



Министерство образования и науки 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»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Реализация загружаемого модуля ядра для отслеживания
USB-устройств, их идентификации, предоставления или
отказа в доступе»

Студент ИУ7-76Б
(Группа)

(Подпись, дата) Ж.Р.Турсунов
(И.О.Фамилия)

Руководитель курсового проекта

(Подпись, дата) Н.Ю.Рязанова
(И.О.Фамилия)

Консультант

(Подпись, дата) _____
(И.О.Фамилия)

Москва, 2021 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7
(Индекс)

И.В.Рудаков
(И.О.Фамилия)

« ____ » _____ 2021 г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине _____ Операционные системы

Студент группы _____ ИУ7-76Б

Турсунов Жасурбек Рустамович
(Фамилия, имя, отчество)

Тема курсовой работы _____ Реализация загружаемого модуля ядра для отслеживания USB-устройств,
их идентификации, предоставления или отказа в доступе.

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) _____ кафедра

График выполнения проекта: 25% к ____ нед., 50% к ____ нед., 75% к ____ нед., 100% к ____ нед.

Задание

Необходимо реализовать загружаемый модуль ядра для отслеживания изменений в USB-
портах и проверки на наличие доступа к секретным файлам. В случае отсутствия допуска - зашиф-
ровать все запрещенные для копирования файлы.

Оформление курсовой работы:

Расчетно-пояснительная записка на 20-30 листах формата А4.

Расчетно-пояснительная записка должна содержать введение, аналитическую часть, конст-
рукторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение,
список литературы, приложения.

Дата выдачи задания « ____ » _____ 2021 г.

Руководитель курсового проекта

(Подпись, дата) Н.Ю.Рязанова
(И.О.Фамилия)

Студент

(Подпись, дата) Ж.Р.Турсунов
(И.О.Фамилия)

Содержание

Введение	4
1 Аналитическая часть	5
1.1 Постановка задачи	5
1.2 Загружаемый модуль ядра	5
1.3 Уведомления в ядре Linux	5
1.3.1 Уведомители	5
1.3.2 Уведомитель изменений на USB портах	6
1.4 Хранение информации о доступных USB-устройствах	6
1.5 Вызов приложения пользовательского пространства из ядра	6
1.6 Чтение и запись файлов в пространстве ядра	6
1.7 Основные используемые структуры	7
1.7.1 usb_device	7
1.7.2 usb_device_id	7
1.8 Вывод	7
2 Конструкторская часть	8
2.1 Перехват сообщений	8
2.2 Хранение информации	8
2.3 Алгоритм работы функции-обработчика	8
2.4 Алгоритм шифрования файла	10
2.5 Вывод	10
3 Технологическая часть	11
3.1 Стек технологий	11
3.2 Хранение данных	11
3.3 Загружаемый модуль	11
3.4 Функция-обработчик	11
3.5 Идентификация устройства	11
3.6 Вывод	12
4 Исследовательская часть	13
4.1 Системные характеристики	13
4.2 Постановка эксперимента	13
4.3 Пример работы	13
4.4 Вывод	13
Заключение	14
Список литературы	15
Листинги	16

Введение

Персональные компьютеры, системы управления и сети на их основе быстро входят во все области человеческой деятельности. Среди них можно выделить такие сферы применения, как военная, коммерческая, банковская, научная. Очевидно, широко используя компьютеры для обработки и передачи информации, эти отрасли должны быть надежно защищены от возможности доступа к ней посторонних лиц. Ее утраты или искажения. Согласно данным, более 80% компаний несут финансовые убытки из-за нарушения целостности и конфиденциальности используемых данных.

В настоящее время одним из способов защиты персональных данных на компьютере, это доступ к ним по USB-устройству. А именно если вы не хотите, чтобы кто-то извлекал документы, устанавливал вредоносное ПО с вашего компьютера через внешние носители, необходимо отслеживать USB-устройства, их идентифицировать и предоставлять или отказывать в доступе [1].

Linux - это операционная система с монолитным ядром. Для того, чтобы избежать перекомпиляции ядра при добавлении нового функционала, используются загружаемые модули ядра.

Цель данной работы - реализация загружаемого модуля ядра , для отслеживания USB-устройств, являющихся ключом для секретных файлов. Для достижения поставленной цели необходимо решить следующие задачи:

1. определение основных понятий;
2. разработка алгоритмов;
3. реализация загружаемого модуля.

1 Аналитическая часть

1.1 Постановка задачи

Требуется разработать программное обеспечение для отслеживания USB-устройств, который обладает следующей функциональностью:

1. список разрешенных устройств;
2. список путей к секретным файлам;
3. отслеживание появления новых USB-устройств;
 - (a) если устройство опознано, секретные файлы расшифрованы и доступны;
 - (b) если устройство не опознано, происходит зашифровка секретных файлов;

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

1.2 Загружаемый модуль ядра

Ядро Linux динамически изменяемое – это означает, что вы можете загружать в ядро дополнительную функциональность, выгружать функции из ядра и даже добавлять новые модули, использующие другие модули ядра. Преимущество загружаемых модулей заключается в возможности сократить расход памяти для ядра, загружая только необходимые модули (это может оказаться важным для встроенных систем).

Загружаемый модуль представляет собой специальный объектный файл в формате ELF (Executable and Linkable Format). 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:

1. `insmod` – Загружает модуль в ядро из конкретного файла, если модуль зависит от других модулей. Только суперпользователь может загрузить модуль в ядро;
2. `lsmod` – Выводит список модулей, загруженных в ядро;
3. `modinfo` – Извлекает информацию из модулей ядра (лицензия, автор, описание);
4. `rmmod` – Команда используется для выгрузки модуля из ядра, в качестве параметра передается имя файла модуля. Только суперпользователь может выгрузить модуль из ядра.

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 [3].

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 нового устройства, *USB_DEVICE_REMOVE* – удаление устройства.

1.4 Хранение информации о доступных USB-устройствах

Для хранения устройств будет использовать двусвязный список ядра Linux, реализованный в файле `#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 приложения 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.

Параметры:

1. *const char * path* – путь к исполняемому файлу пользовательского режима;
2. *char ** argv* – параметры;
3. *char ** envp* – переменные среды;
4. *int wait* – дождитесь завершения работы приложения и возврата статуса.

1.6 Чтение и запись файлов в пространстве ядра

Иногда необходимо читать и записывать файловые данные в ядре Linux [7].

В основном это 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, вам нужно использовать функцию `set_fs()`. Ее функция состоит в том, чтобы изменить способ, которым ядро обрабатывает проверку адресов памяти. На самом деле параметр `fs` этой функции имеет только два значения: `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 Основные 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 3 – представление USB-устройства в ядре. 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 4 – идентификация USB устройств для отслеживания и подключения.

Используемые поля:

1. `idVendor` – ID поставщика;
2. `idProduct` – ID изделия.

1.8 Вывод

В данном разделе была поставлена задача и рассмотрены основные задачи. Также были рассмотрены базовые принципы работы загружаемых модулей ядра, необходимый функционал для работы с USB-устройствами, а также их отслеживание на портах.

2 Конструкторская часть

2.1 Перехват сообщений

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

Для этого была создана следующая структура представленная в листинге 1.

```
1  static struct notifier_block usb_notify = {  
2      .notifier_call = notify,  
3  };  
4
```

Листинг 1: Структура usb_notify

В этой структуре содержится указатель на прототип нашей функции обработки:

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

Для создания уведомителя передаем созданную структуру в функцию:

```
usb_register_notify(&usb_notify);
```

Для удаления уведомителя передаем структуру в функцию:

```
usb_unregister_notify(&usb_notify);
```

2.2 Хранение информации

Для хранения информации о подключенных USB устройствах создадим структуру, листинг 2.

```
1  typedef struct our_usb_device {  
2      struct usb_device_id dev_id;  
3      struct list_head list_node;  
4  } our_usb_device_t;  
5
```

Листинг 2: Структура our_usb_device

Инициализируем список: *LIST_HEAD(connected_devices);*

Для добавления нового подключенного устройства используется функция 5, для удаления – 6.

2.3 Алгоритм работы функции-обработчика

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

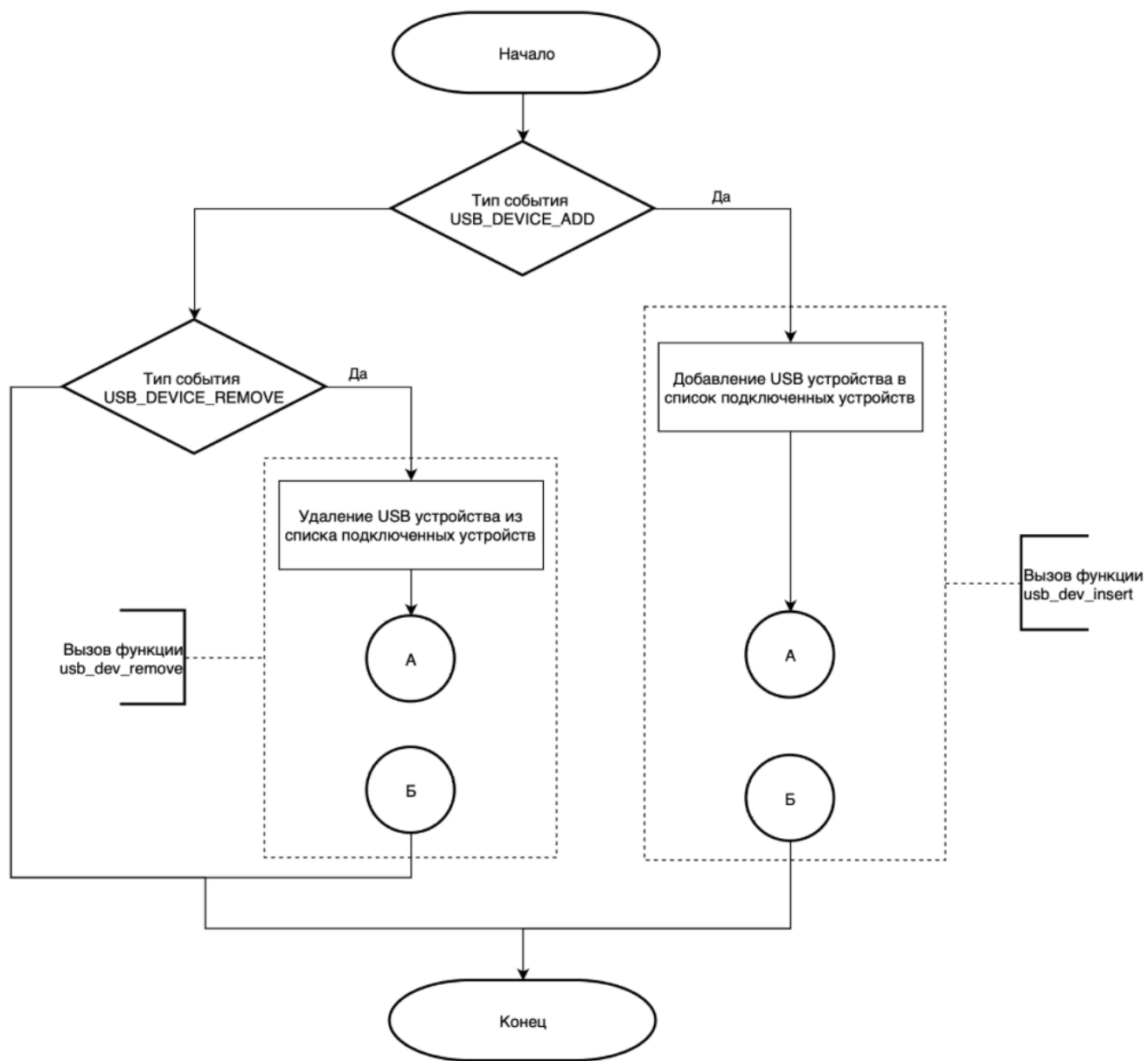


Рис. 1: Алгоритм работы функции-обработчика.

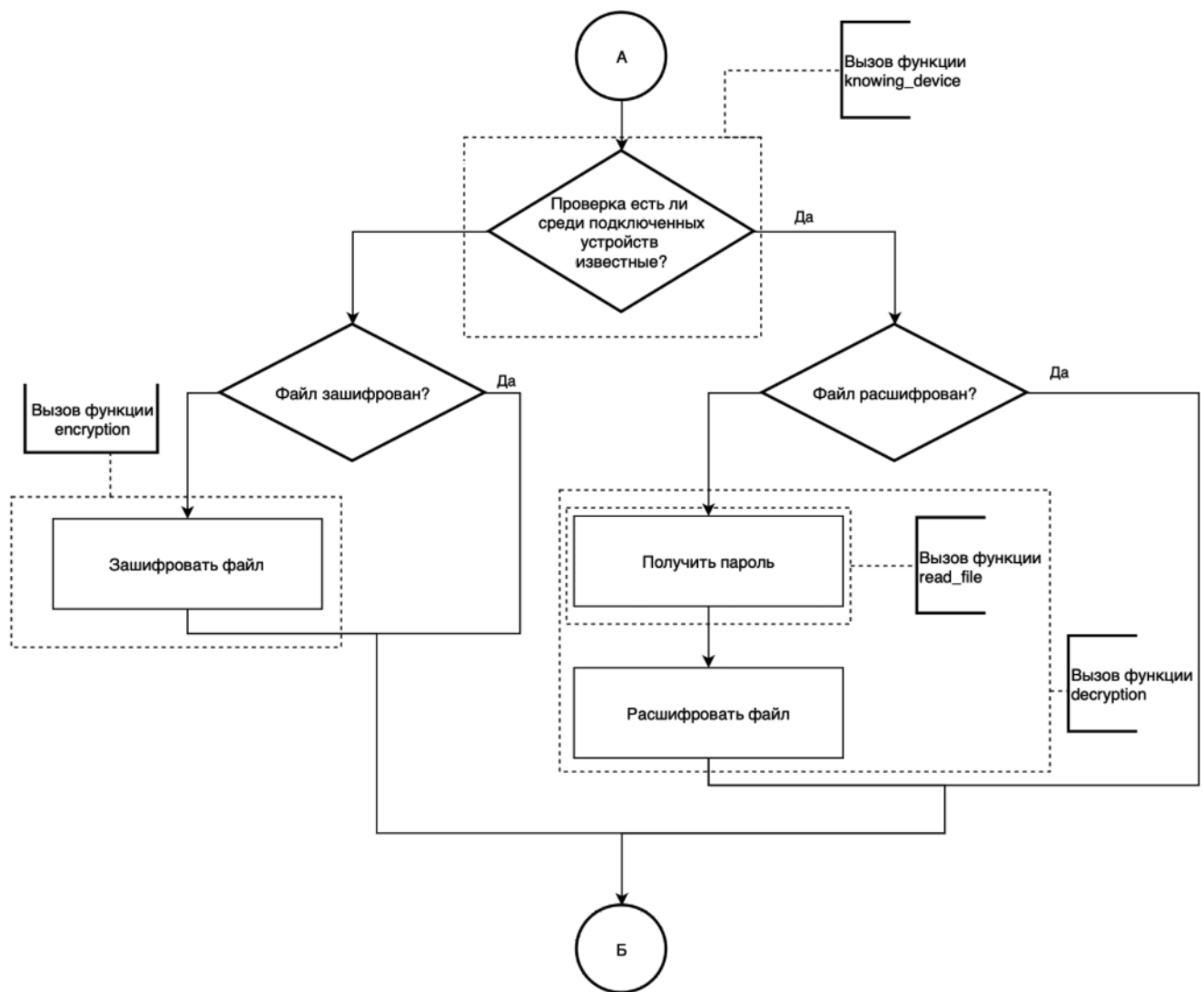


Рис. 2: Алгоритм работы функции-обработчика.

2.4 Алгоритм шифрования файла

В качестве алгоритма шифрования был выбран самый простой метод:

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

2.5 Вывод

Были рассмотрены все необходимые алгоритмы обработки и шифрования, метод хранения информации, а также был изучен способ перехвата сообщений. Структура программного обеспечения разработана и готова к реализации.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации.

3.1 Стек технологий

Для реализации был выбран язык программирования C. Компилятор – gcc. Для сборки был написан Makefile, позволяющий запускать сборку, одной командой – листинг 7.

3.2 Хранение данных

Параметры USB устройств, идентификатор поставщика и изделия, а также список секретных файлов и приложений хранятся в конфигурационном файле. Пример конфигурационного файла USB устройств представлен в листинге 8. Конфигурационный файл секретных файлов и приложений – листинг 9.

Пароль для доступа к зашифрованным данным хранится на разрешенном USB устройстве в файле *password.txt*.

3.3 Загружаемый модуль

Реализация загрузки и удаления представлена в листинге 10. После компиляции загружаемого модуля объектный файл может быть загружен в ядро с помощью команды *insmod* с правами суперпользователя, для выгрузки используется команда *rmmod*.

3.4 Функция-обработчик

В листинге 11 представлена реализация функции обратного вызова 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 Идентификация устройства

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 12.

Считывание пароля представлено в листинге 13.

После проверки принадлежности, при необходимости вызываются функции шифровки и расшифровки файлов, которые вызывают исполняемый файл пользовательского пространства.

Реализация этих функций представлена в листинге 14.

3.6 Вывод

Реализовано спроектированное программное обеспечение.

4 Исследовательская часть

В данном разделе будет проведен эксперимент. Также будут показаны примеры работы программы.

4.1 Системные характеристики

Характеристики компьютера на котором проводился эксперимент:

1. операционная система - Linux Ubuntu 18.04;
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz;
3. объем оперативной памяти - 16 ГБ;
4. количество ядер - 4;
5. количество логических процессов - 8;

4.2 Постановка эксперимента

В рамках данного проекта были проведены эксперименты, описанные ниже:

1. проверка защищенности секретных файлов при подключение опознанного устройства;
2. проверка защищенности секретных файлов при подключение неопознанного устройства;
3. Корректная обработка в случае, если подключено разрешенное и неразрешенное устройства одновременно.

4.3 Пример работы

4.4 Вывод

Все поставленные эксперименты прошли проверку. Разработанное программное обеспечение отвечает требованиям.

Заключение

В результате проделанной работы выполнены следующие задачи.

1. Определены основные понятия, такие как загружаемый модуль ядра, уведомления и уведомители. Рассмотрены структуры `usb_device`, `usb_device_id`.
2. Разработаны алгоритмы работы функции-обработчика и шифрования файлов.
3. Реализован загружаемого модуля.

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

Список литературы

- [1] Утечки данных 2019: статистика // <https://vc.ru/services/103616-utechki-dannyh-2019-statistika-tendencii-kiberbezopasnosti-i-mery-po-snizheniyu-riskov-vzloma> (дата обращения: 14.10.2021).
- [2] Анатомия загружаемых модулей ядра Linux // <https://www.ibm.com/developerworks/ru/library/l-lkm/index.html> (дата обращения: 15.10.2021).
- [3] Notification Chains in Linux Kernel // <https://0xax.gitbooks.io/linux-insides/content/Concepts/linux-cpu-4.html> (дата обращения: 25.10.2021)
- [4] `include/linux/usb.h` // <https://elixir.bootlin.com/linux/latest/source/include/linux/usb.hL2020> (дата обращения: 03.11.2021).
- [5] Doubly Linked Lists // <https://www.kernel.org/doc/html/v4.14/core-api/kernel-api.html> (дата обращения: 03.11.2021).
- [6] Invoking user-space applications from the kernel // <https://developer.ibm.com/technologies/linux/articles/l-user-space-apps/> (дата обращения: 04.11.2021).
- [7] Reading and writing of files in Linux kernel driver // <https://www.programmersought.com/article/83015124510/> (дата обращения: 04.11.2021).

Листинги

```
1 struct usb_device {
2     int      devnum;
3     char      devpath[16];
4     u32      route;
5     enum usb_device_state  state;
6     enum usb_device_speed  speed;
7     unsigned int      rx_lanes;
8     unsigned int      tx_lanes;
9
10    struct usb_tt      *tt;
11    int      ttport;
12
13    unsigned int toggle[2];
14
15    struct usb_device *parent;
16    struct usb_bus *bus;
17    struct usb_host_endpoint ep0;
18
19    struct device dev;
20
21    struct usb_device_descriptor descriptor;
22    struct usb_host_bos *bos;
23    struct usb_host_config *config;
24
25    struct usb_host_config *actconfig;
26    struct usb_host_endpoint *ep_in[16];
27    struct usb_host_endpoint *ep_out[16];
28
29    char **rawdescriptors;
30
31    unsigned short bus_mA;
32    u8 portnum;
33    u8 level;
34    u8 devaddr;
35
36    unsigned can_submit:1;
37    unsigned persist_enabled:1;
38    unsigned have_langid:1;
39    unsigned authorized:1;
40    unsigned authenticated:1;
41    unsigned wusb:1;
42    unsigned lpm_capable:1;
43    unsigned usb2_hw_lpm_capable:1;
44    unsigned usb2_hw_lpm_bes1_capable:1;
45    unsigned usb2_hw_lpm_enabled:1;
46    unsigned usb2_hw_lpm_allowed:1;
47    unsigned usb3_lpm_u1_enabled:1;
48    unsigned usb3_lpm_u2_enabled:1;
49    int string_langid;
```



```

50
51     /* static strings from the device */
52     char *product;
53     char *manufacturer;
54     char *serial;
55
56     struct list_head filelist;
57
58     int maxchild;
59
60     u32 quirks;
61     atomic_t urbnum;
62
63     unsigned long active_duration;
64
65 #ifdef CONFIG_PM
66     unsigned long connect_time;
67
68     unsigned do_remote_wakeup:1;
69     unsigned reset_resume:1;
70     unsigned port_is_suspended:1;
71 #endif
72     struct wusb_dev *wusb_dev;
73     int slot_id;
74     enum usb_device_removable removable;
75     struct usb2_lpm_parameters l1_params;
76     struct usb3_lpm_parameters u1_params;
77     struct usb3_lpm_parameters u2_params;
78     unsigned lpm_disable_count;
79
80     u16 hub_delay;
81     unsigned use_generic_driver:1;
82 };

```

Листинг 3: Структура usb_device

```

1 struct usb_device_id {
2     /* which fields to match against? */
3     __u16      match_flags;
4
5     /* Used for product specific matches; range is inclusive */
6     __u16      idVendor;
7     __u16      idProduct;
8     __u16      bcdDevice_lo;
9     __u16      bcdDevice_hi;
10
11     /* Used for device class matches */
12     __u8       bDeviceClass;
13     __u8       bDeviceSubClass;
14     __u8       bDeviceProtocol;
15
16     /* Used for interface class matches */

```

```

17     __u8        bInterfaceClass;
18     __u8        bInterfaceSubClass;
19     __u8        bInterfaceProtocol;
20
21     /* Used for vendor-specific interface matches */
22     __u8        bInterfaceNumber;
23
24     /* not matched against */
25     kernel_ulong_t driver_info
26         __attribute__((aligned(sizeof(kernel_ulong_t))));
27 };

```

Листинг 4: Структура usb_device_id

```

1 static void add_our_usb_device(struct usb_device *dev)
2 {
3     our_usb_device_t* new_usb_device = (our_usb_device_t *)kmalloc
4         (sizeof(our_usb_device_t), GFP_KERNEL);
5     struct usb_device_id new_id = { USB_DEVICE(dev->descriptor.idVendor,
6         dev->descriptor.idProduct) };
7     new_usb_device->dev_id = new_id;
8     list_add_tail(&new_usb_device->list_node, &connected_devices);
9 }

```

Листинг 5: Добавление usb устройства

```

1 static void delete_our_usb_device(struct usb_device *dev)
2 {
3     our_usb_device_t *device, *temp;
4     list_for_each_entry_safe(device, temp, &connected_devices, list_node)
5     {
6         if (device_match_device_id(dev, &device->dev_id))
7         {
8             list_del(&device->list_node);
9             kfree(device);
10        }
11    }
12 }

```

Листинг 6: Удаление usb устройства

```

1 ifneq ($(KERNELRELEASE),)
2     obj-m := md.o
3 else
4     CURRENT = $(shell uname -r)
5     KDIR = /lib/modules/$(CURRENT)/build
6     PWD = $(shell pwd)
7
8 default:
9     $(MAKE) -C $(KDIR) M=$(PWD) modules
10
11 clean:
12     rm -rf .tmp_versions

```

```

13     rm *.ko
14     rm *.o
15     rm *.mod.c
16     rm *.symvers
17     rm *.order
18
19 endif

```

Листинг 7: Makefile

```

1 struct known_usb_device {
2     struct usb_device_id dev_id;
3     char *name;
4 };
5
6 // List of all USB devices you know
7 static const struct known_usb_device known_devices[] = {
8     { .dev_id = { USB_DEVICE(0x0951, 0x1666) },
9       .name = "Kingston Technology DataTraveler G4" },
10 };

```

Листинг 8: Конфигурационный файл USB устройств

```

1 static char *secret_apps[] = {
2     "/home/jasur/projects/courseWork-OS/code/file.txt",
3     NULL,
4 };

```

Листинг 9: Конфигурационный файл секретных файлов и приложений

```

1 static int __init my_module_init(void)
2 {
3     usb_register_notify(&usb_notify);
4     call_encryption();
5     printk(KERN_INFO "USB MODULE: loaded.\n");
6     return 0;
7 }
8
9 static void __exit my_module_exit(void)
10 {
11     usb_unregister_notify(&usb_notify);
12     printk(KERN_INFO "USB MODULE: unloaded.\n");
13 }
14
15 module_init(my_module_init);
16 module_exit(my_module_exit);

```

Листинг 10: Загрузка и удаление модуля ядра

```

1 // If usb device inserted.
2 static void usb_dev_insert(struct usb_device *dev)
3 {
4     add_our_usb_device(dev);
5     char *name = knowing_device();

```

```

6
7     if (name)
8     {
9         if (state_encrypt)
10            call_decryption(name);
11        state_encrypt = false;
12        printk(KERN_INFO "USB MODULE: New device we can encrypt.\n");
13    }
14    else
15    {
16        if (!state_encrypt)
17            call_encryption();
18        state_encrypt = true;
19        printk(KERN_INFO "USB MODULE: New device, we can't encrypt.\n");
20    }
21 }
22
23 // If usb device removed.
24 static void usb_dev_remove(struct usb_device *dev)
25 {
26     delete_our_usb_device(dev);
27     char *name = knowing_device();
28
29     if (name)
30     {
31         if (state_encrypt)
32            call_decryption(name);
33        state_encrypt = false;
34        printk(KERN_INFO "USB MODULE: Delete device, we can encrypt.\n");
35    }
36    else
37    {
38        if (!state_encrypt)
39            call_encryption();
40        state_encrypt = true;
41        printk(KERN_INFO "USB MODULE: Delete device, we can't encrypt.\n");
42    }
43 }
44
45 // New notify.
46 static int notify(struct notifier_block *self, unsigned long action, void *dev)
47 {
48     // Events, which our notifier react.
49     switch (action)
50     {
51         case USB_DEVICE_ADD:
52             usb_dev_insert(dev);
53             break;
54         case USB_DEVICE_REMOVE:
55             usb_dev_remove(dev);
56             break;

```

```

57         default:
58             break;
59     }
60     return 0;
61 }

```

Листинг 11: Функция-обработчик

```

1 // Match device id with device id.
2 static bool device_id_match_device_id(struct usb_device_id *new_dev_id,
3                                     const struct usb_device_id *dev_id)
4 {
5     // Check idVendor and idProduct, which are used.
6     if (dev_id->idVendor != new_dev_id->idVendor)
7         return false;
8     if (dev_id->idProduct != new_dev_id->idProduct)
9         return false;
10    return true;
11 }
12
13 // Check our list of devices, if we know device.
14 static char *usb_device_id_is_known(struct usb_device_id *dev)
15 {
16     unsigned long known_devices_len = sizeof(known_devices) / sizeof(known_devices[0]);
17     int i = 0;
18     for (i = 0; i < known_devices_len; i++)
19     {
20         if (device_id_match_device_id(dev, &known_devices[i].dev_id))
21         {
22             int size = sizeof(known_devices[i].name);
23             char *name = (char *)kmallocc(size + 1, GFP_KERNEL);
24             int j = 0;
25             for (j = 0; j < size; j++)
26                 name[j] = known_devices[i].name[j];
27             name[size + 1] = '\0';
28
29             return name;
30         }
31     }
32     return NULL;
33 }
34
35 static char *knowing_device(void)
36 {
37     our_usb_device_t *temp;
38     int count = 0;
39     char *name;
40
41     list_for_each_entry(temp, &connected_devices, list_node) {
42         name = usb_device_id_is_known(&temp->dev_id);
43         if (!name)
44             return NULL;

```

```

45     count++;
46 }
47 if (0 == count)
48     return NULL;
49 return name;
50 }

```

Листинг 12: Функции для проверки разрешенных устройств

```

1 static char *read_file(char *filename)
2 {
3     struct kstat *stat;
4     struct file *fp;
5     mm_segment_t fs;
6     loff_t pos = 0;
7     char *buf;
8     int size;
9
10    fp = filp_open(filename, O_RDWR, 0644);
11    if (IS_ERR(fp))
12    {
13        return NULL;
14    }
15
16    fs = get_fs();
17    set_fs(KERNEL_DS);
18
19    stat = (struct kstat *)kmalloc(sizeof(struct kstat), GFP_KERNEL);
20    if (!stat)
21    {
22        return NULL;
23    }
24
25    vfs_stat(filename, stat);
26    size = stat->size;
27
28    buf = kmalloc(size, GFP_KERNEL);
29    if (!buf)
30    {
31        kfree(stat);
32        return NULL;
33    }
34
35    kernel_read(fp, buf, size, &pos);
36
37    filp_close(fp, NULL);
38    set_fs(fs);
39    kfree(stat);
40    buf[size]='\0';
41    return buf;

```

42 }

Листинг 13: Считывание пароля из файла USB устройства

```
1 static int call_decryption(char *name_device) {
2     printk(KERN_INFO "USB MODULE: Call_decrypt\n");
3
4     char path[80];
5     strcpy(path, USB_FOLDER);
6     strcat(path, name_device);
7     strcat(path, "/");
8     strcat(path, PASSWORD_FILE);
9     char *data = read_file(path);
10
11     char *argv[] = {
12         "/home/jasur/projects/courseWork-OS/code/crypto",
13         data,
14         NULL };
15
16     static char *envp[] = {
17         "HOME=",
18         "TERM=linux",
19         "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
20         NULL };
21
22     if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC) < 0)
23     {
24         return -1;
25     }
26
27     return 0;
28 }
29
30 static int call_encryption(void) {
31     printk(KERN_INFO "USB MODULE: Call_encrypt\n");
32     char *argv[] = {
33         "/home/jasur/projects/courseWork-OS/code/crypto",
34         NULL };
35
36     static char *envp[] = {
37         "HOME=",
38         "TERM=linux",
39         "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
40         NULL };
41
42     if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC) < 0)
43     {
44         return -1;
45     }
46
47     return 0;
```

Листинг 14: Функции вызывающие исполняемый файл пользовательского пространства