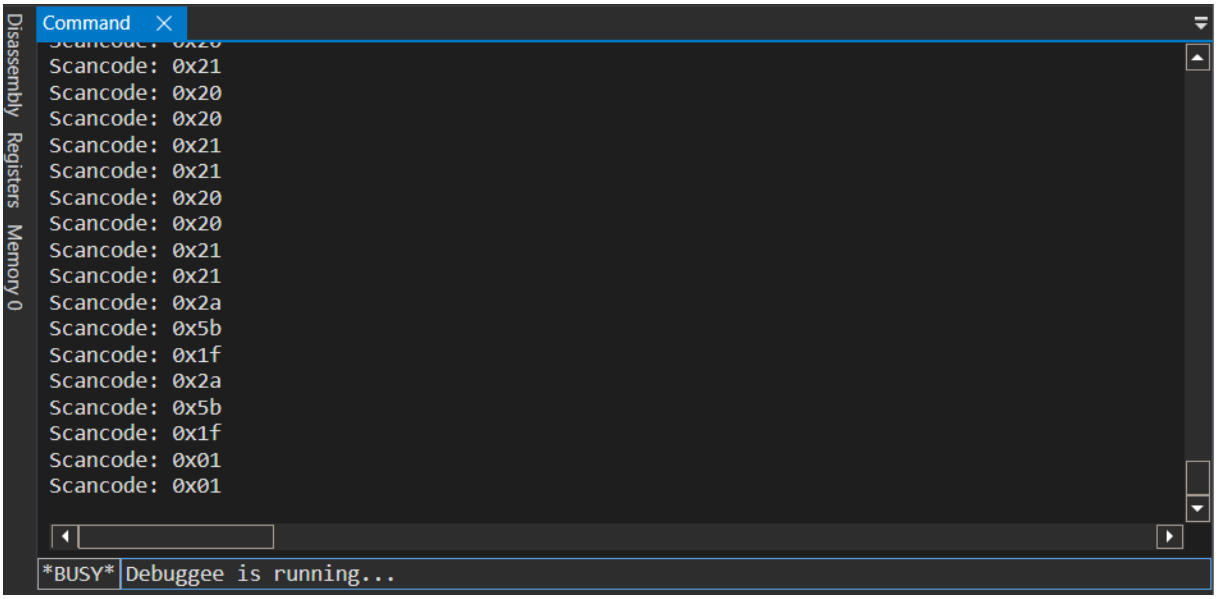
2.6 Modyfikacja filtra

To właśnie tutaj mamy bezpośredni dostęp do wykonywanych makecodów i możemy je zmodyfikować (usunąć, dodawać, czy modyfikować). Mamy taką możliwość bezpośrednio przed zwróceniem makecodów do KbdClass w funkcji ServiceCallBack.

Zacząłem od sprawdzenia działania filtra i wyświetlania makecodów w debugerze za pomocą DbgPrint. Wymagało to małych zmian w sterowniku (dopisanie pętli, która wyświetla każdy bajt z klawiszami w funkcji ServiceCallBack). Oprócz tego należy wpisać dwie komendy do WinDbg

- *ed nt!Kd_DEFAULT_MASK 0xFFFFFFFF* - przesyła wszystkie komunikaty wywołane za pomocą DbgPrint do debugera
- *!dbgprint* - wyświetla przesłane do debugera komunikaty

Debuger powinien wyświetlać wszystkie wciśnięte makecody.



Teraz pozostaje nam tylko zapis do pliku. Zrobiłem tego za pomocą metod zawartych w bibliotece <wdm.h>. Niestety poziomy przerwania klawiatury są na IRQL 2, a zapis do pliku na IRQL 0, także bezpośredni zapis makecodów do pliku w ServiceCallBack wywołuje fatal error odwoływania się do niższego poziomu IRQL.

W tym celu należy stworzyć Work Item, który po dodaniu do głównej kolejki wykona się na PASSIVE_LEVEL (ten sam poziom, co zapis do pliku). Do Work Itemu przypiszemy funkcję, która otwiera plik i zapisuje przekazane, jako argumenty funkcji, makecody.

Funkcja Work Itemu jest odpowiedzialna za dwie rzeczy:

- otworzenie pliku - przy pomocy ZwCreateFile używając flagi IF_OPEN, jeżeli plik istnieje to zostanie otwarty, a jeżeli nie to zostanie utworzony. Ścieżka do pliku jest na sztywno zdefiniowana w kodzie sterownika.
- zapis makecodów do pliku - przy pomocy ZwWriteFile zapisujemy przekazane parametry do funkcji.

Po zapisie do pliku należy pamiętać o zwolnieniu Work Itemu z głównej kolejki.

2.7 Wgranie filtra

Po budowie sterownika wgrałem na maszynę:

- folder kbfiltr wygenerowany w Visual Studio, który zawiera:
  - .sys - plik zawierający binarny kod sterownika
  - .inf - plik instalacyjny, który zawiera informacje o sterowniku
  - .cat - plik katalogu bezpieczeństwa
- certyfikat - plik do cyfrowego podpisu
- devcon - aplikacja do wgrywania sterowników oferowana w WDK przez Microsoft

Kroki do wgrania sterownika:

1. Wypełnienie certyfikatu
2. Zmiana zawartości .inx z PNPBAAD na identyfikator PS/2 (w moim przypadku PNP0303)
3. Otwarcie konsoli w miejscu, gdzie znajdują się zgrane pliki i wgranie sterownika za pomocą devcona:

   Devcon install kbfiltr.inf ACPI\VEN_PNP&DEV_0303

Po wykonaniu powyższych powinniśmy dostać komunikat o poprawnym wgraniu i jedyne co pozostało to zrestartowanie systemu i sprawdzenie działania sterownika.

3. Wnioski

Najważniejsze wnioski z projektu:

- Pisanie sterowników działających w jądrze systemu wymaga dużej wiedzy i uwagi.
- Przerwania mają własne poziomy uprzywilejowania, a zakłócenie zasad ich przetwarzania może się skończyć przerwaniem działania systemu (fatal error).
- Prowadzenie arkusza (“dziennika”) z projektu pomaga zebrać myśli, materiały i nie utknąć w miejscu (szczególnie przy projektach, które wymagają większego przygotowania teoretycznego). Przydał się nie tylko w czasie pisania projektu, ale także przy raporcie.
- Projekt pokazuje, jak istotne jest ograniczenie dostępu tylko do niezbędnych zasobów, aby zminimalizować zagrożenia dla codziennego użytkownika. Keylogger realizowany w ramach projektu jest ciężki do wykrycia.

4. Bibliografia

1. Konfiguracja Visual Studio, instalatory SDK i WDK
2. Windows SDK
3. ISO Windows 11
4. WinDbg
5. Konfiguracja wirtualnej maszyny z WinDbg
6. Github - Kbfiltr
7. Makecody
8. Arkusz prowadzonego projektu