



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

(Подпись, дата)

П. А. Калашков

(И.О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Н. Ю. Рязанова

(И.О. Фамилия)

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 20 с., 0 рис., 0 табл., TODO ист.

ОПЕРАЦИОННЫЕ СИСТЕМЫ, ЗАГРУЖАЕМЫЙ МОДУЛЬ ЯДРА, TODO

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

Результаты: TODO

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 Аналитическая часть	7
1.1 Анализ предотвращаемых действий	7
1.1.1 Использование привилегий root	7
1.1.2 Запись в файл	8
1.1.3 Перемещение и переименования файла	8
1.1.4 Изменение владельцев и группы пользователей файла	9
1.2 Анализ структур ядра	10
1.2.1 struct inode	10
1.2.2 struct dentry	15
1.2.3 Регистры процессоров Intel	16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	19

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 предметной области;
- 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;
- 4) исследовать разработанную программу.

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 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 описываются 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 `-rwxrwx-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 текстовом 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 проанализировать 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

```
1      kgid_t i_gid;                                /* group id of owner */
2      unsigned int i_flags;                        /* filesystem flags */
3      #ifdef CONFIG_FS_POSIX_ACL                    /* if FS supports ACL */
4          struct posix_acl * i_acl;                /* ACL mode */
5          struct posix_acl * i_default_acl;        /* default ACL */
6      #endif
7      const struct inode_operations *i_op;         /*inode operations */
8      struct super_block *i_sb;                   /* superblock pointer */
9      struct address_space *i_mapping; /* associated mapping */
10     #ifdef CONFIG_SECURITY                         /* if with security module */
11         void *i_security;                        /* security module */
12     #endif
13     unsigned long i_ino;                          /* inode number */
14     union {
15         const unsigned int i_nlink; /* number of hard links */
16         unsigned int __i_nlink; /* number of hard links */
17     };
18     dev_t i_rdev;                                /* real device node */
19     loff_t i_size;                               /* file size in bytes */
20     struct timespec64 i_atime;                   /* last access time */
21     struct timespec64 i_mtime;                   /* last modify time */
22     struct timespec64 i_ctime;                   /* last change time */
23     spinlock_t i_lock;                          /* spinlock */
24     unsigned short i_bytes;                      /* bytes consumed */
25     u8 i_blkbits;                               /* block size in bits */
26     u8 i_write_hint;                            /* hint about block */
27     blkcnt_t i_blocks;                          /* file size in blocks */
28     #ifdef __NEED_I_SIZE_ORDERED                  /* sequence counters */
29         seqcount_t i_size_seqcount;              /* counter */
30     #endif
31     unsigned long i_state;                      /* state flags */
```

Листинг 5 – Структура inode, часть 3

```

1      struct rw_semaphore i_rwsem;                /* semaphore */
2      unsigned long dirtied_when; /* jiffies of first dirtying */
3      unsigned long dirtied_time_when;            /* last dirtying */
4      struct hlist_node i_hash;                   /* hash list */
5      struct list_head i_io_list;                 /* backing dev IO list */
6
7      #ifdef CONFIG_CGROUP_WRITEBACK /* flush dirty from page cache */
8          struct bdi_writeback *i_wb; /* the associated cgroup wb */
9          /* foreign inode detection, see wbc_detach_inode() */
10         int i_wb_frn_winner;
11         ul6 i_wb_frn_avg_time;
12         ul6 i_wb_frn_history;
13     #endif
14     struct list_head i_lru;                       /* inode LRU list */
15     struct list_head i_sb_list;                   /* sb of inodes */
16     struct list_head i_wb_list;                   /* backing dev wb list */
17     union {
18         struct hlist_head i_dentry; /* list of dentries */
19         struct rcu_head i_rcu; /* list of rcu updates */
20     };
21     atomic64_t i_version;                         /* version number */
22     atomic64_t i_sequence;                        /* see futex */
23     atomic_t i_count;                             /* reference counter */
24     atomic_t i_dio_count;                         /* DIO counter */
25     atomic_t i_writecount;                        /* writers counter */
26     #if defined(CONFIG_IMA) || defined(CONFIG_FILE_LOCKING) /*IMA*/
27         atomic_t i_readcount; /* struct files open RO */
28     #endif
29     union {
30         const struct file_operations *i_fop /*file operations*/
31         void (*free_inode)(struct inode *); /* free function */
32     };

```

Листинг 6 – Структура inode, часть 4

```
1      struct file_lock_context *i_flctx;          /* for fcntl */
2      struct address_space i_data;               /* mapping for device */
3      struct list_head i_devices; /* list of block devices */
4      union {
5          struct pipe_inode_info *i_pipe; /* pipe information */
6          struct cdev *i_cdev;             /*char devices */
7          char *i_link;                    /* symlink */
8          unsigned i_dir_seq;              /* open-coded seqcounter */
9      };
10
11     __u32 i_generation;                    /* inode version number */
12
13 #ifdef CONFIG_FSNOTIFY
14     __u32 i_fsnotify_mask; /* all events inode cares about */
15     struct fsnotify_mark_connector __rcu *i_fsnotify_marks;
16 #endif
17
18 #ifdef CONFIG_FS_ENCRYPTION
19     struct fscrypt_info *i_crypt_info;
20 #endif
21
22 #ifdef CONFIG_FS_VERITY
23     struct fsverity_info *i_verity_info;
24 #endif
25
26     void *i_private; /* fs or device private pointer */
27 } __randomize_layout;
```

Для предотвращения операции 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

```
1 struct file_operations {
2     // ...
3     ssize_t (*read) (struct file *, char __user *,
4         size_t, loff_t *);
5     ssize_t (*write) (struct file *, const char __user *,
6         size_t, loff_t *);
7     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
8     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
9     int (*open) (struct inode *, struct file *);
10    // ...
11 } __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

```
1 struct inode_operations {
2     // ...
3     int (*setattr) (struct user_namespace *, struct dentry *,
4                     struct iattr *);
5     // ...
6 } ____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_for` 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

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

Листинг 10 – Структура dentry, часть 2

```
1 struct super_block *d_sb; /* The root of the dentry tree */
2 unsigned long d_time; /* revalidate time */
3 void *d_fsdata; /* fs-specific data */
4
5 union {
6     struct list_head d_lru; /* LRU list */
7     wait_queue_head_t *d_wait; /* in-lookup ones only */
8 };
9 struct list_head d_child; /* child of parent list */
10 struct list_head d_subdirs; /* our children */
11
12 union {
13     struct hlist_node d_alias; /* inode alias list */
14     struct hlist_bl_node d_in_lookup_hash; /* in-lookup */
15     struct rcu_head d_rcu; /* RCU locking */
16 } d_u;
17 } __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 производства Intel данный бит по умолчанию установлен, поэтому прежде чем заменять значения полей `i_or` и `i_for` 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 текстового 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_for` у `inode`, относящегося к файлу. Для предотвращения изменения доступа к файлу, а также владельца файла и группы пользователей, необходимо пере-

определить значение 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 ассемблерных вставок для возможности изменить необходимые поля у `inode`. При выгрузке разработанного модуля ядра значения изменённых полей должны возвращаться к исходным.

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).