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**»

КАФЕДРА ИУ7 «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Загружаемый модуль ядра для защиты файла от изменения привилегированным пользователем»

Студент группы **ИУ7-76Б** _____ **П. А. Калашков**
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ **Н. Ю. Рязанова**
(Подпись, дата) (И.О. Фамилия)

***2023** 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 ИУ-7
_____ И. В. Рудаков
«16» сентября 2023 г.

ЗАДАНИЕ
на выполнение курсовой работы

по теме

«Загружаемый модуль ядра для защиты файла от изменения привилегированным пользователем»

Студент группы **ИУ7-76Б**

Калашков Павел Александрович

Направленность КР

учебная

Источник тематики

НИР кафедры

График выполнения КР: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание

Разработать загружаемый модуль ядра для защиты файла от изменения привилегированным пользователем. Обеспечить защиту файла от изменения владельца файла и группы пользователей, защиту от операций записи, переименования, перемещения и удаления.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на **12-20** листах формата А4.

Дата выдачи задания «16» сентября 2023 г.

Руководитель курсовой работы	_____	**Рязанова Н. Ю.**
	(Подпись, дата)	(Фамилия И. О.)
Студент	_____	**Калашков П. А.**
	(Подпись, дата)	(Фамилия И. О.)

РЕФЕРАТ

Расчетно-пояснительная записка 35 с., 5 рис., 15 ист.

ОПЕРАЦИОННЫЕ СИСТЕМЫ, ЗАГРУЖАЕМЫЙ МОДУЛЬ ЯДРА, ЗАЩИТА ФАЙЛА

Цель работы — разработка загружаемого модуля ядра для защиты файла от изменения привилегированым пользователем.

Разработанная программа выполняет поставленную задачу: защищает файл от воздействия привилегированного пользователя, а именно от операций записи в файл, операций переименования, перемещения и удаления, а также операций изменения доступа к файлу, владельца файла и группы пользователей файла.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В работе используются следующие термины с соответствующими определениями:

ОС — Операционная система

ПК — Персональный компьютер

ВВЕДЕНИЕ

Операционные системы, в основе которых лежит ядро Linux, являются одними из самых популярных ОС в мире: к апрелю 2023 года ими пользуются 42% пользователей мобильных устройств, 1.2% пользователей ПК [1], а также 40.23% разработчиков [2].

В администрировании UNIX-подобных ОС существует понятие root аккаунта, у которого есть доступ ко всем командам и файлам системы [3]. Захват root-прав или других привилегированных учетных данных повышает шанс хакеров остаться незамеченным в сети и получить доступ к важным системам и данным.

Целью работы является разработка загружаемого модуля ядра для защиты файла от изменения привилегированным пользователем. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ действий, защиту от которых нужно обеспечить;
2) провести анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;
3) разработать алгоритмы и структуру загружаемого модуля ядра, обеспечивающего защиту выбранного файла от изменения привилегированным пользователем, с учётом предотвращения работы функций write, rename, remove и операций изменения владельца файла и группы пользователей файла;

1 Аналитическая часть

В данном разделе проводится анализ действий, которые должна предотвращать разрабатываемая программа, а также структур ядра, необходимых для реализации загружаемого модуля.

1.1 Анализ предотвращаемых действий

Для того, чтобы разработать необходимую программу, требуется проанализировать выполнение в ОС Linux следующих действий:

— использование привилегий root аккаунта;

— запись в в файл;

— перемещение и переименование файла;

— удаление файла;

— изменение владельца файла и группы пользователей файла.

Анализ данных действий и дальнейшая работа будет производиться на основе дистрибутива Ubuntu 22.04, в основе которого лежит ядро Linux версии 5.16.0 для процессоров x86_64.

1.1.1 Использование привилегий root

root аккаунт (root-пользователь, суперпользователь) — аккаунт, у которого есть доступ ко всем командам и файлам системы. Привилегии root — возможности взаимодействия с системой, которые даёт root-аккаунт. root аккаунт обладает абсолютным контролем над системой: он имеет полный доступ ко всем файлам и командам, может изменять систему любым желаемым способом, а также разрешать или запрещать доступ к файлам другим пользователям системы (доступ на чтение, доступ на запись и доступ на исполнение файла) [3].

Существует команда sudo, позволяющая дать пользователю или группе пользователей временные (по умолчанию 5 минут) привилегии root аккаунта. Чтобы использовать данную команду, необходимо написать sudo, после чего написать команду, которую необходимо исполнить с использованием root-прав.

Таким образом, реализуемая программа должна защищать указанный файл

от воздействия команд, исполненных как без использования sudo, так и с использованием.

1.1.2 Запись в файл

В ОС Linux запись в файл, а также команды и программы, в той или иной мере изменяющие файл (команда перенаправления ввода-вывода, текстоовые редакторы) используют функцию write, имеющую заголовок, представленный на листинге 1.

Листинг 1 – Заголовок функции write

```
1 ssize_t write(int fd, const void buf[.count], size_t count);
```

Функция write записывает count байт из буфера, начинающегося по указателю buf в открытый файл с дескриптором fd [4].

Таким образом, чтобы предотвратить запись в указанный файл, необходимо предотвратить открытие данного файла в режиме записи. Для этого необходимо провести анализ структур ядра, при помощи которых описывается открытый файл.

1.1.3 Перемещение и переименования файла

Перемещение и переименование файла в ОС Linux может быть осуществлено при помощи команды mv, варианты использования которой приведены в листинге 1.

Листинг 2 – Варианты использования команды mv

```
1 mv [опции...] исходный_файл файл_назначения
2 mv [опции...] исходный_файл... каталог
```

Если последний аргумент является именем существующего каталога, то mv перемещает все остальные файлы в этот каталог. В противном случае, если задано только два файла, то имя первого файла будет изменено на имя второго.

Если последний аргумент не является каталогом и задано более чем два файла, то будет выдано сообщение об ошибке [5].

Так, mv /a/x/y /b переименует файл /a/x/y в /b/y, если /b является существующим каталогом, и в /b, если нет.

Для предотвращения перемещения и переименования указанного файла необходимо проанализировать структуры ядра, при помощи которых описывается элементы пути в файловой системе.

1.1.4 Изменение владельцев и группы пользователей файла

Для каждого объекта (файла, каталога, ссылки) в Linux и других Unix-подобных ОС действует система доступов [6]. Каждый объект обладает тремя типами доступов: доступ на чтение, доступ на запись (модификацию) и доступ на выполнение.

Каждый из этих доступов определён для трёх категорий пользователей:

— владелец;

— группа пользователей (т. е. набор пользователей, обладающих одинаковыми правами доступа), к которой принадлежит владелец файла;

— все остальные пользователи.

Доступы для конкретного файла могут быть представлены разными способами, один из них — текстовый, который предоставляет команда ls с опцией -l. В этом случае доступы представляются в виде 10-символьной строки, первый символ которой обозначает тип файла (например, дефис для обычного файла и *d* для каталога), а оставшиеся 9 символов состоят из трёх групп по 3 символа, представляющих доступы на чтение (символ *r* в случае наличия доступа), запись (символ *w*) и выполнение (символ *x*). В случае, если конкретного доступа нет, символом будет дефис.

Так, строка *-rwxrw-r--*, возвращаемая для объекта командой *ls - l* будет означать, что объект является обычным файлом, его владелец имеет доступ на чтение, запись, и выполнение, группа пользователей имеет доступ на чтение и запись, а все остальные пользователи имеют доступ только на чтение.

Доступы также могут быть представлены в численном виде как три восьмеричных числа. Значения этих чисел определены следующим образом: 0 для запрета всех типов доступа, 1 для доступа только на выполнение, 2 для доступа только на запись, 3 для доступа на запись и выполнение, 4 для доступа только на чтение, 5 для доступа на чтение и выполнение, 6 для доступа на чтение и запись, 7 для доступа на чтение, запись и выполнение.

Так, число 777 будет означать, что все пользователи имеют все доступы к данному файлу.

Для изменения доступов к конкретному файлу существует команда chmod, которая принимает необходимые доступы в численном или текстовоом виде, команда chown, меняющая владельца файла и команда chgrp, изменяющая группу пользователей файла.

Таким образом, для предотвращения изменения доступов к файлу (в том числе владельца файла и группы пользователей файла) необходимо проанализироовать структуры ядра, описывающие доступы к конкретному файлу.

1.2 Анализ структур ядра

Рассмотрим структуры ядра (ядро Linux версии 5.16.0), описывающие открытый файл и элемент пути в файловой системе.

1.2.1 struct inode

inode — структура ядра Linux, описывающая открытый файл. Она содержит в себе информацию, необходимую ядру для действий, связанных с файлом. На листингах 3–6 представлена структура inode с комментариями отдельных полей [7].

Листинг 3 – Структура inode, часть 1

```
1 struct inode {
2     umode_t i_mode;                    /* access permissions */
3     unsigned short i_opflags;         /* xattr support flags */
4     kuid_t i_uid;                       /* user id of owner */
```

Листинг 4 – Структура inode, часть 2

```
    kgid_t i_gid;                          /* group id of owner */
    unsigned int i_flags;                   /* filesystem flags */
#ifdef CONFIG_FS_POSIX_ACL                /* if FS supports ACL */
    struct posix_acl * i_acl;                        /* ACL mode */
    struct posix_acl * i_default_acl;           /* default ACL */
#endif
    const struct inode_operations *i_op;   /*inode operations */
    struct super_block *i_sb;             /* superblock pointer */
    struct address_space    *i_mapping; /* associated mapping */
#ifdef CONFIG_SECURITY               /* if with security module */
    void *i_security;                       /* security module */
#endif
    unsigned long i_ino;                         /* inode number */
    union {
        const unsigned int i_nlink;   /* number of hard links */
        unsigned int __i_nlink;       /* number of hard links */
    };
    dev_t i_rdev;                           /* real device node */
    loff_t i_size;                         /* file size in bytes */
    struct timespec64 i_atime;             /* last access time */
    struct timespec64 i_mtime;             /* last modify time */
    struct timespec64 i_ctime;             /* last change time */
    spinlock_t i_lock;                               /* spinlock */
    unsigned short i_bytes;                 /* bytes consumed */
    u8 i_blkbits;                          /* block size in bits */
    u8 i_write_hint;                        /* hint about block */
    blkcnt_t i_blocks;                    /* file size in blocks */
#ifdef __NEED_I_SIZE_ORDERED              /* sequence counters */
    seqcount_t i_size_seqcount;                      /* counter */
#endif
    unsigned long i_state;                       /* state flags */
```

Листинг 5 – Структура inode, часть 3

```
    struct rw_semaphore i_rwsem;                /* semaphore */
    unsigned long dirtied_when;  /* jiffies of first dirtying */
    unsigned long dirtied_time_when;         /* last dirtying */
    struct hlist_node i_hash;                    /* hash list */
    struct list_head i_io_list;        /* backing dev IO list */

#ifdef CONFIG_CGROUP_WRITEBACK /* flush dirty from page cache */
    struct bdi_writeback *i_wb;   /* the associated cgroup wb */
    /* foreign inode detection, see wbc_detach_inode() */
    int i_wb_frn_winner;
    u16 i_wb_frn_avg_time;
    u16 i_wb_frn_history;
#endif
    struct list_head i_lru;                /* inode LRU list */
    struct list_head i_sb_list;              /* sb of inodes */
    struct list_head i_wb_list;       /* backing dev wb list */
    union {
        struct hlist_head   i_dentry;    /* list of dentries */
        struct rcu_head     i_rcu;    /* list of rcu updates */
    };
    atomic64_t i_version;                  /* version number */
    atomic64_t i_sequence;                     /* see futex */
    atomic_t i_count;                    /* reference counter */
    atomic_t i_dio_count;                     /* DIO counter */
    atomic_t i_writecount;                /* writers counter */
#if defined(CONFIG_IMA) || defined(CONFIG_FILE_LOCKING) /*IMA*/
    atomic_t i_readcount;             /* struct files open RO */
#endif
    union {
        const struct file_operations *i_fop /*file operations*/
        void (*free_inode)(struct inode *); /* free function */
    };
```

Листинг 6 – Структура inode, часть 4

```
    struct file_lock_context *i_flctx;          /* for fcntl */
    struct address_space i_data;       /* mapping for device */
    struct list_head    i_devices;  /* list of block devices */
    union {
        struct pipe_inode_info *i_pipe;  /* pipe information */
        struct cdev *i_cdev;                  /*char devices */
        char *i_link;                            /* symlink */
        unsigned i_dir_seq;          /* opencoded seqcounter */
    };

    __u32 i_generation;              /* inode version number */

#ifdef CONFIG_FSNOTIFY
    __u32 i_fsnotify_mask;  /* all events inode cares about */
    struct fsnotify_mark_connector __rcu    *i_fsnotify_marks;
#endif

#ifdef CONFIG_FS_ENCRYPTION
    struct fscrypt_info    *i_crypt_info;
#endif

#ifdef CONFIG_FS_VERITY
    struct fsverity_info *i_verity_info;
#endif

    void *i_private;       /* fs or device private pointer */
} __randomize_layout;
```

Для предотвращения опрерации открытия файла в режиме записи, а также самой записи необходимо изменить содержимое поля i_fop. Структура file_operations содержит 4 поля, имеющих значение для поставленной задачи (листинг 7).

Листинг 7 – Структура file_operations

```
struct file_operations {
    // ...
    ssize_t (*read) (struct file *, char __user *,
        size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *,
        size_t, loff_t *);
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
    int (*open) (struct inode *, struct file *);
    // ...
} __randomize_layout;
```

Структура file_operations содержит поля read и write, в которых хранятся указатели на функции, вызывающиеся для чтения и записи данных соответственно. Также в ней содержатся поля read_iter и write_iter, содержащие указатели на функции, вызывающиеся перед read и write соответственно. Для предотвращения записи в файл необходимо переопределить значение поля write_iter таким образом, чтобы в случае, когда запись возможна, возвращался флаг EACCES, означающий запрет на запись.

Для предотвращения открытия файла в режиме записи необходимо переопределить значение поля open таким образом, чтобы в случае открытия файла на чтение работа осуществлялась без изменений, а в случае открытия файла на запись возвращался флаг EACCES.

Для предотвращения изменений доступов к файлу, владельца файла и группы пользователей файла необходимо обратиться к полю i_op в inode. В нём есть поле setattr, содержащее указатель на функцию изменения таких атрибутов inode, как i_uid, i_gid, а также i_mode (листинг 8).

Листинг 8 – Структура inode_operations

```
struct inode_operations {
    // ...
    int (*setattr) (struct user_namespace *, struct dentry *,
            struct iattr *);
    // ...
} ____cacheline_aligned;
```

Для корректной работы разрабатываемого модуля ядра необходимо, чтобы после выгрузки модуля значения изменённых полей возвращались обратно. Для этого необходимо предусмотреть сохранение исходных объектов полей i_fop и i_op.

1.2.2 struct dentry

dentry — структура ядра Linux, описывающая элемент пути в файловой системе. При таких операциях, как переименование и перемещение файла, взаимодействие происходит именно с объектами dentry, а не с inode. В листингах 9–9 представлена структура dentry с комментариями полей [8].

Листинг 9 – Структура dentry, часть 1

```
struct dentry {
    unsigned int d_flags;                       /* dentry flags */
    seqcount_spinlock_t d_seq;                 /* entry seqlock */
    struct hlist_bl_node d_hash;/* list of hash table entries */
    struct dentry *d_parent;                /* parent directory */
    struct qstr d_name;                          /* dentry name */
    struct inode *d_inode;                  /* associated inode */
    unsigned char d_iname[DNAME_INLINE_LEN];   /* small names */
    struct lockref d_lockref; /* per-dentry lock and refcount */
    const struct dentry_operations *d_op;       /* operations */
```

Листинг 10 – Структура dentry, часть 2

```
    struct super_block *d_sb;  /* The root of the dentry tree */
    unsigned long d_time;                   /* revalidate time */
    void *d_fsdata;                        /* fs-specific data */

    union {
        struct list_head d_lru;                    /* LRU list */
        wait_queue_head_t *d_wait;     /* in-lookup ones only */
    };
    struct list_head d_child;          /* child of parent list */
    struct list_head d_subdirs;                /* our children */

    union {
        struct hlist_node d_alias;         /* inode alias list */
        struct hlist_bl_node d_in_lookup_hash;   /* in-lookup */
         struct rcu_head d_rcu;                  /* RCU locking */
    } d_u;
} __randomize_layout;
```

Поле d_flags, содержащее флаги файловой системы, позволяет определить возможность изменения прав доступа для описываемого файла. Существует флаг FS_IMMUTABLE_FL, запрещающий изменение полей i_uid, i_gid, а также i_mode у inode, указатель на который содержится в поле d_inode. Установка данного флага позволит предотвратить изменение владельца файла, группы пользователей файла, а также доступы к файлу.

После выгрузки разрабатываемого модуля ядра необходимо, чтобы значение поля d_flags устанавливалось в исходное значение.

1.2.3 Регистры процессоров Intel

Страницы памяти, в которых хранятся структуры file_operations и inode_operations помечены как read-only. В процессорах, построенных по ар-

хитектуре x86, существует понятие регистров контроля (англ. *Control Registers, CR*), которые содержат значения флагов, относящихся к защите памяти, многопоточности, разбиению памяти на страницы и т. д. [9]. 16-й бит регистра контроля CR0 отвечает за возможность изменения страниц, помеченных как read-only в режиме суперпользователя. Когда он установлен, суперпользователь не может изменять страницы, помеченные как read-only.

На некоторых процессорах x86 прозводства Intel данный бит по умолчанию установлен, поэтому прежде чем заменять значения полей i_op и i_fop на новые, необходимо выключить данный флаг и установить его в исходное значение после выполнения необходимых операций.

Для улучшения безопасности регистры CR0 и CR4 в ОС Linux с версии 5.3 являются защищёнными, и функция write_cr0, с помощью которой можно было установить значение регистра CR0, уже не входит в набор предоставляемых разработчикам функций. Необходимо найти альтернативный способ установки значения регистра CR0 при помощи ассемблерных вставок из загружаемого модуля ядра.

Вывод

В данном разделе был проведён анализ действий, которые должна предотвращать разрабатываемая программа, а также структур ядра, необходимых для реализации загружаемого модуля. Определены действия, которые необходимо предотвращать: команды записи (использование вызова write и текстовго редактора), команд chmod, chown и chgrp, изменяющих доступы к файлу, а также команды mv, перемещающей или переименовывающей файл. Данные действия необходимо предотвращать, даже если они выполнены при помощи команды sudo, дающей привилегии суперпользователя.

Для предотвращения открытия и записи в файл необходимо переопределить исходные значения полей open и write_iter структуры file_operations в поле i_fop у inode, относящегося к файлу. Для предотвращения изменения доступа к файлу, а также владельца файла и группы пользователей, необходимо пере-

определить заначение поля setattr структуры inode_operations в поле i_op того же inode.

Для предотвращения перемещения и переименования файла необходимо обеспечить неизменяемость соответствующего файлу объекта dentry при помощи установки флага FS_IMMUTABLE_FL в поле d_flags.

Для корректной работы программы необходимо обеспечить работу с регистром контроля CR0 при помощи ассеммблерных вставок для возможности изменить необходимые поля у inode. При выгрузке разработанного модуля ядра значения изменённых полей должны возвращаться к исходным.

2 Конструкторская часть

В данном разделе рассматривается спроектированная база данных, соответствующая выбранной в аналитической части СУБД, приводится диаграмма базы данных, рассматриваются сценарии создания таблиц БД и основные используемые паттерны проектирования.

Для таблиц необходимо описать информацию о стоблцах (типы данных, ограничения, значения), а также описать связь между описанными таблицами при помощи диаграммы базы данных.

2.1 Разработка алгоритмов

На рисунке 1 представлена схема алгоритма функции загрузки модуля ядра, на рисунке 2 представлена схема алгоритма функции создания нового объекта file_operations и схемы алгоритмов функций write_iter_hook и open_hook, а на рисунке 3 представлена схема алгоритма функции создания нового объекта inode_operations и схема алгоритма функции setattr_hook. На рисунке 4 представлена схема алгоритма функции выгрузки загружаемого модуля.

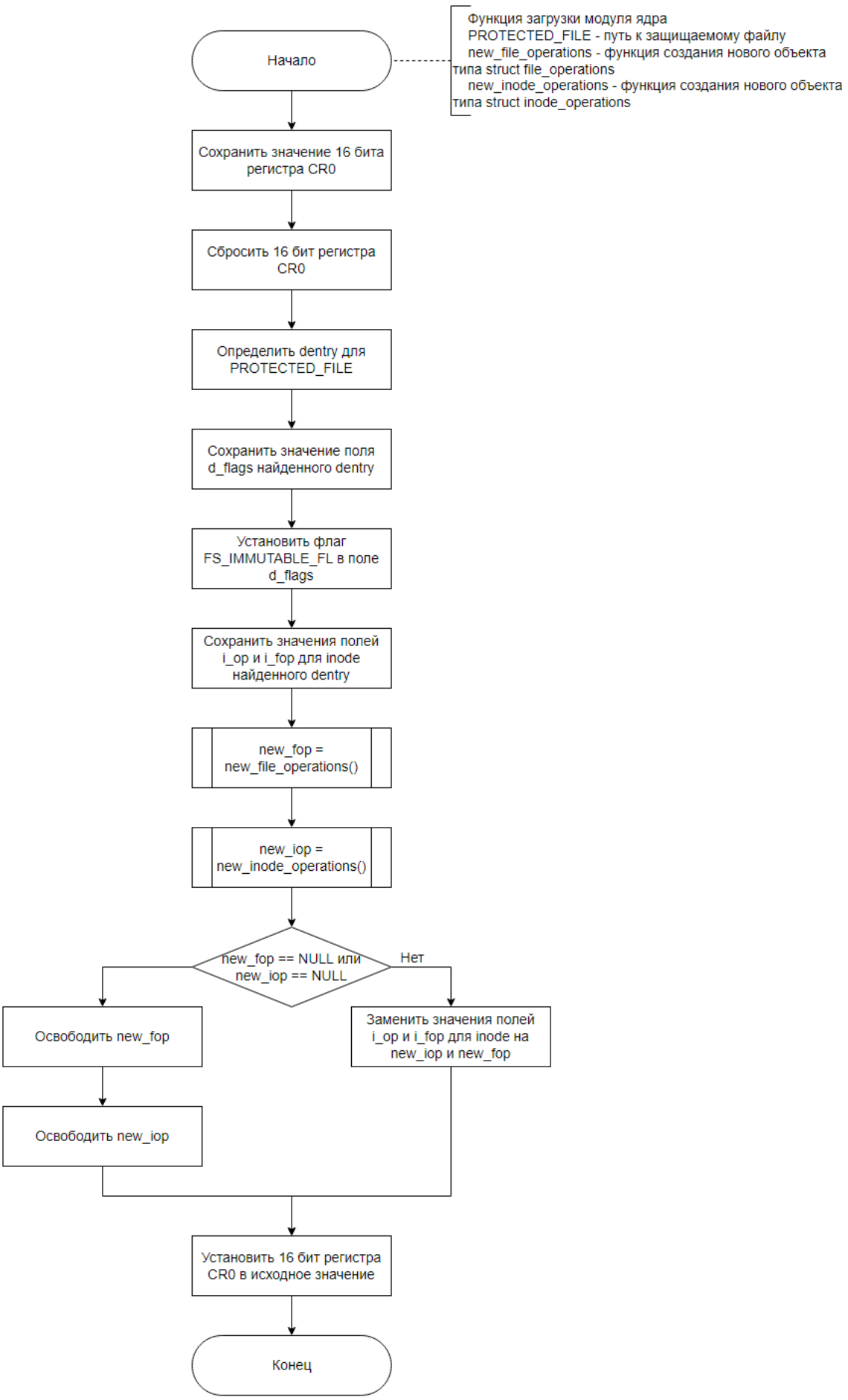

Рисунок 1 – Схема алгоритма функции загрузки модуля ядра

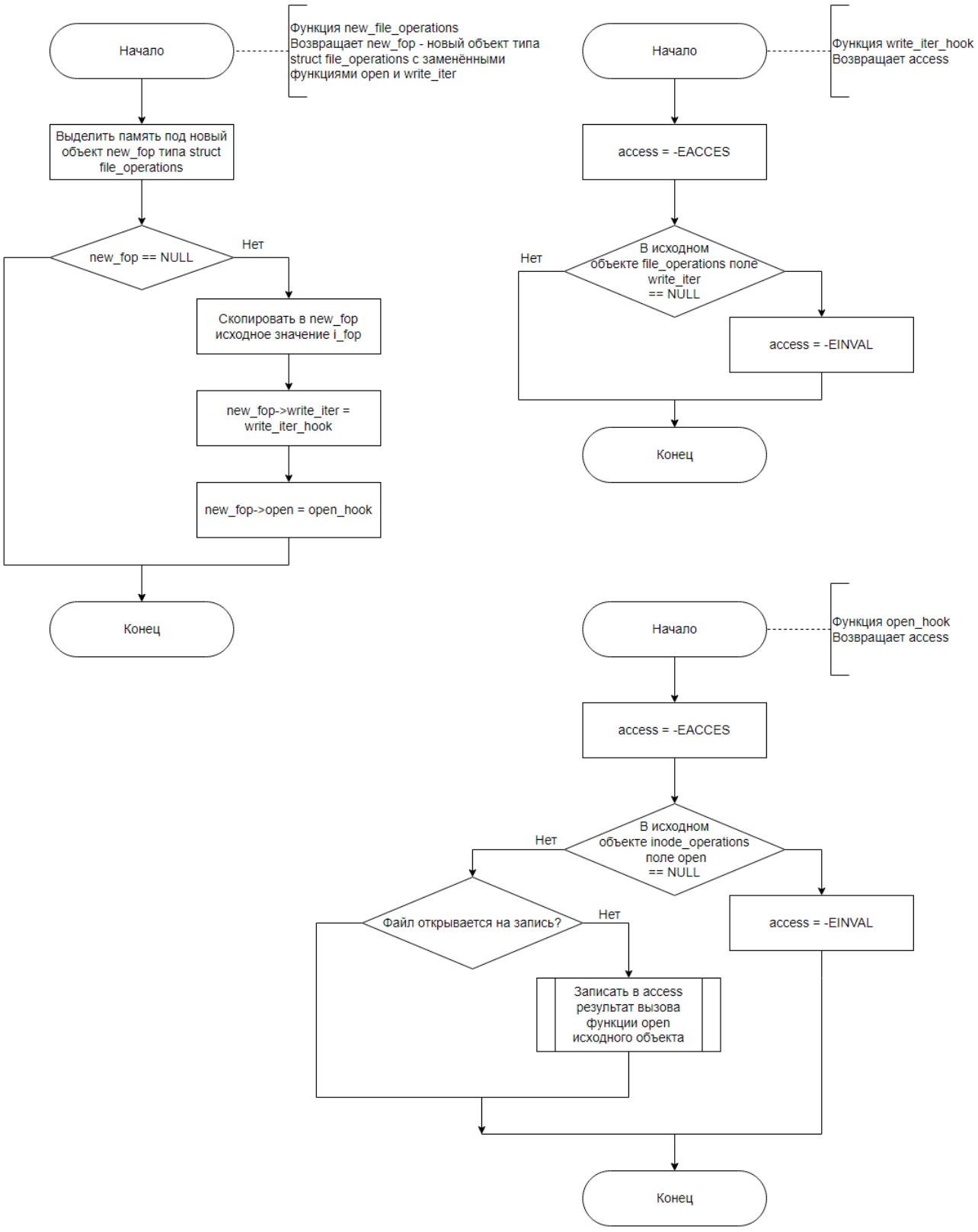

Рисунок 2 – Схемы алгоритмов создания нового объекта file_operations и функций write_iter_hook и open_hook

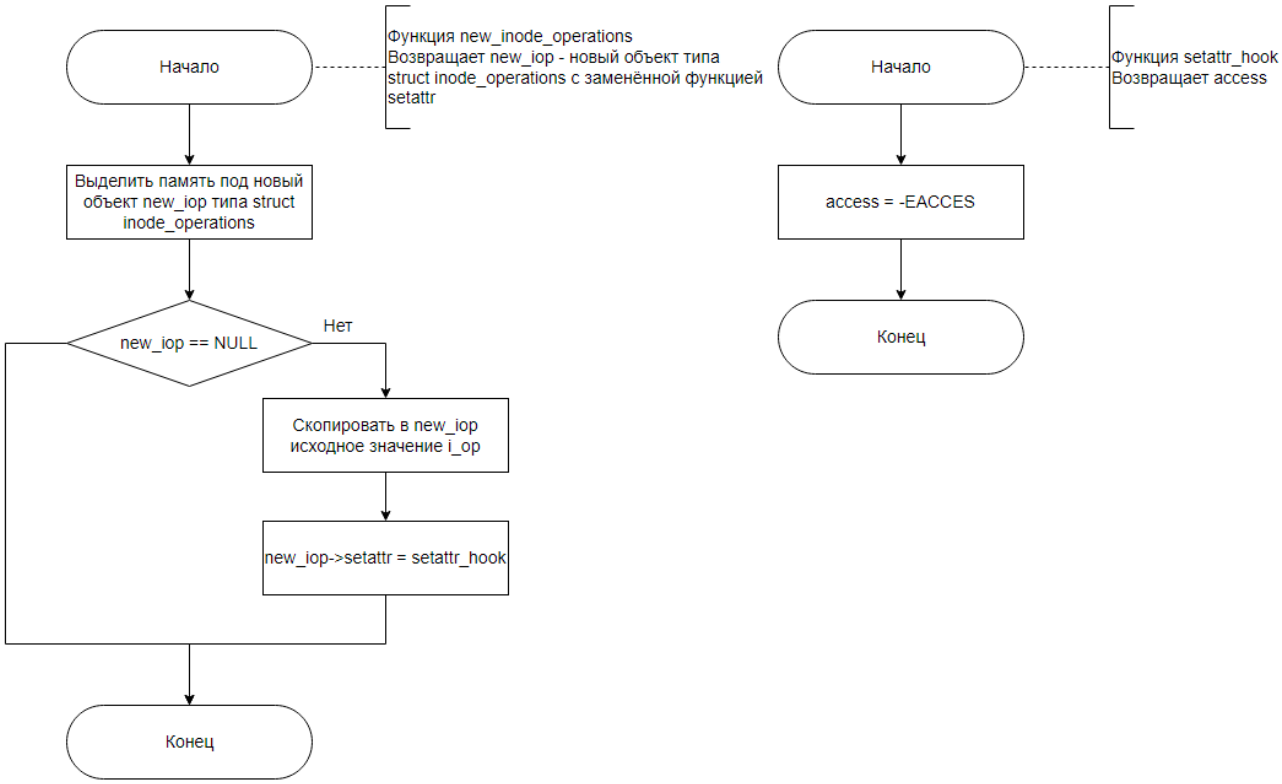

Рисунок 3 – Схемы алгоритма создания нового объекта inode_operations и функциии setattr_hook

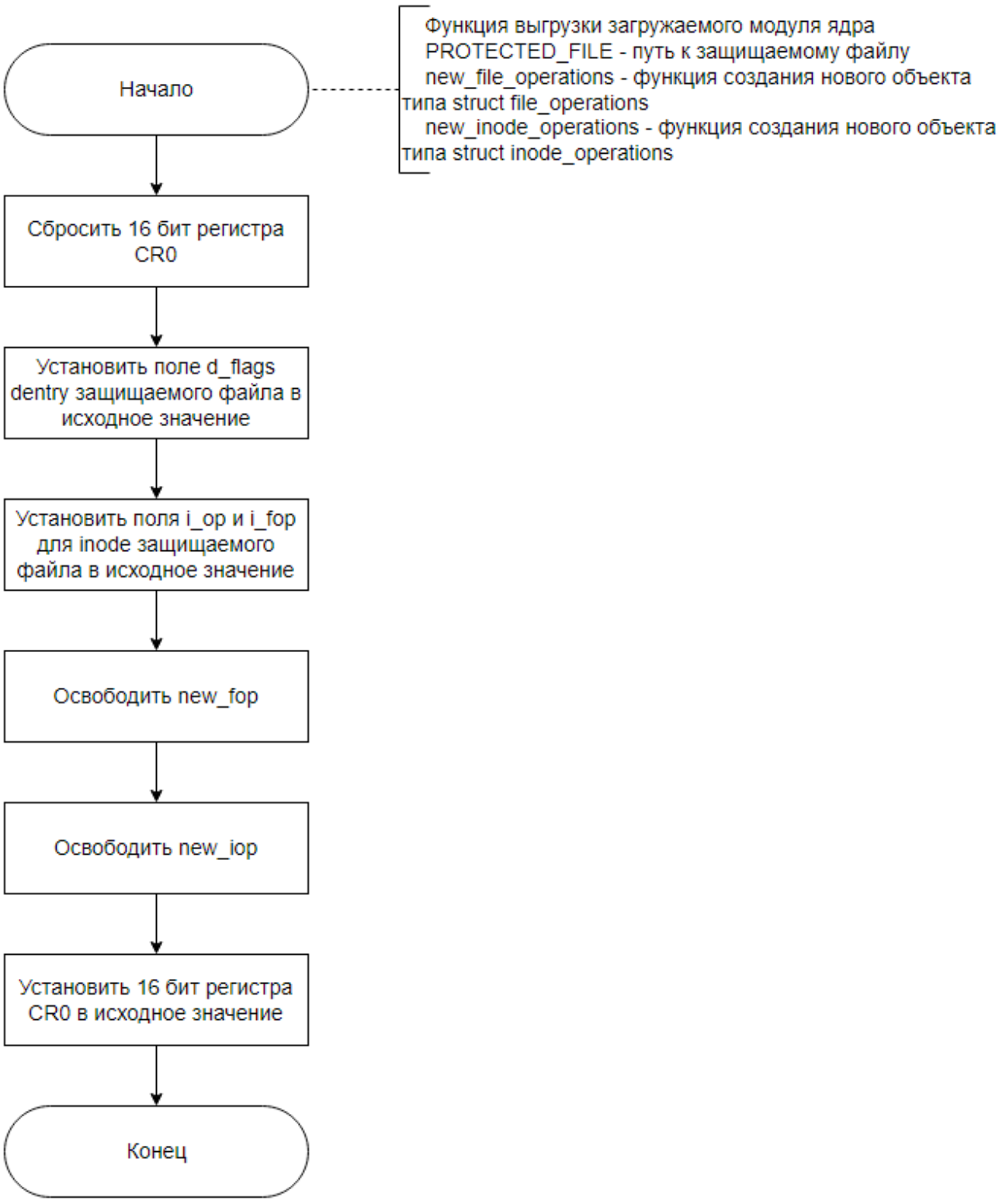

Рисунок 4 – Схема алгоритма выгрузки загружаемого модуля

3 Технологическая часть

В данном разделе рассмотрены средства разработки программного обеспечения, приведены листинги исходного кода загружаемого модуля ядра.

3.1 Средства реализации

Как основное средство реализации и разработки ПО был выбран язык программирования Си [10], поскольку в нём есть все инструменты для реализации загружаемого модудя ядра. Средой разработки послужил графический редактор Visual Studio Code [11], который известен содержанием большого количество плагинов, ускоряющих процесс разработки программы.

3.2 Реализация загружаемого модуля ядра

В расположенных ниже листингах 11 – 15 приведена реализация загружаемого модуля ядра согласно разработанным алгоритмам.

Листинг 11 – Реализация загружаемого модуля ядра, часть 1

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/unistd.h>
#include <asm/uaccess.h>
#include <linux/namei.h>
#include <asm/string.h>

#define PROTECTED_FILE "/home/paul/Desktop/bmstu-os-cw/\
    src/protected"
MODULE_LICENSE ("GPL");

struct dentry* protected_dentry;
struct inode* protected_inode;
struct file_operations *original_fop, *new_fop;
struct inode_operations *original_iop, *new_iop;
unsigned int initial_dentry_flags;
int permissions;
unsigned long initial_cr0;

inline void write_cr0_custom(unsigned long cr0)
{
    unsigned long __force_order;
    asm volatile("mov %0,%%cr0" : "+r"(cr0),
                 "+m"(__force_order));
}

void disable_write_protection(void)
{
    write_cr0_custom(initial_cr0 & (~0x10000));
}
```

Листинг 12 – Реализация загружаемого модуля ядра, часть 2

```
void enable_write_protection(void)
{
    write_cr0_custom(initial_cr0);
}

struct inode* protect_inode(void)
{
    int error;
    struct inode *inode;
    struct path protected_path;
    error = kern_path(PROTECTED_FILE, LOOKUP_FOLLOW,
                                      &protected_path);
    if (!error)
    {
        protected_dentry = protected_path.dentry;
        inode = protected_path.dentry->d_inode;
        initial_dentry_flags = protected_path.dentry->d_flags;
        protected_path.dentry->d_flags = FS_IMMUTABLE_FL;
        return inode;
    }
    printk(KERN_INFO"kern_path error %llx\n", protected_path);
    return NULL;
}

ssize_t write_iter_hook(struct kiocb *X, struct iov_iter *Y)
{
    ssize_t access = -EACCES;
    if(original_fop->write_iter == NULL)
        access = -EINVAL;
    return access;
}
```

Листинг 13 – Реализация загружаемого модуля ядра, часть 3

```
int open_hook(struct inode* inode, struct file* filep)
{
    int access = -EACCES;
    if(original_fop->open == NULL)
        access = -EINVAL;
    else if (!(filep->f_mode & FMODE_WRITE))
        access =  original_fop->open(inode, filep);
    return access;
}

int setattr_hook(struct user_namespace *un, struct dentry *d,
                                            struct iattr *attr)
{
    return -EACCES;
}

struct file_operations* new_file_operations(void)
{
    struct file_operations* new_fop = kmalloc(
        sizeof(struct file_operations), GFP_KERNEL);
    if (new_fop)
    {
        memcpy(new_fop, original_fop,
                sizeof(struct file_operations));
        new_fop->write_iter = write_iter_hook;
        new_fop->open = open_hook;
    }
    return new_fop;
}
```

Листинг 14 – Реализация загружаемого модуля ядра, часть 4

```
struct inode_operations* new_inode_operations(void)
{
    struct inode_operations* new_iop = kmalloc(
        sizeof(struct inode_operations), GFP_KERNEL);
    if (new_iop)
    {
        memcpy(new_iop, original_iop,
                sizeof(struct inode_operations));
        new_iop->setattr = setattr_hook;
    }
    return new_iop;
}

static int __init mod_init (void)
{
    initial_cr0 = read_cr0();
    disable_write_protection();
    protected_inode = protect_inode();
    if (protected_inode == NULL)
    {
        printk(KERN_INFO"Cannot get inode address\n\
                Maybe protected file doesn't exist\n");
        enable_write_protection();
        return -1;
    }
    original_fop = protected_inode->i_fop;
    original_iop = protected_inode->i_op;
    new_fop = new_file_operations();
    new_iop = new_inode_operations();
    if (!new_fop || ! new_iop)
    {
        kfree(new_fop);
```

Листинг 15 – Реализация загружаемого модуля ядра, часть 5

```
        kfree(new_iop);
    }
    else
    {
        protected_inode->i_fop = new_fop;
        protected_inode->i_op = new_iop;
    }
    enable_write_protection();
    return 0;

}

static void mod_exit (void)
{
    disable_write_protection();
    protected_dentry->d_flags = initial_dentry_flags;
    protected_inode->i_fop = original_fop;
    protected_inode->i_op = original_iop;
    enable_write_protection();
    kfree(new_fop);
    kfree(new_iop);
}

module_init(mod_init);
module_exit(mod_exit);
```

Вывод

В данном разделе были рассмотрены средства разработки программного обеспечения и приведены листинги исходного кода загружаемого модуля ядра.

4 Исследовательская часть

В данном разделе будут приведены примеры работы загружаемого модуля ядра и демонстрация защиты файла от описанных в аналитическом разделе действий.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система: Ubuntu 20.04.6 LTS [12];

— Память: 16 Гб с тактовой частотой 2133 МГц LPDDR3 [13];

— Процессор: Intel Core™ i7-8559U [14] с тактовой частотой 2.70 ГГц;

— Видеокарта: NVIDIA GeForce GTX 1060 [15]

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только системой тестирования (работающим приложением) и системным окружением операционной системы.

4.2 Демонстрация работы программы

На рисунке 5 продемонстрирована работа загружаемого модуля ядра: осуществляется загрузка модуля, демонстрация исходного содержимого защищаемого файла и доступы к нему, производятся попытки осуществить запись в файл, переместить, переименовать, удалить файл, а также поменять доступы к нему, изменить владельца файла и группу пользователей файла (все действия были выполнены с правами суперпользователя). После этого производится повторная демонстрация содержимого файла и прав доступа к нему: содержимое и права доступа не изменились, защита работает. После выгрузки модуля ядра защита перестаёт работать и действия работают корректно.

```
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo insmod file_protection.ko
paul@paul:~/Desktop/bmstu-os-cw/src$ cat protected
Some content
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo echo I changed this file > protected
bash: protected: No such file or directory
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo mv protected renamed_protected
mv: cannot move 'protected' to 'renamed_protected': No such file or directory
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo mv protected ..
mv: cannot move 'protected' to '../protected': No such file or directory
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo rm protected
rm: cannot remove 'protected': No such file or directory
paul@paul:~/Desktop/bmstu-os-cw/src$ ls -l protected
-rw-rw-rw- 1 paul paul 13 дек 22 13:27 protected
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chmod 777 protected
chmod: changing permissions of 'protected': Permission denied
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chown root protected
chown: changing ownership of 'protected': Permission denied
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chgrp sudo protected
chgrp: changing group of 'protected': Permission denied
paul@paul:~/Desktop/bmstu-os-cw/src$ ls -l protected
-rw-rw-rw- 1 paul paul 13 дек 22 13:27 protected
paul@paul:~/Desktop/bmstu-os-cw/src$ cat protected
Some content
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo rmmod file_protection
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo echo I changed this file > protected
paul@paul:~/Desktop/bmstu-os-cw/src$ cat protected
I changed this file
paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chmod 777 protected
paul@paul:~/Desktop/bmstu-os-cw/src$ ls -l protected
-rwxrwxrwx 1 paul paul 20 дек 22 13:30 protected
paul@paul:~/Desktop/bmstu-os-cw/src$
```

Рисунок 5 – Демонстрация работы программы

Вывод

В данном разделе были приведены примеры работы загружаемого модуля ядра и демонстрация защиты файла от описанных в аналитическом разделе действий.

Разработанная программа выполняет поставленную задачу: защищает файл от воздействия привилегированного пользователя, а именно от операций записи в файл, операций переименования, перемещения и удаления, а также операций изменения доступа к файлу, владельца файла и группы пользователей файла.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработанн загружаемый модуль ядра для защиты файла от воздействия привилегированного процесса, а именно от операций записи в файл, операций переименования, перемещения и удаления, а также операций изменения доступа к файлу, владельца файла и группы пользователей файла.

Были выполнены следующие задачи:

1) проведён анализ действий, защиту от которых нужно обеспечить;
2) проведён анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;
3) разработаны алгоритмы и структура загружаемого модуля ядра, обеспечивающего защиту выбранного файла от изменения привилегированным пользователем, с учётом предотвращения работы функций write, rename, remove и операций изменения владельца файла и группы пользователей файла;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Operating System Market Share Worldwide [Электронный ресурс]. — Режим доступа: https://gs.statcounter.com/os-market-share#monthly-202111-202303 (дата обращения: 07.10.2023).

2. Stack Overflow Developer Survey 2022 [Электронный ресурс]. — Режим доступа: https://survey.stackoverflow.co/2022/#operating-system (дата обращения: 07.10.2023).

3. root Definition, The Linux Information Project [Электронный ресурс]. — Режим доступа: https://www.linfo.org/root.html (дата обращения: 07.10.2023).

4. System Calls Manual, write (2) [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man2/write.2.html (дата обращения: 21.12.2023).

5. System Calls Manual, mv (1) [Электронный ресурс]. — Режим доступа: https://www.opennet.ru/man.shtml?topic=mv&category=1&russian=0 (дата обращения: 21.12.2023).

6. Permissions Definition, The Linux Information Project [Электронный ресурс]. — Режим доступа: https://www.linfo.org/permissions.html (дата обращения: 21.12.2023).

7. Bootlin, Elixir Cross Referencer, struct inode [Электронный ресурс]. — Режим доступа: https://elixir.bootlin.com/linux/v5.16/source/include/linux/fs.h#L620 (дата обращения: 21.12.2023).

8. Bootlin, Elixir Cross Referencer, struct dentry [Электронный ресурс]. — Режим доступа: https://elixir.bootlin.com/linux/v5.16/source/include/linux/dcache.h#L91 (дата обращения: 21.12.2023).

9. Intel's Software Developer's Manual [Электронный ресурс]. — Режим доступа: https://software.intel.com/content/dam/develop/public/us/en/documents/325384-sdm-vol-3abcd.pdf (дата обращения: 21.12.2023).

10. Документация по языку C. Узнайте, как использовать C и библиотеку времени выполнения C [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/c-language/?view=msvc-170 (дата обращения: 22.12.2023).

11. Visual Studio Code — Code Editing. Redefined [Электронный ресурс]. — Режим доступа: https://code.visualstudio.com (дата обращения: 22.12.2023).

12. macOS Ventura — official site [Электронный ресурс]. — Режим доступа: https://releases.ubuntu.com/focal/ (дата обращения: 22.12.2023).

13. LPDDR3 SDRAM Documentation [Электронный ресурс]. — Режим доступа: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf (дата обращения: 17.05.2023).

14. Процессор Intel Core™ i7-8559U [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/137979/intel-core-i78559u-processor-8m-cache-up-to-4-50-ghz.html (дата обращения: 17.05.2023),

15. GeForce GTX 1060 Specifications [Электронный ресурс]. — Режим досту-

па: https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1060/specifications/ (дата обращения: 22.12.2023),