|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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)

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc88318957)

[1 АНАЛИТИЧЕСКИЙ РАЗДЕЛ 4](#_Toc88318958)

[1.1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc88318959)

[1.2 ЗАГРУЖАЕМЫЙ МОДУЛЬ ЯДРА 5](#_Toc88318960)

[1.3 УВЕДОМЛЕНИЯ В ЯДРЕ LINUX 6](#_Toc88318961)

[1.4 ХРАНЕНИЕ ИНФОРМАЦИИ О ДОСТУПНЫХ USB УСТРОЙСТВАХ 7](#_Toc88318962)

[1.5 ВЫЗОВ ПРИЛОЖЕНИЙ ПОЛЬЗОВАТЕЛЬСКОГО ПРОСТРАНСТВА ИЗ ЯДРА 7](#_Toc88318963)

[1.6 ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ В ПРОСТРАНСТВЕ ЯДРА 8](#_Toc88318964)

[1.7 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ 9](#_Toc88318965)

[2 КОНСТРУКТОРСКИЙ РАЗДЕЛ 10](#_Toc88318966)

[2.1 ПЕРЕХВАТ СООБЩЕНИЙ 10](#_Toc88318967)

[2.2 ХРАНЕНИЕ ИНФОРМАЦИИ 11](#_Toc88318968)

[2.3 АЛГОРИТМ РАБОТЫ ФУНКЦИИ-ОБРАБОТЧИКА 11](#_Toc88318969)

[2.4 АЛГОРИТМ ШИФРОВАНИЯ ФАЙЛА 13](#_Toc88318970)

[3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 14](#_Toc88318971)

[3.1 ВЫБОР ТЕХНОЛОГИЙ 14](#_Toc88318972)

[3.2 ХРАНЕНИЕ ДАННЫХ 14](#_Toc88318973)

[3.3 ЗАГРУЖАЕМЫЙ МОДУЛЬ 14](#_Toc88318974)

[3.4 ФУНКЦИЯ-ОБРАБОТЧИК 14](#_Toc88318975)

[3.5 ПРОВЕРКА ПРИНАДЛЕЖНОСТИ УСТРОЙСТВА ИЗВЕСТНЫМ 15](#_Toc88318976)

[3.6 ПРОВЕРКА ПРИНАДЛЕЖНОСТИ УСТРОЙСТВА ИЗВЕСТНЫМ 15](#_Toc88318977)

[3.7 НАСТРОЙКА И КОНФИГУРАЦИЯ МОДУЛЯ 15](#_Toc88318978)

[3.8 КОМПИЛЯЦИЯ МОДУЛЕЙ ШИФРОВАНИЯ И ЯДРА 16](#_Toc88318979)

[3.9 ПРИМЕР РАБОТЫ 17](#_Toc88318980)

[ЗАКЛЮЧЕНИЕ 18](#_Toc88318981)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc88318982)

[ПРИЛОЖЕНИЕ А 20](#_Toc88318983)

# 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.

## 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.

### ПОСТАНОВКА ЗАДАЧИ

Требуется разработать программное обеспечение для отслежива­ни USB-устройств на OC 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.

### 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 OC 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.

### УВЕДОМЛЕНИЯ В ЯДРЕ LINUX

#### 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.

#### 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.

### 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.

### 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.

### 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:

* *struсt 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 ()*

### ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ

В данной работе происходит отслеживание изменений на usb пор­тах, основными структурами являются usb\_device и usb\_device\_id.

#### 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.

#### 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.

## КОНСТРУКТОРСКИЙ РАЗДЕЛ

Разрабатываемое программное обеспечения можно разделить на подзадачи:

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

### 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);*

### 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.

### АЛГОРИТМ РАБОТЫ ФУНКЦИИ-ОБРАБОТЧИКА

На рисунке 1 представлен алгоритм работы функции обратного

вызова добавления или удаления USB устройства.

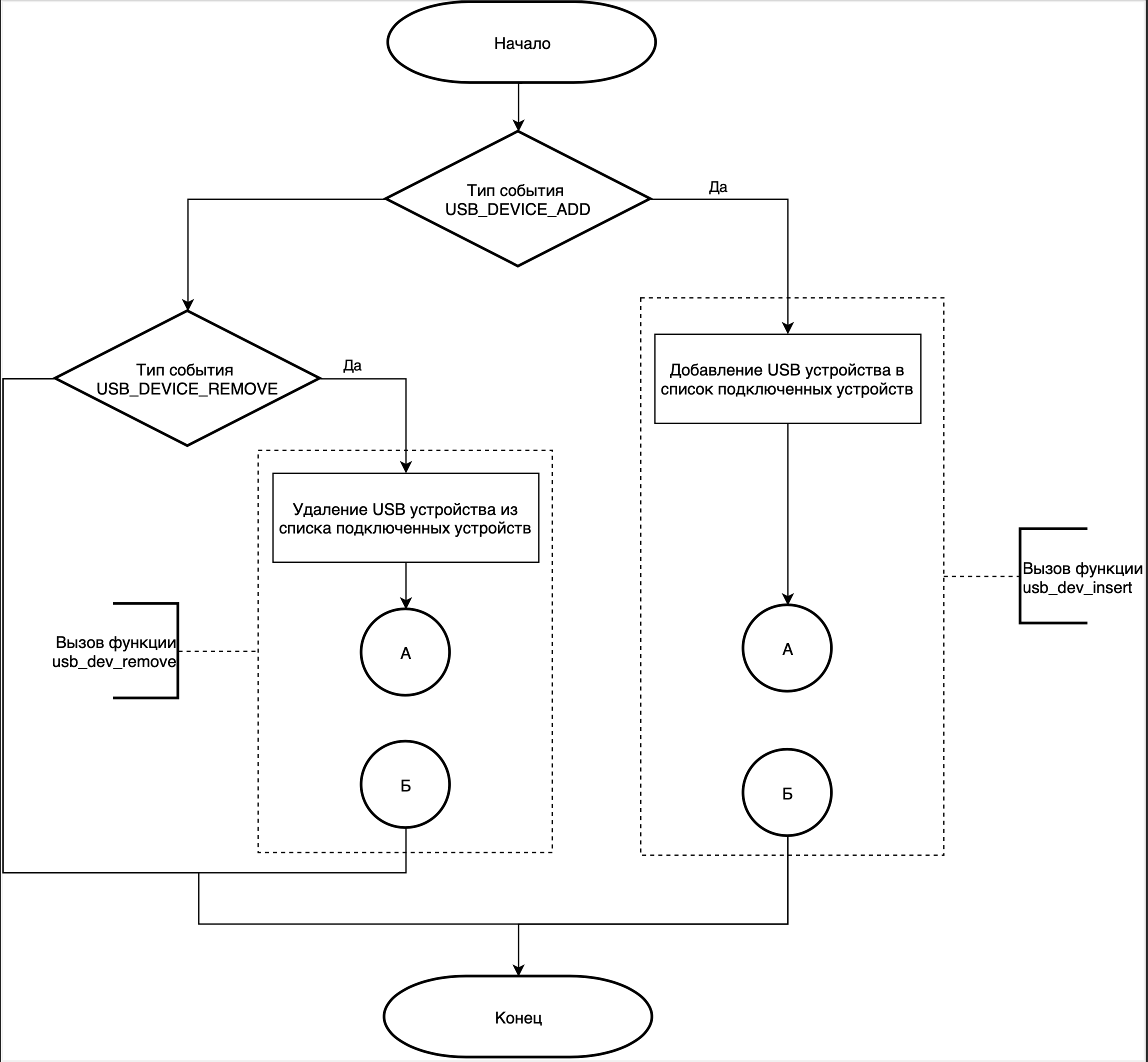


Рисунок 1 — Алгоритм работы функции-обработчика.

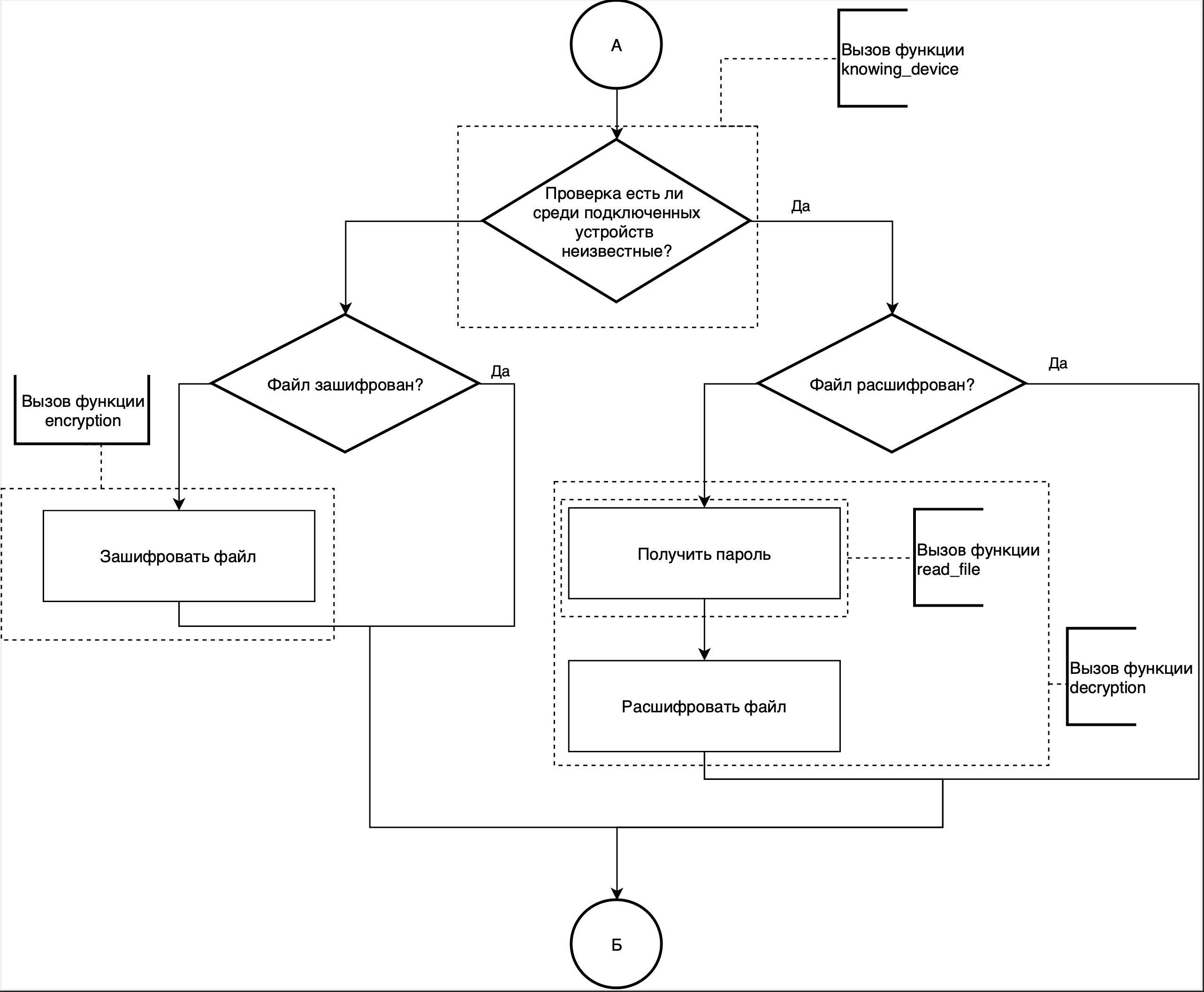


Рисунок 2 — Алгоритм работы функции-обработчика.

### 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.

## 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.

### 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.

### 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

### ЗАГРУЖАЕМЫЙ МОДУЛЬ

Цель работы создание загружаемого модуля, реализация загрузки

и удаления представлена в листинге А.8.

### 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).*

### 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.

### 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.

### 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.

#### Получение списка подключенных устройств

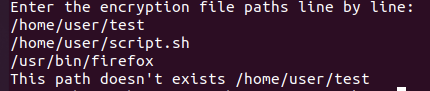
Для получения списка подключенных USB устройств, запускается подпроцесс lsusb. На рисунке 3 представлен пример работы загружаемого модуля.

#### 

#### Рисунок 3 – Выбор доверенного устройства.

#### 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.

#### 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.

### 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 A.5.

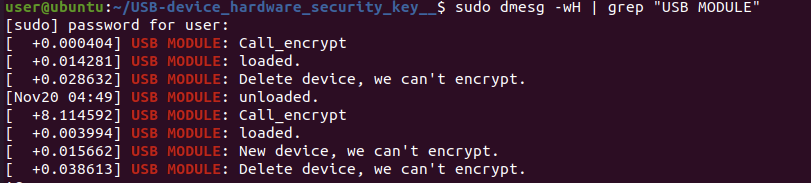
Python скрипт вызывает подпроцесс make при помощи модуля subporocess. Если компиляция успешно завершается, то вызываются дальнейшие функции для загрузки модуля ядра, в противном случае вызывается исключение, и программа завершается.

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.

### ПРИМЕР РАБОТЫ

На рисунке 6 представлен пример работы загружаемого модуля.



#### 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<VendorId>\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()