Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Загружаемый модуль ядра для отслеживания USB-устройств, являющихся ключом для доступа к файлам.

Студент ИУ8-94 _____ А. А. Кириченков
(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Д. Е. Родионов
(Подпись, дата) (И.О.Фамилия)

Москва
2021г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире информация является таким же ценным, а в ряде случаев и более ценным, ресурсом, как всем привычные природные ресурсы в виде различного рода полезных ископаемых, территориальных, человеческих и пищевых ресурсов. По этой причине необходимо обеспечивать защиту данных.

Одним из способов защиты персональных данных на компьютере является предоставление доступа к ним по определенным факторам, например, наличию подключенного USB-устройства [1].

Linux – это операционная система с монолитным ядром. Для того, чтобы избежать перекомпиляции ядра при добавлении нового функционала, используются загружаемые модули ядра. Целью данной работы является реализация загружаемого модуля ядра для отслеживания USB устройств, являющихся ключом для доступа к определенным файлам, среди которых могут быть и исполняемые приложения.

Необходимая функциональность:

1. Список разрешенных устройств.
2. Список секретных файлов, приложений.
3. Предоставление или отказ в доступе, при наличии различных USB-устройств.

Для достижения поставленной цели необходимо решить следующие задачи.

1. Определение основных понятий.
2. Разработка алгоритмов.
3. Реализация загружаемого модуля.

1 АНАЛИТИЧЕСКИЙ РАЗДЕЛ

Целью данной работы является реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению. В данном разделе производится постановка задачи и рассмотрение основных понятий.

1.1 ПОСТАНОВКА ЗАДАЧИ

Требуется разработать программное обеспечение для отслеживани USB-устройств на ОС Linux, который обладает следующей функциональностью:

- список доверенных устройств;
- список путей к секретным файлам;
- отслеживание подключения USB-устройств;
    - при отключении доверенного устройства файлы зашифровываются;
    - при подключении устройства, если оно есть в списке доверенных, происходит расшифровка файла;
    - если устройство не известно, файл не расшифровывается.

На вход подается USB устройство с паролем. На выходе получаем зашифрованный или расшифрованный файл.

1.2 ЗАГРУЖАЕМЫЙ МОДУЛЬ ЯДРА

Ядро Linux динамически изменяемое – это позволяет без перекомпиляции всего ядра загружать дополнительную функциональность, выгружать функции из ядра и даже добавлять новые модули, использующие другие модули. Преимущество загружаемых модулей заключается в возможности сократить расход памяти для ядра, загружая только необходимые модули.

Загружаемый модуль представляет собой специальный объектный файл в формате ELF (Executable and Linkable Format). Обычно объектные файлы обрабатываются компоновщиком, который разрешает символы и формирует исполняемый файл. Однако в связи с тем, что загружаемый модуль не может разрешить символы до загрузки в ядро, он остается ELF объектом. Для работы с загружаемыми модулями можно использовать стандартные средства работы с объектными файлами (имеют суффикс .ko, от kernel object). [2]

В ОС Linux существуют специальные команды для работы с загружаемыми модулями ядра:

insmod – загружает модуль в ядро из конкретного файла, если модуль зависит от других модулей. Только суперпользователь может загрузить модуль в ядро.

lsmod – выводит список модулей, загруженных в ядро.

modinfo – извлекает информацию из модулей ядра (лицензия, автор, описание и т.д.).

rmmod – команда используется для выгрузки модуля из ядра, в качестве параметра передается имя файла модуля. Только суперпользо ватель может выгрузить модуль из ядра.

depmod – создает список зависимостей модулей, прочитывая каждый модуль в каталоге /lib/modules/ и определяя, какие символы они экспортируют, а какие символы им нужны.

modprobe - обавляет или удаляет модуль из ядра Linux

Загружаемые модули ядра должны содержать два макроса module_init и module_exit.

1.3 УВЕДОМЛЕНИЯ В ЯДРЕ LINUX

1.3.1 Уведомители

Ядро Linux содержит механизм, называемый «уведомителями» (notifiers) или «цепочками уведомлений» (notifiers chains), который позволяет различным подсистемам подписываться на асинхронные события от других подсистем. Цепочки уведомлений в настоящее время активно используется в ядре; существуют цепочки для событий hotplug памяти, изменения политики частоты процессора, события USB hotplug, загрузка и выгрузка модулей, перезагрузки системы, изменения сетевых устройств и т. д. [3]

Основной является структура notifier_block, листинг которой пред ставлен в 1.1.

Листинг 1.1 —Структура notifier_block

```
struct notifier_block {
  notifier_fn_t notifier_call ;
  struct notifier_block __rcu *next ;
  int priority ;
};
```

Структура определен в #include/linux/notifier.h. Эта структура содержит указатель на функцию обратного – notifier_call, ссылку на следующий notifier_block и приоритет функции, функции с более высоким приоритетом выполняются первыми.

1.3.2 Уведомитель изменений на USB портах

Существует уведомитель, позволяющий отслеживать изменения на usb портах. [4]

*void usb_register_notify(struct notifier_block *nb);*

*void usb_unregister_notify(struct notifier_block *nb);*

Существующие события:

- *USB_DEVICE_ADD* – добавление ново го устройства;
- *USB_DEVICE_REMOVE* – удаление устройства.

1.4 ХРАНЕНИЕ ИНФОРМАЦИИ О ДОСТУПНЫХ USB УСТРОЙСТВАХ

Для хранения устройств будем использовать двусвязный список ядра Linux, реализованный в библиотеке #include/linux/list.h. [5]

- *LIST_HEAD* – объявление и инициализация головы списка;
- *list_for_each_entry(temp, &connected_devices, list_node)* – проход по списку;
- *list_for_each_entry_safe(device, temp, &connected_devices, list_node)* – «защищенный» проход по всем элементам списка, используется для удаления записей списка;
- *list_add_tail(struct list_head * new, struct list_head * head)* – добавление нового элемента.

1.5 ВЫЗОВ ПРИЛОЖЕНИЙ ПОЛЬЗОВАТЕЛЬСКОГО ПРОСТРАНСТВА ИЗ ЯДРА

Usermode-helper API – это простой API с известным набором опций. Например, чтобы создать процесс из пользовательского пространства, обычно необходимо указать имя исполняемого файла, параметры исполняемого файла и набор переменных среды. [6]

- *int call_usermodehelper(const char *path, char **argv, char **envp, int wait)* – подготовить и запустить приложение пользовательского режима;
- *const char * path* – путь к исполняемому файлц пользовательского режима;

- *char ** argv* – параметры;
- *char ** envp* – переменные среды;
- *int wait* – дождитесь завершения работы приложения и возврата статуса.

1.6 ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ В ПРОСТРАНСТВЕ ЯДРА

Для реализации всего функционала, необходимо читать и записывать файловые данные в ядре Linux. [7]

Для этого используются следующие функции:

- *struct file* filp_open(const char* filename, int open_mode, int mode)* – открытие файла в ядре. filename – имя файла, который может быть создан или открыт, включает путь до файла; open_mode – режим открытия файла O_CREAT, O_RDWR, O_RDONLY, mode – используется при создании файла, установите разрешения на чтение и запись созданного файла, в противном случае он может быть установлен в 0;
- *int filp_close(struct file*filp, fl_owner_t id)* – закрытие файла;
- *ssize_t vfs_read(struct file* filp, char __user* buffer, size_t len, loff_t* pos), ssize_t vfs_write(struct file* filp, const char __user* buffer, size_t len, loff_t* pos)* – чтение и запись файлов в ядре.

Второй параметр этих двух функций имеет перед собой модификатор __user, который требует, чтобы оба указателя буфера указывали на память пространства пользователя. Чтобы эти две функции чтения и записи правильно работали с указателем буфера в пространстве ядра, нужно использовать функцию set_fs(). Ее задача состоит в том, чтобы изменить способ, которым ядро обрабатывает проверку адресов памяти. На самом деле параметр fs этой функции имеет только два зна чения: USER_DS и KERNEL_DS, которые представляют пространство пользователя и пространство ядра соответственно.

void set_fs(mm_segment_t fs)

mm_segment_t get_fs ()

1.7 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ

В данной работе происходит отслеживание изменений на usb портах, основными структурами являются usb_device и usb_device_id.

1.7.1 usb_device

Структура usb_device приведена в листинге А.1 – представление USB-устройста в ядре. Используемые поля:

- descriptor – дескриптор USB устройства.

Каждое продающееся устройство с USB требует сертификации на соответствие требованиям USB, для чего ему необходимо иметь ID поставщика (vendor ID) и ID изделия (product ID). Эти поля присутствуют в descriptor, используются для идентификации USB устройства.

1.7.2 usb_device_id

Структура usb_device_id приведена в листинге А.2 – идентификация USB устройств для отслеживания и подключения.

Используемые поля:

- idVendor – ID поставщика;
- idProduct – ID изделия.

2 КОНСТРУКТОРСКИЙ РАЗДЕЛ

Разрабатываемое программное обеспечения можно разделить на подзадачи:

- загружаемый модуль ядра;
- приложение для шифрования файлов;
- приложение для настройки модуля и взаимодействия с пользователем.

2.1 ПЕРЕХВАТ СООБЩЕНИЙ

Для перехвата сообщений добавление нового USB устройства и удаление USB устройства необходимо в загружаемом модуле ядра разместить уведомитель, принимающий в качества параметра функцию обратного вызова нашей обработки данного события.

Для этого была создана следующая структура представленная в листинге 2.1.

Листинг 2.1 —Структура usb_notify

```
static struct notifier_block usb_notify = { 2 .
notifier_call = notify ,
};
```

В этой структуре содержится указатель на прототип функции обработки:

*static int notify(struct notifier_block *self, unsigned long action, void *dev)*

Для создания уведомителя необходимо передать созданную структуру в функцию:

usb_register_notify(&usb_notify);

Для удаления уведомителя необходимо передать структуру в функцию:

usb_unregister_notify(&usb_notify);

2.2 ХРАНЕНИЕ ИНФОРМАЦИИ

Для хранения информации о подключенных USB устройствах создадим структуру, листинг 2.2.

Листинг 2.2 — Структура our_usb_device

```
typedef struct our_usb_device {
    struct usb_device_id dev_id ;
    struct list_head list_node ;
} our_usb_device_t;
```

Инициализируем список: *LIST_HEAD(connected_devices);*

Для добавления нового подключенного устройства используется функция А.3, для удаления – А.4.

2.3 АЛГОРИТМ РАБОТЫ ФУНКЦИИ-ОБРАБОТЧИКА

На рисунке 1 представлен алгоритм работы функции обратного вызова добавления или удаления USB устройства.

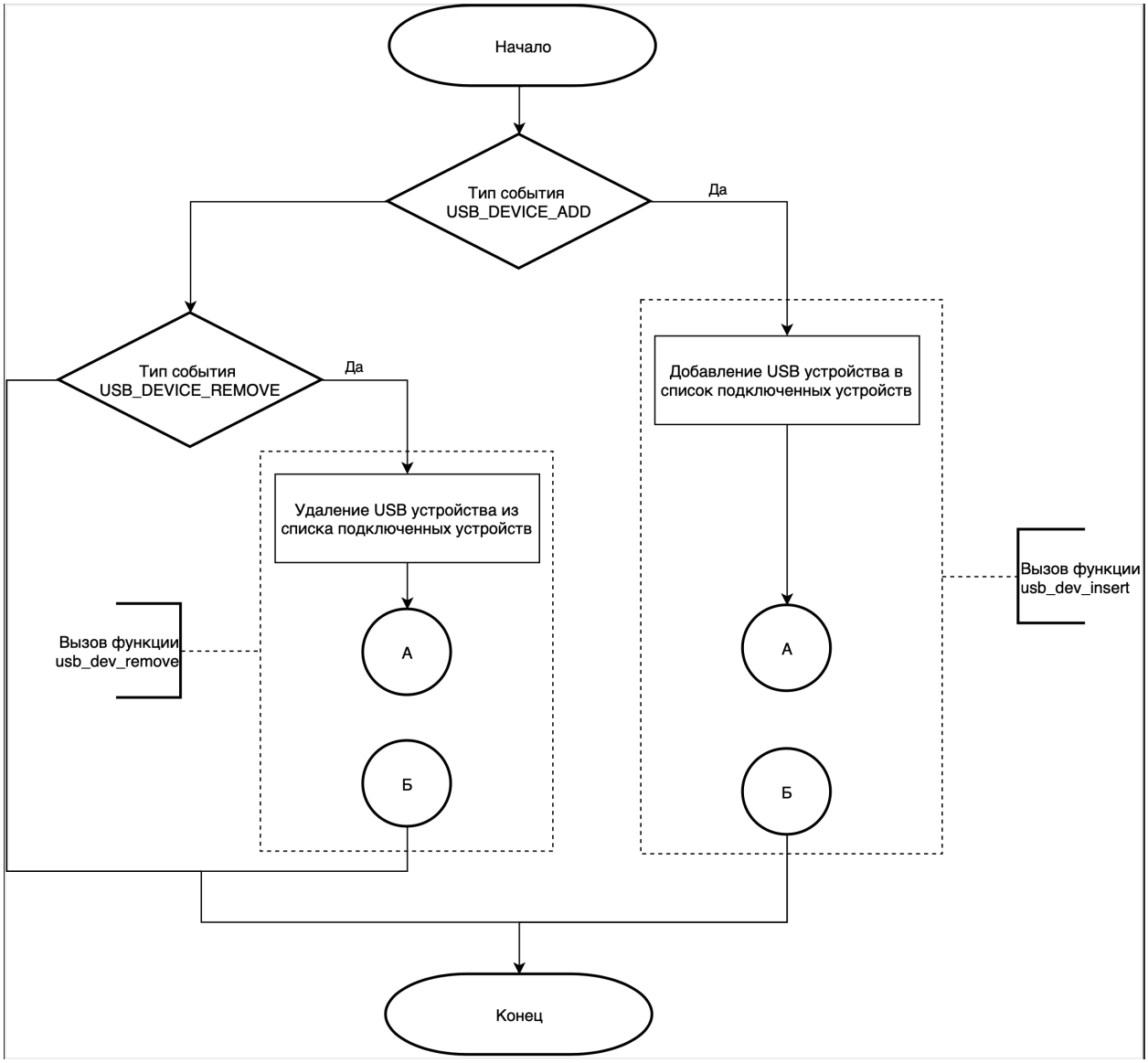

Рисунок 1 — Алгоритм работы функции-обработчика.

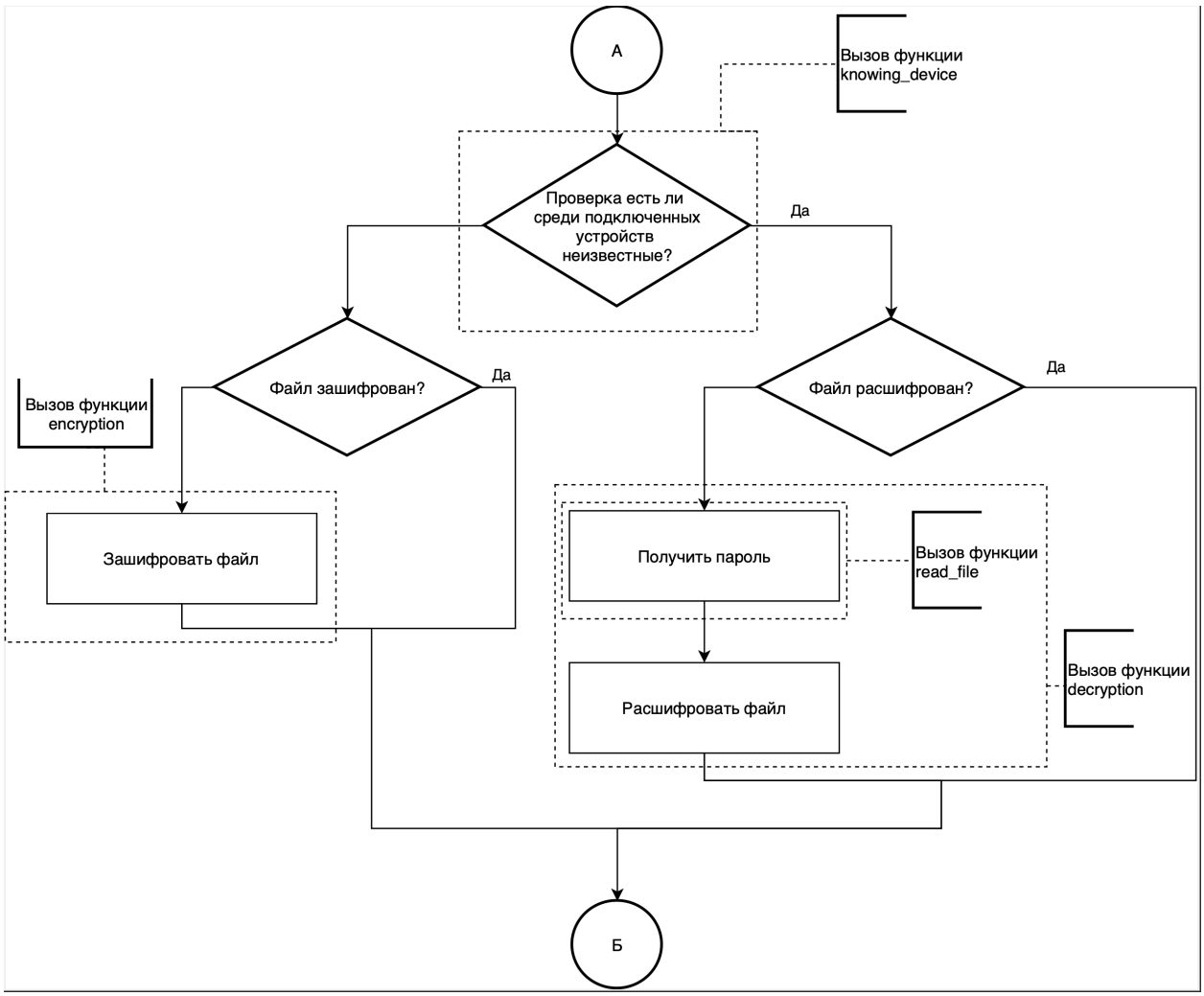

Рисунок 2 — Алгоритм работы функции-обработчика.

2.4 АЛГОРИТМ ШИФРОВАНИЯ ФАЙЛА

Для каждого файла из списка секретных.

1. Побайтовое считывание символов из файла.
2. Применение операции XOR для данных с паролем.
3. Побайтовая запись символов в файл.

3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

В соответствии с выбранной задачей – реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к файлам. Необходимо выбрать средства реализации, создать модули и интерфейс, описать ограничения и порядок работы программы.

3.1 ВЫБОР ТЕХНОЛОГИЙ

Для реализации модуля ядра и шифрования был выбран язык программирования C. Компилятор – gcc. Для облегчения сборки был написан Makefile, позволяющий запускать сборку одной командой, листинг А.5.

3.2 ХРАНЕНИЕ ДАННЫХ

Параметры USB устройств, идентификатор поставщика и изделия, а также список секретных файлов и приложений хранятся в конфигурационном файле. Шаблон конфигурационного файла USB устройств представлен в листинге А.6. Шаблон конфигурационный файл секретных файлов и приложений – листинг А.7.

Пароль для доступа к зашифрованным данным хранится по заданному пользователем пути в файле .key

3.3 ЗАГРУЖАЕМЫЙ МОДУЛЬ

Цель работы создание загружаемого модуля, реализация загрузки и удаления представлена в листинге А.8.

3.4 ФУНКЦИЯ-ОБРАБОТЧИК

В листинге А.9 представлена реализация функции обратного вы зова добавления или удаления USB устройства *static int notify(struct notifier_block *self, unsigned long action, void *dev).*

С последующим вызовом, в зависимости от события *static void usb_dev_remove(struct usb_device *dev), static void usb_dev_insert(struct usb_device *dev).*

3.5 ПРОВЕРКА ПРИНАДЛЕЖНОСТИ УСТРОЙСТВА ИЗВЕСТНЫМ

Чтобы узнать можно ли расшифровать файл, необходимо узнать принадлежит ли устройство списку разрешенных устройств. Каждое устройство имеет уникальную пару идентификатор поставщика и идентификатор изделия, по ней и будет происходить поиск. Также в известных устройствах хранится файл с паролем для расшифровки секретных данных.

Реализация данной проверки представлена в листинге А.10. Считывание пароля представлено в листинге А.11.

3.6 ПРОВЕРКА ПРИНАДЛЕЖНОСТИ УСТРОЙСТВА ИЗВЕСТНЫМ

После проверки принадлежности, при необходимости вызываются функции шифровки и расшифровки файлов, которые вызывают исполняемый файл пользовательского пространства. Реализация этих функций представлена в листинге А.12.

3.7 НАСТРОЙКА И КОНФИГУРАЦИЯ МОДУЛЯ

Для работы модуля необходимо указать id устройства, пути для шифрования. Для удобства получения этих данных и записи их в конфигурационный файл был реализован скрипт на языке программирования python. Скрипт необходимо запускать с правами суперпользователя. Исходный код представлен в листинге А.13.

3.7.1 Получение списка подключенных устройств

Для получения списка подключенных USB устройств, запускается подпроцесс lsusb. На рисунке 3 представлен пример работы загружаемого модуля.

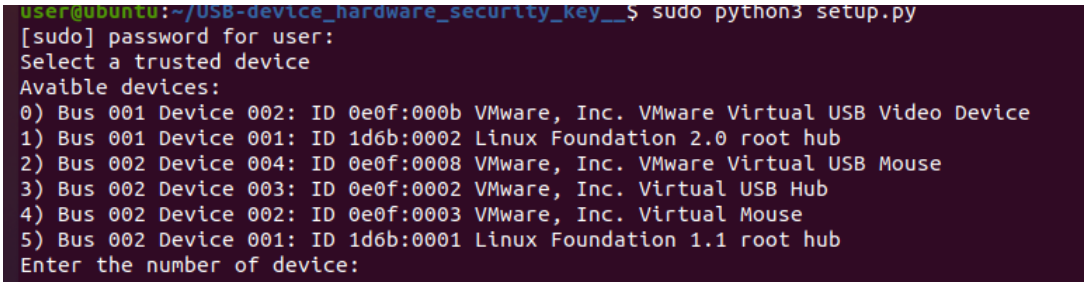

Рисунок 3 – Выбор доверенного устройства.

3.7.2 Получение списка файлов для шифрования

После того как пользователей выберет доверенное устройство, необходимо ввести файлы для шифрования. При этом проверяется существование введенных файлов.

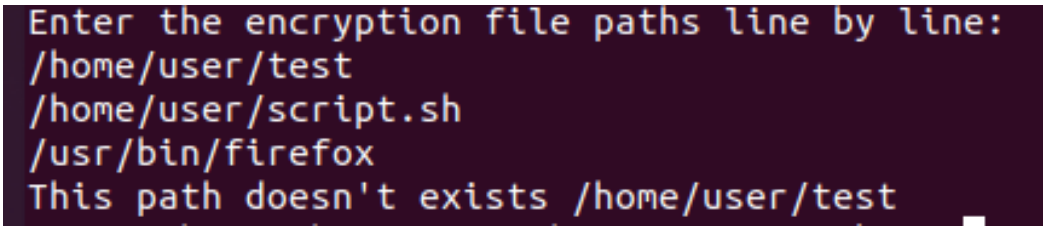

Рисунок 4 – Выбор файлов для шифрования.

3.7.3 Генерация и сохранение ключа шифрования

Пользователю требуется ввести путь для сохранения ключа шифрования. После этого ключ генерируется и сохраняется в файл .key.

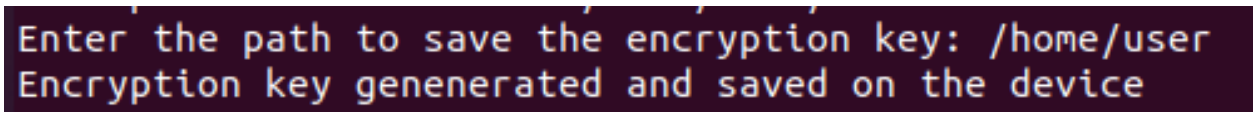

Рисунок 5 – генерация ключа шифрования.

3.8 КОМПИЛЯЦИЯ МОДУЛЕЙ ШИФРОВАНИЯ И ЯДРА

После того как все необходимые данные были введены, они сохраняются в конфигурационные файлы config.h и crypto_config.h.

Компиляция происходит с использованием утилиты make. Makefile представлен в листинге А.5.

Python скрипт вызывает подпроцесс make при помощи модуля subporocess. Если компиляция успешно завершается, то вызываются дальнейшие функции для загрузки модуля ядра, в противном случае вызывается исключение, и программа завершается.

Загрузка модуля ядра происходит с использованием утилиты modporbe. Для запуска модуля при перезагрузки системы, создается конфигурационный файл usb_key.conf в директории /etc/modules-load.d/.

После успешной загрузки модуля, python скрипт завершает работу, а все необходимые модули загружены.

3.9 ПРИМЕР РАБОТЫ

На рисунке 6 представлен пример работы загружаемого модуля.

```
user@ubuntu:~/USB-device_hardware_security_key__$ sudo dmesg -wH | grep "USB MODULE"
[sudo] password for user:
[   +0.000404] USB MODULE: Call_encrypt
[   +0.014281] USB MODULE: loaded.
[   +0.028632] USB MODULE: Delete device, we can't encrypt.
[Nov20 04:49] USB MODULE: unloaded.
[   +8.114592] USB MODULE: Call_encrypt
[   +0.003994] USB MODULE: loaded.
[   +0.015662] USB MODULE: New device, we can't encrypt.
[   +0.038613] USB MODULE: Delete device, we can't encrypt.
```

Рисунок 6 – Пример работы загружаемого модуля.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы выполнены следующие задачи.

- Определены основные понятия, такие как загружаемый модуль ядра, уведомления и уведомители. Рассмотрены структуры usb_device, usb_device_id.
- Разработаны алгоритмы работы функции-обработчика и шифрования файлов.
- Реализован загружаемого модуля.
- Реализован скрипт для конфигурирования модуля пользователем.

Достигнута цель проекта – реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению.

СПИСОК ЛИТЕРАТУРЫ

1. Утечки данных 2019: статистика. — Режим доступа: https://vc.ru/services/ 103616-utechki-dannyh-2019-statistika-tendencii-kiberbezopasnosti-i- (дата обращения: 20.09.2021).
2. Анатомия загружаемых модулей ядра Linux. — Режим доступа: https://www.ibm.com/developerworks/ru/library/l-lkm/ index.html" (дата обращения: 05.10.2021).
3. Notification Chains in Linux Kernel. — Режим доступа: https://0xax.gitbooks.io/linux-insides/content/Concepts/ linux-cpu-4.html (дата обращения: 10.10.2021).
4. include/linux/usb.h. — Режим доступа: https://elixir. bootlin.com/linux/latest/source/include/linux/usb.h#L2020 (дата обращения: 16.10.2021).
5. Doubly Linked Lists. — Режим доступа: https://www.kernel. org/doc/html/v4.14/core-api/kernel-api.html (дата обращения: 25.10.2021).
6. Invoking user-space applications from the kernel. — Режим доступа: https://developer.ibm.com/technologies/linux/articles/ l-user-space-apps/ (дата обращения: 10.11.2021).
7. Reading and writing of files in Linux kernel driver. —Режим доступа: https://www.programmersought.com/article/83015124510/ (дата обращения: 10.11.2021).

ПРИЛОЖЕНИЕ А

Листинг А.1 — Структура usb_device

```
struct usb_device {
    int     devnum;
    char        devpath[16];
    u32     route;
    enum usb_device_state   state;
    enum usb_device_speed   speed;
    unsigned int        rx_lanes;
    unsigned int        tx_lanes;

    struct usb_tt   *tt;
    int     ttport;

    unsigned int toggle[2];

    struct usb_device *parent;
    struct usb_bus *bus;
    struct usb_host_endpoint ep0;

    struct device dev;

    struct usb_device_descriptor descriptor;
    struct usb_host_bos *bos;
    struct usb_host_config *config;
```

```
struct usb_host_config *actconfig;
struct usb_host_endpoint *ep_in[16];
struct usb_host_endpoint *ep_out[16];

char **rawdescriptors;

unsigned short bus_mA;
u8 portnum;
u8 level;
u8 devaddr;

unsigned can_submit:1;
unsigned persist_enabled:1;
unsigned have_langid:1;
unsigned authorized:1;
unsigned authenticated:1;
unsigned wusb:1;
unsigned lpm_capable:1;
unsigned usb2_hw_lpm_capable:1;
unsigned usb2_hw_lpm_besl_capable:1;
unsigned usb2_hw_lpm_enabled:1;
unsigned usb2_hw_lpm_allowed:1;
unsigned usb3_lpm_u1_enabled:1;
unsigned usb3_lpm_u2_enabled:1;
int string_langid;
```

```
    /* static strings from the device */
    char *product;
    char *manufacturer;
    char *serial;

    struct list_head filelist;

    int maxchild;

    u32 quirks;
    atomic_t urbnum;

    unsigned long active_duration;

#ifdef CONFIG_PM
    unsigned long connect_time;

    unsigned do_remote_wakeup:1;
    unsigned reset_resume:1;
    unsigned port_is_suspended:1;
#endif
    struct wusb_dev *wusb_dev;
    int slot_id;
    enum usb_device_removable removable;
    struct usb2_lpm_parameters l1_params;
    struct usb3_lpm_parameters u1_params;
```

```
    struct usb3_lpm_parameters u2_params;
    unsigned lpm_disable_count;

    u16 hub_delay;
    unsigned use_generic_driver:1;
};
```

Листинг А.2 — Структура usb_device_id

```
struct usb_device_id {
    /* which fields to match against? */
    __u16       match_flags;

    /* Used for product specific matches; range is inclusive */
    __u16       idVendor;
    __u16       idProduct;
    __u16       bcdDevice_lo;
    __u16       bcdDevice_hi;

    /* Used for device class matches */
    __u8        bDeviceClass;
    __u8        bDeviceSubClass;
    __u8        bDeviceProtocol;

    /* Used for interface class matches */
    __u8        bInterfaceClass;
    __u8        bInterfaceSubClass;
    __u8        bInterfaceProtocol;
```

```
    /* Used for vendor-specific interface matches */

    __u8        bInterfaceNumber;


    /* not matched against */

    kernel_ulong_t  driver_info

        __attribute__((aligned(sizeof(kernel_ulong_t))));

};
```

Листинг А.3 — Добавление usb устройства

```
static void add_our_usb_device(struct usb_device *dev)

{

    our_usb_device_t* new_usb_device = (our_usb_device_t
*)kmalloc(sizeof(our_usb_device_t), GFP_KERNEL);

    struct usb_device_id new_id = { USB_DEVICE(dev-
>descriptor.idVendor, dev->descriptor.idProduct) };

    new_usb_device->dev_id = new_id;

    list_add_tail(&new_usb_device->list_node,
&connected_devices);

}
```

Листинг А.4 — Удаление usb устройства

```
{

    our_usb_device_t *device, *temp;

    list_for_each_entry_safe(device, temp, &connected_devices,
list_node)

    {

        if (device_match_device_id(dev, &device->dev_id))
```

```
        {
            list_del(&device->list_node);
            kfree(device);
        }
    }
}
```

Листинг А.5 — Makefile

```
ifneq ($(KERNELRELEASE),)
     obj-m := usb_key.o
else
     CURRENT = $(shell uname -r)
     KDIR = /lib/modules/$(CURRENT)/build
     PWD = $(shell pwd)

default:
     gcc -o crypto crypto.c
     cp crypto /usr/bin/
     $(MAKE) -C $(KDIR) M=$(PWD) modules
     mkdir -p /lib/modules/$(CURRENT)//usb_key
     cp usb_key.ko usb_key.mod.o Module.symvers
/lib/modules/$(CURRENT)//usb_key
     echo "usb_key" >> /etc/modules-load.d/usb_key.conf

clean:
     rm -rf .tmp_versions
```

```
	rm *.ko
	rm *.o
	rm *.mod.c
	rm *.symvers
	rm *.order
	rm crypto

endif
```

Листинг А.6 — Конфигурационный файл USB устройств

```
#define KEY_PATH "key_path_template"

bool state_encrypt = true;

struct known_usb_device {
    struct usb_device_id dev_id;
    char *name;
};

// List of all USB devices you know
static const struct known_usb_device known_devices[] = {
    { .dev_id = { USB_DEVICE("IdVendor_template",
"IdProduct_template") }, .name = "name_template" },
};
```

Листинг А.7 — Конфигурационный файл секретных файлов и приложений

```
#define KEY "key_template\n"
```

```
// Secret files will crypt or decrypt upon detecting change in
usb state.

static char *secret_apps[] = {

    "encryption_paths_tempalate",

     NULL,

};
```

Листинг А.8 — Загрузка и удаление модуля ядра

```
static int __init my_module_init(void)

{

    usb_register_notify(&usb_notify);

    call_encryption();

    printk(KERN_INFO "USB MODULE: loaded.\n");

    return 0;

}


static void __exit my_module_exit(void)

{

    usb_unregister_notify(&usb_notify);

    printk(KERN_INFO "USB MODULE: unloaded.\n");

}
```

Листинг А.9 — Функция-обработчик

```
// If usb device inserted.

static void usb_dev_insert(struct usb_device *dev)

{
```

```
    add_our_usb_device(dev);

    char *name = knowing_device();


    if (name)

    {

        if (state_encrypt)

            call_decryption(name);

        state_encrypt = false;

        printk(KERN_INFO "USB MODULE: New device we can
encrypt.\n");

    }

    else

   {

        if (!state_encrypt)

            call_encryption();

        state_encrypt = true;

        printk(KERN_INFO "USB MODULE: New device, we can't
encrypt.\n");

    }

}


// If usb device removed.

static void usb_dev_remove(struct usb_device *dev)

{

    delete_our_usb_device(dev);

    char *name = knowing_device();
```

```
    if (name)
    {
        if (state_encrypt)
            call_decryption(name);
        state_encrypt = false;
        printk(KERN_INFO "USB MODULE: Delete device, we can
encrypt.\n");
    }
    else
    {
        if (!state_encrypt)
            call_encryption();
        state_encrypt = true;
        printk(KERN_INFO "USB MODULE: Delete device, we can't
encrypt.\n");
    }
}


// New notify.
static int notify(struct notifier_block *self, unsigned long
action, void *dev)
{
    // Events, which our notifier react.
    switch (action)
    {
        case USB_DEVICE_ADD:
            usb_dev_insert(dev);
             break;
```

```
        case USB_DEVICE_REMOVE:
            usb_dev_remove(dev);
             break;
        default:
             break;
    }
    return 0;
}
```

Листинг А.10 — Функции для проверки разрешенных устройств

```
// Match device id with device id.
static bool device_id_match_device_id(struct usb_device_id
*new_dev_id, const struct usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != new_dev_id->idVendor)
        return false;
    if (dev_id->idProduct != new_dev_id->idProduct)
        return false;
    return true;
}

// Check our list of devices, if we know device.
static char *usb_device_id_is_known(struct usb_device_id *dev)
{
    unsigned long known_devices_len = sizeof(known_devices) /
sizeof(known_devices[0]);
```

```
    int i = 0;
    for (i = 0; i < known_devices_len; i++)
    {
        if (device_id_match_device_id(dev,
&known_devices[i].dev_id))
        {
            int size = sizeof(known_devices[i].name);
            char *name = (char *)kmalloc(size + 1, GFP_KERNEL);
            int j = 0;
            for (j = 0; j < size; j++)
                name[j] = known_devices[i].name[j];
            name[size + 1] = '\0';

            return name;
        }
    }
    return NULL;
}

static char *knowing_device(void)
{
    our_usb_device_t *temp;
    int count = 0;
    char *name;

    list_for_each_entry(temp, &connected_devices, list_node) {
        name = usb_device_id_is_known(&temp->dev_id);
```

```
        if (!name)
            return NULL;
        count++;
    }
    if (0 == count)
        return NULL;
    return name;
}
```

Листинг А.11 — Считывание ключа для шифрования

```
static char *read_file(char *filename)
{
    struct kstat *stat;
    struct file *fp;
    mm_segment_t fs;
    loff_t pos = 0;
    char *buf;
    int size;

    fp = filp_open(filename, O_RDWR, 0644);
    if (IS_ERR(fp))
    {
        return NULL;
    }

    fs = get_fs();
```

```
    set_fs(KERNEL_DS);


    stat = (struct kstat *)kmalloc(sizeof(struct kstat),
GFP_KERNEL);
    if (!stat)
    {
        return NULL;
    }


    vfs_stat(filename, stat);
    size = stat->size;


    buf = kmalloc(size, GFP_KERNEL);
    if (!buf)
    {
        kfree(stat);
        return NULL;
    }


    kernel_read(fp, buf, size, &pos);


    filp_close(fp, NULL);
    set_fs(fs);
    kfree(stat);
    buf[size]='\0';
    return buf;
}
```

Листинг А.12 — Функции вызывающие исполняемый файл пользовательского пространства

```
// Call decryption from user space

static int call_decryption(char *name_device)

{

    printk(KERN_INFO "USB MODULE: Call_decrypt\n");


    char path[80];

    strcat(path, KEY_PATH);

    char *data = read_file(path);


    char *argv[] = {

         "/usr/bin/crypto",

        data,

        NULL};


    static char *envp[] = {

        "HOME=/",

        "TERM=linux",

        "PATH=/sbin:/bin:/usr/sbin:/usr/bin",

        NULL};


    if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC)
< 0)

    {

        return -1;
```

```
    }


    return 0;
}


// Call encryption from user space
static int call_encryption(void)
{
    printk(KERN_INFO "USB MODULE: Call_encrypt\n");
    char *argv[] = {
        "/usr/bin/crypto",
        NULL};


    static char *envp[] = {
        "HOME=/",
        "TERM=linux",
        "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
        NULL};


    if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC)
< 0)
    {
        return -1;
    }


    return 0;
}
```

Листинг А.13 — Исходный код python модуля

```
import subprocess

import re

import os


'''

Проверка на запуск скрипта от рут-пользователя

'''



def is_root():

    return os.geteuid() == 0



'''

Вызывая процесс lsusb, получаем список доступных USB-устройств

'''



def get_avaible_device():

    lsusb_process = subprocess.Popen(

        ["lsusb"], shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)

    lsusb_process_stdout, lsusb_process_error = [

        x.decode('utf-8') for x in lsusb_process.communicate()]


    if lsusb_process.poll() != 0:
```

```
        raise Exception(f"Error: ${lsusb_process_error}")


    avaible_devices = lsusb_process_stdout.split('\n')[0:-1]
    return(avaible_devices)



'''
Меню для выбора пользователем доверенного устройства
'''



def select_device(avaible_devices):
    print("Select a trusted device\nAvaible devices:")
    for i, device in enumerate(avaible_devices):
        print(f"{i}) {device}")


    trusted_device_index = 0
    try:
        trusted_device_index = int(input("Enter the number of
device: "))
        if trusted_device_index > len(avaible_devices) - 1:
            raise ValueError
    except ValueError:
        print("Sorry, this index id unavaible")


    return(avaible_devices[trusted_device_index])
```

```
'''
Получаем название устройства и его уникальные индентификаторы
'''



def get_device_id(device):
    print(device)
    device_re = re.compile(

b"Bus\s+(?P<bus>\d+)\s+Device\s+(?P<device>\d+).+ID\s(?P<VendorI
d>\w+):(?P<ProductId>\w+)\s(?P<tag>.+)$", re.I)

    device_info = device_re.match(str.encode(device))
    if not device_info:
        raise Exception(f"Error Unexpected device: {device}")
    parsed_device_info = device_info.groupdict()


    device_id = int(device_info.groupdict()['VendorId'].decode(
        'utf-8'), 16),
int(parsed_device_info['ProductId'].decode('utf-8'), 16),
parsed_device_info['device'].decode('utf-8')
    return device_id



'''
Вызываем сборку проектов при помощи Makefile
'''
```

```
def make_build():

    make_process = subprocess.Popen(

        ["make"], shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)

    make_process_stdout, make_process_error = [

        x.decode('utf-8') for x in make_process.communicate()]



    if make_process.poll() != 0:

        raise Exception(f"Error make: {make_process_error}")

    return




'''

Удаляем временные файлы после сборки

'''




def make_clean():

    make_process = subprocess.Popen(

        ["make clean"], shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)

    make_process_stdout, make_process_error = [

        x.decode('utf-8') for x in make_process.communicate()]



    if make_process.poll() != 0:

        raise Exception(f"Error make clean:
{make_process_error}")
```

```
    return



'''
Выгружаем модуль ядра
'''



def unload_module():
    rm_module_process = subprocess.Popen(
        ["modprobe -r usb_key"], shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    rm_module_process_stdout, rm_module_process_stderr = [
        x.decode('utf-8') for x in
rm_module_process.communicate()]


    if rm_module_process.poll() != 0:
        raise Exception(f"modprobe -r usb_key:
{rm_module_process_stderr}")


    return



'''
Подгружаем модуль ядра, добавляя его зависимости в modules.dep
'''
```

```
def load_module():

    depmod_process = subprocess.Popen(

        ["depmod"], shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)

    depmod_process_stdout, depmod_process_stderr =
depmod_process.communicate()



    if depmod_process.poll() != 0:

        raise Exception(f"Error depmod:
{depmod_process_stderr}")



    modprobe_process = subprocess.Popen(

        ["modprobe usb_key"], shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    modprobe_process_stdout, modprobe_process_stderr =
modprobe_process.communicate()



    if modprobe_process.poll() != 0:

        raise Exception(f"Error modprobe usb_key:
{modprobe_process_stderr}")



    return




'''

Функиция генерации ключа для шифрования

'''
```

```
def generate_key():
    return os.urandom(24).hex()


'''
Сохранение ключа в указанную директорию
'''


def save_key(filepath, key):
    with open(filepath+"/.key", "w") as file:
        file.write(key)


'''
Меню для выбора пользователем пути для сохранения ключа
'''


def get_encryption_key_path():
    path = input("Enter the path to save the encryption key: ")

    while not os.path.exists(path):
        path = input(f"This path doesn't exists. Try again: ")
    return path
```

```
'''
Функция валидации на существование путей файлов для шифрования
'''


def validate_paths():
    paths = []
    while True:
        try:
            paths.append(input())
        except EOFError:
            break

    for path in paths:
        if not os.path.exists(path):
            print(f"This path doesn't exists {path}")
            paths.remove(path)
    return paths


'''
Меню для ввода пользователем защищаемых файлов
'''
```

```
def get_encryption_path():
    print("Enter the encryption file paths line by line:")
    paths = validate_paths()
    while len(paths) == 0:
        print(f"Zero valid paths have been entered. Try again")
        paths = validate_paths()
    return paths



'''
Сохраняем конфигурацию
'''



def save_config(device, encryption_paths, key_path, key):
    cur_config = ""
    updated_config = ""
    with open('config.template', 'r') as f:
        cur_config = f.read()


    IdVendor_template, IdProduct_template, name_template =
device
    cur_config = cur_config.replace('key_path_template',
key_path)
    cur_config = cur_config.replace(
        '"IdVendor_template"', str(IdVendor_template))
    cur_config = cur_config.replace(
```

```
        '"IdProduct_template"', str(IdProduct_template))
    updated_config = cur_config.replace('name_template',
name_template)

    with open('config.h', 'w') as f:
        f.write(updated_config)

    cur_config = ""
    updated_config = ""
    with open('crypto_config.template', 'r') as f:
        cur_config = f.read()

    cur_config = cur_config.replace('key_template', key)
    paths = [f'"{x}"' for x in encryption_paths]
    updated_config = cur_config.replace(
        '"encryption_paths_tempalate"', ",".join(paths))

    with open('crypto_config.h', 'w') as f:
        f.write(updated_config)


def main():
    if not is_root():
        print("You need root permissions to do this")
        return
    avaible_devices = get_avaible_device()
    device = get_device_id(select_device(avaible_devices))
```

```
    encryption_paths = get_encryption_path()
    key_path = get_encryption_key_path()
    key = generate_key()
    save_key(key_path, key)
    print("Encryption key genenerated and saved on the device")

    save_config(device, encryption_paths, key_path, key)

    unload_module()
    make_build()
    load_module()
    print("Kernel module successfully loaded")
    make_clean()


if __name__ == "__main__":
    main()
```