# Keylogger z przesyłem do sieci poprzez http na linuxie

Michał Turski s193453

24 stycznia 2025

Sprawozdanie zawiera opis działania keyloggera oraz omówienie kluczowych funkcji, które były niezbędne do jego stworzenia. Ponadto, przedstawiono napotkane problemy oraz wykorzystane źródła, które miały wpływ na proces tworzenia oprogramowania.

## 1 Działanie programu

Oprogramowanie składa się z:

1. **Keyloggera**, który zapisuje wciśnięte klawisze do pliku tekstowego.
2. **Sendera**, który pobiera wartości z pliku tekstowego i wysyła je na serwer.
3. **Serwera**, który odbiera dane przesyłane przez Sendera i zapisuje je do logów.

Na początku działania programu generowany jest unikalny identyfikator (ID) w oddzielnym pliku tekstowym, który umożliwia identyfikację urządzenia, z którego pochodzą zarejestrowane naciśnięcia klawiszy. Jeśli ID zostanie usunięte przez system lub użytkownika, program wykrywa to i generuje nowy identyfikator.

### 1.1 Keylogger

W celu przechwycenia naciśnięć klawiszy wykorzystano przerwanie systemowe. Zastosowano funkcję:

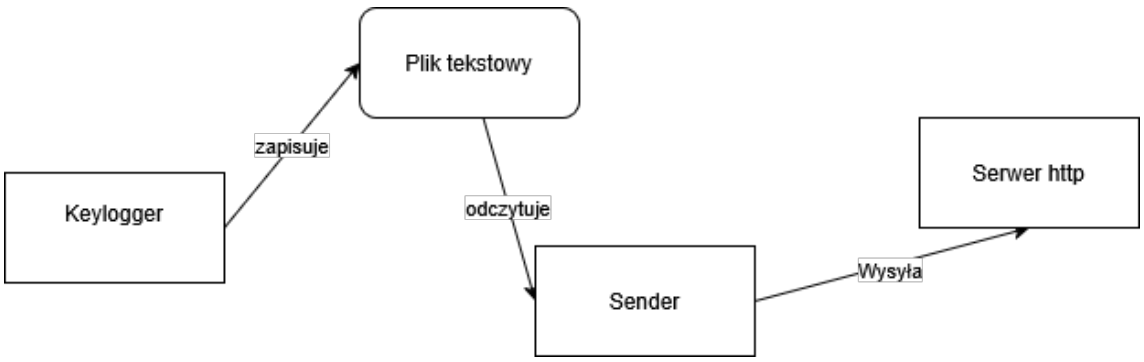
```
int request_irq(unsigned int irq, irq_handler_t handler,
            unsigned long flags, const char *name, void *dev);
```

dzięki której można uzyskać scancode z klawiatury. Dla klawiatury ustawiono wartości ‘IRQ=1‘ oraz ‘BUTTON_PORT=0x60‘.

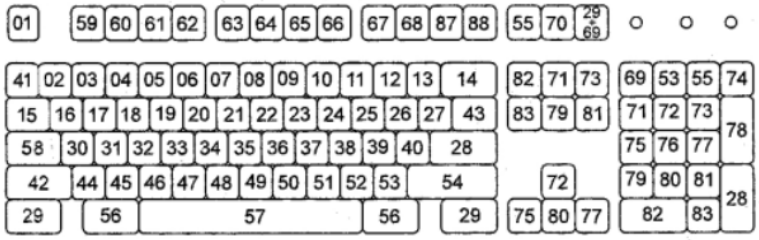
Dodatkowo konieczne było zarejestrowanie nowego urządzenia wejściowego za pomocą funkcji:

```
int input_register_device(struct input_dev *dev);
```

Po wywołaniu przerwania scancode jest przesyłany do pliku tekstowego, którego ścieżka znajduje się w stałej ‘LOG_FILE_PATH‘.




Rysunek 1: Uproszczony schemat działania oprogramowania




Rysunek 2: Scancody przycisków z klawiatury.

### 1.2 Sender

Sender cyklicznie wysyła pobrane scancody z pliku txt na server. W celu stworzenia cyklicznosci trzeba było zainicjalizować timer dzięki któremu nasz sterownik cyklicznie będzie wykonywał pewne instrukcje. Było to możliwe za pomoca:

```
void timer_setup(struct timer_list *timer, void (*callback)
(struct timer_list *), unsigned int flags);
int mod_timer(struct timer_list *timer, unsigned long expires);
```

W programie cyklicznie tworzymy nowy wątek za pomocą workqueue, który w sposób bezpieczny tworzy wątki. Następnie w utworzonym wątku odczytujemy scancody z pliku txt. Odczytywane jest pierwsze 512 bajtów pliku (wartość możliwa do zmiany, wyznaczona przez stała w programie) i czyszczona zawartość pliku. Kolejno w tym wątku tworzymy nowy wątek za pomoca w workqueue w którym wysyłamy pobrane scancody. Wysyłanie odbywa sie za pomoga socketów kernelowych.

```
int sock_create_kern(struct net *net, int family, int type,
int protocol, struct socket **res);
```

w których konieczne było zawarcie stringa który zawiera wszystkie konieczne nagłowki używane w komunikacji http.

Następnie socket moglismy połączyć sie z zadanym adresem ip za pomocą funkcji:

```
int kernel_connect(struct socket *sock, struct sockaddr *addr,
int addr_len, int flags);
```

### 1.3 Server

Po stronie serwera wykorzystano C# .NET REST API, które umożliwia łatwą i efektywną komunikację ze keyloggerem. API przyjmuje zapytanie typu POST, w którym dane są przesyłane w formacie JSON, zawierającym następujące informacje:

```
{
    ID: <ID urzadzenia z którego odebrano komunikta>
    Key: <ciąg scancodeow które zostały przycisniete w danej jednostce czasu>
}
```

A następnie zapisywane w odpowiednim pliku którego nazywa zawiera ID urządzenia.

## 2 Przydatne instrukcje w trakcie pisania sterownika

W trakcie pisania programu przydało mi sie pare komend dzięki którym mogłem efektywnie debugować kod

### 2.1 Dynamiczne dodanie programu 'w locie'

Dzięki poniższej instrukcji (w bashu) tworzymy strukture pliku

```
obj-m += hello_world.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Jeśli chcemy dodać do kernela nasz program:

```
sudo insmod <nazwa_drivera>.ko
```

Natomiast jeśli chcemy usunąc z kernela przyda sie instrukcja:

```
sudo rmmod <nazwa_drivera>.ko
```

Warto stworzyć skrypt w bashu z powyzszymi instrukcjami by usprawnić development.

### 2.2 Podgląd portów i logów

Przy przesyle danych warto sprawdzic czy dane sa poprawnie wysyłane do danego portu:

```
sudo tcpdump -i any port <number_portu> -X
```

Przy pisaniu programu możemy ustawić wysyłanie komunikatów do logów które możemy odczytać za pomoca:

```
sudo dmesg
```

Oto poprawiona wersja tekstu:

## 3 Problemy zaistniałe w trakcie pisania programu

W tej sekcji przedstawiono napotkane przeze mnie problemy podczas pisania kodu oraz proces ich rozwiązywania.

### 3.1 Zrównoleglenie operacji wysyłania do sieci i odczytywania wartości z klawiatury

W trakcie pisania oprogramowania rozpocząłem od stworzenia dwóch niezależnych modułów, które poprawnie wykonywały swoje zadania (wysyłanie wartości do serwera oraz odczyt z klawiatury). Jednak przy próbie wywołania funkcji obsługującej wysyłanie w callbacku przerwania występował deadlock.

Problem wynikał z konieczności stworzenia struktury, która umożliwiłaby zapisanie odczytanej wartości z klawiatury w sposób niezależny od przerwania. Rozwiązaniem okazało się zapisanie wciśniętego klawisza do pliku tekstowego, co umożliwiło programowi odpowiedzialnemu za wysyłanie wciśnięć odczytanie wartości bez ryzyka zakleszczenia.

### 3.2 Ignorowanie zapytania przez serwer

Podczas pisania sterownika odpowiedzialnego za wysyłanie zapytań napotkałem problem z ignorowaniem mojego zapytania przez serwer. Aby upewnić się, że problem leży po stronie serwera, sprawdziłem, czy zapytanie jest wysyłane w poprawny sposób, używając wspomnianej wcześniej komendy na porcie, na który wysyłałem zapytanie.

Po zauważeniu, że zapytanie jest poprawne i nie zawiera błędów, zmodyfikowałem serwer, aby akceptował wszystkie wartości za pomocą protokołu UDP. Okazało się, że po tej modyfikacji serwer poprawnie wyświetlał zapytanie, które było widoczne na porcie.

Problematyczna okazała się literówka w nagłówku, którą trudno było zauważyć na pierwszy rzut oka.

### 3.3 Nieodczytywanie danych po stronie nadawcy

Nadawca napotkał problem z pobieraniem danych z pliku tekstowego. Prawdopodobnie wynikało to z faktu, że w jednym wątku próbowałem jednocześnie odczytywać dane z pliku i wysyłać je na serwer. Po podzieleniu tych operacji na dwa oddzielne workqueue problem został rozwiązany.

# 4 Możliwe dalsze ulepszenia programu (nierozwiązane problemy)

## 4.1 Kontrola zapisu/odczytu pliku tekstowego

Konieczne byłoby wprowadzenie semafora do kontroli odczytu/zapisu, aby mieć pewność, że dane są poprawnie przekazywane do serwera, i żadna wartość nie zostanie pominięta, gdy równolegle programy chcą zapisać i odczytać dane.

## 4.2 Generowanie ID urządzenia po adresie MAC

Zamiast generować losowo ID dla urządzenia, można by je identyfikować po adresie MAC. Próbując pobrać za pomocą odpowiednich instrukcji adres MAC, otrzymywałem od funkcji odpowiedzialnej za to same zera. Może to wynikać z braku karty sieciowej w moim komputerze.

## 4.3 Ukrycie pliku zawierającego dane do przesyłu

Plik, w którym zapisywane są wciśnięcia, obecnie znajduje się w folderze /tmp. Folder ten jest dosyć łatwo dostępny, co sprawia, że plik tekstowy jest bardzo widoczny. Dobrym rozwiązaniem może być przechowywanie plików z danymi w mniej dostępnym miejscu, jak np. pamięć innego procesu lub biblioteki.

# 5 Wykorzystane źródła warte uwagi


1. https://www.kernel.org/doc/html/v5.4/networking/kapi.html
2. https://wiki.osdev.org/I/O_Ports
3. https://linux-kernel-labs.github.io/refs/heads/master/labs/networking.html
4. https://www.kernel.org/doc/html/v4.12/core-api/genericirq.html
5. https://docs.kernel.org/core-api/workqueue.html
6. https://litux.nl/mirror/kerneldevelopment/0672327201/ch10lev1sec7.html
7. http://www.haifux.org/lectures/217/netLec5.pdf - ciekawy wykład