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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Загружаемый модуль ядра, позволяющий скрывать файлы или запрещать их изменение, чтение и удаление»

Студент ИУ7-74Б _____ Д.А. Татаринова
(группа) (Подпись, дата) (И.О. Фамилия)

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

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Обеспечение безопасного доступа к файлам в операционной системе Linux является актуальной задачей. При работе с файлами в Linux необходимо обеспечивать конфиденциальность данных и предоставлять защиту от вредоносных действий.

Данная курсовая работа посвящена разработке модуля, позволяющего ограничивать доступ к определенным файлам.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать загружаемый модуль ядра для ОС Linux, позволяющий скрывать файлы или запрещать их изменение, чтение и удаление. Предусмотреть возможность ввода пароля для отображения файлов или разрешения операций над ними. Предоставить пользователю возможность задавать список таких файлов.

Для решения поставленной задачи необходимо:

— проанализировать возможности перехвата функций ядра Linux;

— выбрать системные вызовы, которые необходимо перехватить;

— разработать алгоритм перехвата, алгоритмы hook–функций и структуру программного обеспечения;

— реализовать программное обеспечение;

— исследовать работу ПО.

1.2 Способы перехвата функций ядра

1.2.1 ftrace

ftrace предоставляет возможности для трассировки функций. С его помощью можно отслеживать контекстные переключения, измерять время обработки прерываний, высчитывать время на активизацию заданий с высоким приоритетом и многое другое [3].

Ftrace был разработан Стивеном Ростедтом и добавлен в ядро в 2008 году, начиная с версии 2.6.27. Ftrace — фреймворк, предоставляющий отладочный

кольцевой буфер для записи данных. Собирают эти данные встроенные в ядро программы–трассировщики [3].

Работает ftrace на базе файловой системы debugfs, которая в большинстве современных дистрибутивов Linux смонтирована по умолчанию.

Каждую перехватываемую функцию можно описать следующей структурой:

Листинг 1.1 — Структура ftrace_hook

```
1 struct ftrace_hook {
2     const char *name;
3     void *function;
4     void *original;
5
6     unsigned long address;
7     struct ftrace_ops ops;
8 };
```

Поля структуры:

name — имя перехватываемой функции;

function — адрес функции–обертки, которая будет вызываться вместо перехваченной функции;

original — указатель на место, куда следует записать адрес перехватываемой функции, заполняется при установке;

address — адрес перехватываемой функции, заполняется при установке;

ops — служебная информация ftrace.

Листинг 1.2 — Пример заполнения структуры ftrace_hook

```
1 #define HOOK(_name, _function, _original)       \
2 {                                                 \
3     .name = (_name),                              \
4     .function = (_function),                      \
5     .original = (_original),                      \
```

```
 6 }
 7
 8 static struct ftrace_hook hooked_functions[] = {
 9     HOOK("sys_clone",  fh_sys_clone,  &real_sys_clone),
10     HOOK("sys_execve",  fh_sys_execve,  &real_sys_execve),
11 };
```

1.2.2 kprobes

Kprobes — специализированное API, в первую очередь предназначенное для отладки и трассирования ядра. Этот интерфейс позволяет устанавливать пред- и постобработчики для любой инструкции в ядре, а также обработчики на вход и возврат из функции. Обработчики получают доступ к регистрам и могут их изменять.

Kprobes реализуются с помощью точек останова (инструкции int3), внедряемых в исполнимый код ядра, что позволяет устанавливать kprobes в любом месте любой функции, если оно известно. Аналогично, kretprobes реализуются через подмену адреса возврата на стеке и позволяют перехватить возврат из любой функции.

При использовании kprobes для получения аргументов функции или значений локальных переменных надо знать, в каких регистрах или где на стеке они лежат, и самостоятельно их оттуда извлекать. Для решения данной проблемы существует jprobes — надстройка над kprobes, самостоятельно извлекающая аргументы функции из регистров или стека и вызывающая обработчик, который должен иметь ту же сигнатуру, что и перехватываемая функция. Однако jprobes объявлен устаревшим и удален из современных ядер.

1.2.3 Linux Security API

Linux Security API — интерфейс, созданный для перехвата функций ядра. В критических местах кода ядра расположены вызовы security–функций, которые в свою очередь вызывают коллбеки, установленные security–модулем. Security–модуль может анализировать контекст операции и принимать решение о ее разрешении или запрете.

Для Linux Security API характерны следующие ограничения:

— security–модули не могут быть загружены динамически, они являются частью ядра и требуют его перекомпиляции;

— в системе может быть только один security–модуль.

Таким образом, для использования Security API необходимо поставлять собственную сборку ядра, а также интегрировать дополнительный модуль с SELinux или AppArmor, которые используются популярными дистрибутивами.

1.2.4 Модификация таблицы системных вызовов

В ядре Linux все обработчики системных вызовов хранятся в таблице sys_call_table [1]. Подмена значений в этой таблице приводит к смене поведения всей системы. Таким образом, сохранив старое значения обработчика и подставив в таблицу собственный обработчик, можно перехватить системный вызов.

Однако данный подход обладает следующими недостатками:

— Техническая сложность реализации, заключающаяся в необходимости обхода защиты от модификации таблицы, атомарное и безопасное выполнение замены.

— Невозможность перехвата некоторых обработчиков. В ядрах до версии 4.16 обработка системных вызовов для архитектуры x86_64 содержала целый ряд оптимизаций. Некоторые из них требовали того, что обработчик системного вызова являлся специальным переходником, реализованным на ассемблере. Соответственно, подобные обработчики порой сложно, а иногда и вовсе невозможно заменить на собственные, написанные на Си [2].

1.2.5 Сплайсинг

Сплайсинг заключается в замене инструкций в начале функции на безусловный переход, ведущий в обработчик. Оригинальные инструкции переносятся в другое место и исполняются перед переходом обратно в перехваченную функцию. Именно таким образом реализуется jump–оптимизация для kprobes. Используя сплайсинг, можно добиться тех же результатов, но без дополнительных расходов на kprobes и с полным контролем ситуации.

Сложность использования сплайсинга заключается в необходимости синхронизации установки и снятия перехвата, обхода защиты от модификации регионов памяти с кодом, инвалидации кешей процессора после замены инструкций, дизассемблировании заменяемых инструкций и проверки на отсутствие переходов внутрь заменяемого кода.

1.2.6 Сравнительный анализ способов перехвата

Результаты сравнения различных способов перехвата функций ядра представлены в таблице 1.1

Таблица 1.1 — Результаты сравнения

Способ перехвата	Необходимость перекомпиляции ядра	Возможность перехвата любых обработчиков	Доступ к аргументам функции через переменные	Необходимость обхода защиты от модификации регионов памяти
Linux Security API	да	нет	да	нет
Модификация таблиц системных вызовов	нет	нет	да	да
kprobes	нет	да	нет	нет
Сплайсинг	нет	да	да	да
ftrace	нет	да	да	нет

1.3 Перехватываемые функции ядра

1.3.1 Функция getdents64

Для сокрытия файла необходимо перехватить функцию ядра getdents64, так как она возвращает записи каталога.

Листинг 1.3 — Функции getdents64

```
1 #include <fcntl.h>
2
3 int getdents64(unsigned int fd, struct linux_dirent64 *dirp, unsigned int
      count);
```

Системный вызов getdents64 читает несколько структур linux_dirent64 из каталога, на который указывает открытый файловый дескриптор fd, в буфер, указанный в dirp. В аргументе count задается размер этого буфера.

Структура linux_dirent64 определена следующим образом:

Листинг 1.4 — Структура linux_dirent64

```
1 struct linux_dirent64 {
2         ino64_t      d_ino;
3         off64_t      d_off;
4         unsigned short  d_reclen;
5         unsigned char   d_type;
6         char     d_name[];
7 };
```

В d_ino указан номер inode. В d_off задается расстояние от начала каталога до начала следующей linux_dirent64. В d_reclen указывается размер данного linux_dirent64. В d_name задается имя файла, в d_type — тип файла.

Таким образом, перехват системного вызова getdents64 позволяет удалить запись из списка записей каталога. Перехват вызова осуществляется при помощи создания указателя на системный вызов getdents64.

Листинг 1.5 — Указатель на getdents64

```
1 static asmlinkage long (*real_sys_getdents64)(const struct pt_regs *);
```

1.3.2 Функция unlink

Удаление записей из каталога производится с помощью функции unlink.

Листинг 1.6 — Функция unlink

```
1 #include <unistd.h>
2
3 int unlink(const char *pathname);
```

Эта функция удаляет запись из файла каталога и уменьшает значение счетчика ссылок на файл pathname. Если на файл указывает несколько ссылок, то его содержимое будет через них по–прежнему доступно.

Таким образом, для того, чтобы запретить удаление файла, необходимо перехватить системный вызов unlink. Перехват вызова осуществляется при помощи создания указателя на системный вызов.

Листинг 1.7 — Указатель на unlink

```
1 static asmlinkage long (*real_sys_unlinkat) (struct pt_regs *regs);
```

1.3.3 Функции open, write, read

Запись в файл осуществляется при помощи системного вызова write, чтение из файла — read.

Листинг 1.8 — Функции write и read

```
1 #include <fcntl.h>
2
3 ssize_t write(int fd, const void *buf, size_t size);
4 ssize_t read(int fd, void *buf, size_t count);
```

Функции write и read работают не с именем файла, а с файловым дескриптором. Однако для того, чтобы определить, можно ли выполнить операцию с файлом, необходимо знать его имя, так как файлы, для которых необходимо запретить те или иные операции, хранятся в виде списка имен этих файлов. Тогда следует перехватить функцию open, которая позволяет получить имя файла.

Системный вызов open осуществляет открытие файла.

Листинг 1.9 — Функция open

```
1 #include <fcntl.h>
2
3 int open(int dirfd, const char *pathname, int flags);
4 int open(int dirfd, const char *pathname, int flags, mode_t mode);
```

При перехвате open, если имя файла есть в списке, то необходимо сохранить идентификатор процесса, открывающего файл. Затем при перехвате write и read ориентироваться не на имя файла, а на идентификатор процесса и файловый дескриптор. Так можно будет однозначно определить, необходимо ли запретить чтение из файла или запись в него.

Таким образом, для того, чтобы запретить чтение из файла и запись в него, необходимо перехватить системные вызовы open, write и read. Перехват осуществляется при помощи создания указателей на системные вызовы.

Листинг 1.10 — Указатели на open, write и read.

```
1 static asmlinkage long (*real_sys_open)(struct pt_regs *regs);
2 static asmlinkage long (*real_sys_write)(struct pt_regs *regs);
3 static asmlinkage long (*real_sys_read)(struct pt_regs *regs);
```

Вывод

В результате сравнительного анализа выбран способ перехвата функций ядра — ftrace, так как он позволяет перехватывать любые функции ядра и не требует его перекомпиляции. Для сокрытия файла необходимо перехватить функцию getdents64, для запрета чтения из файла и записи в файл — функции open, read и write, удаления — unlink.

2 Конструкторский раздел

2.1 IDEF0

На рисунке 2.1 приведена диаграмма состояний IDEF0 нулевого уровня, а на рисунке 2.2 — диаграмма состояний IDEF0 первого уровня.

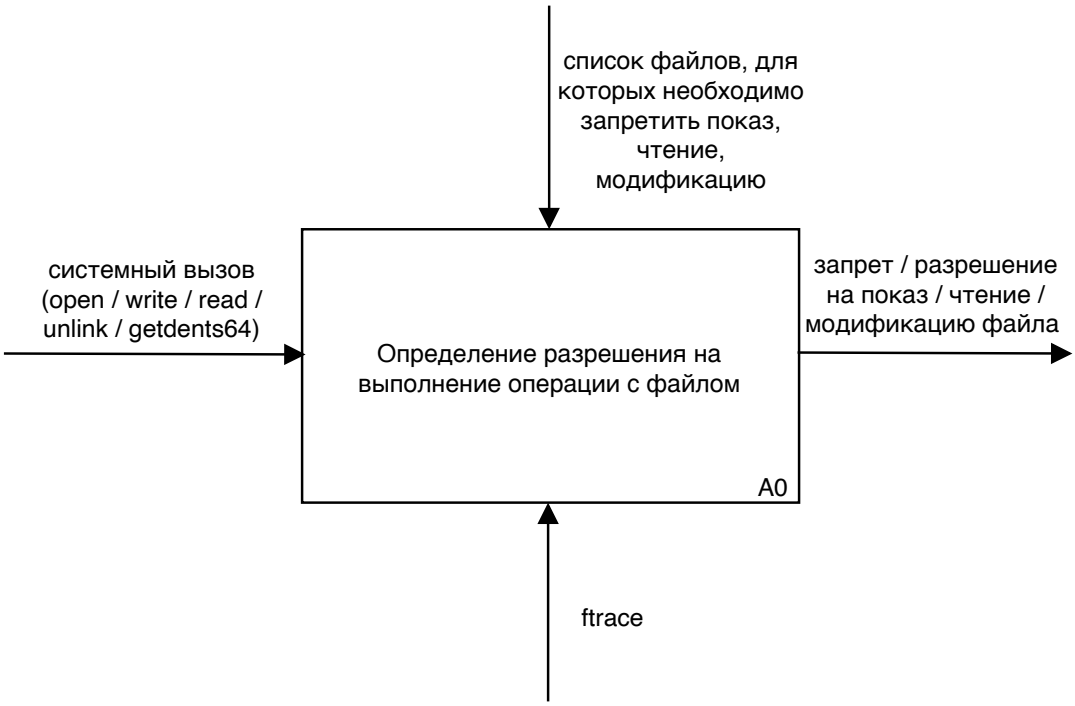

Рисунок 2.1 — Диаграмма состояний IDEF0 нулевого уровня

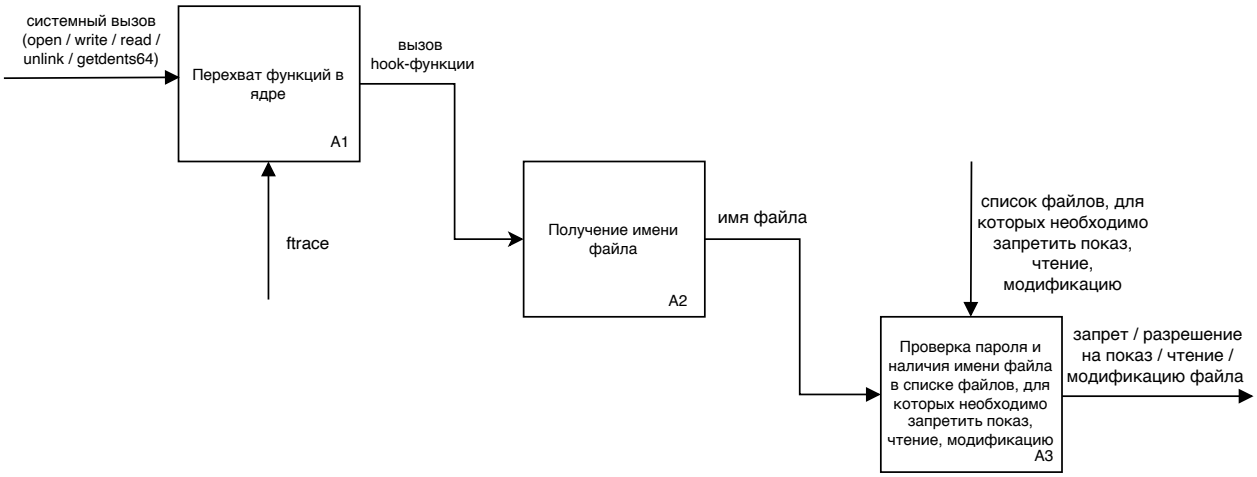

Рисунок 2.2 — Диаграмма состояний IDEF0 первого уровня

2.2 Алгоритм проверки необходимости сокрытия файла

На рисунке 2.3 приведена схема алгоритма проверки необходимости сокрытия файла.

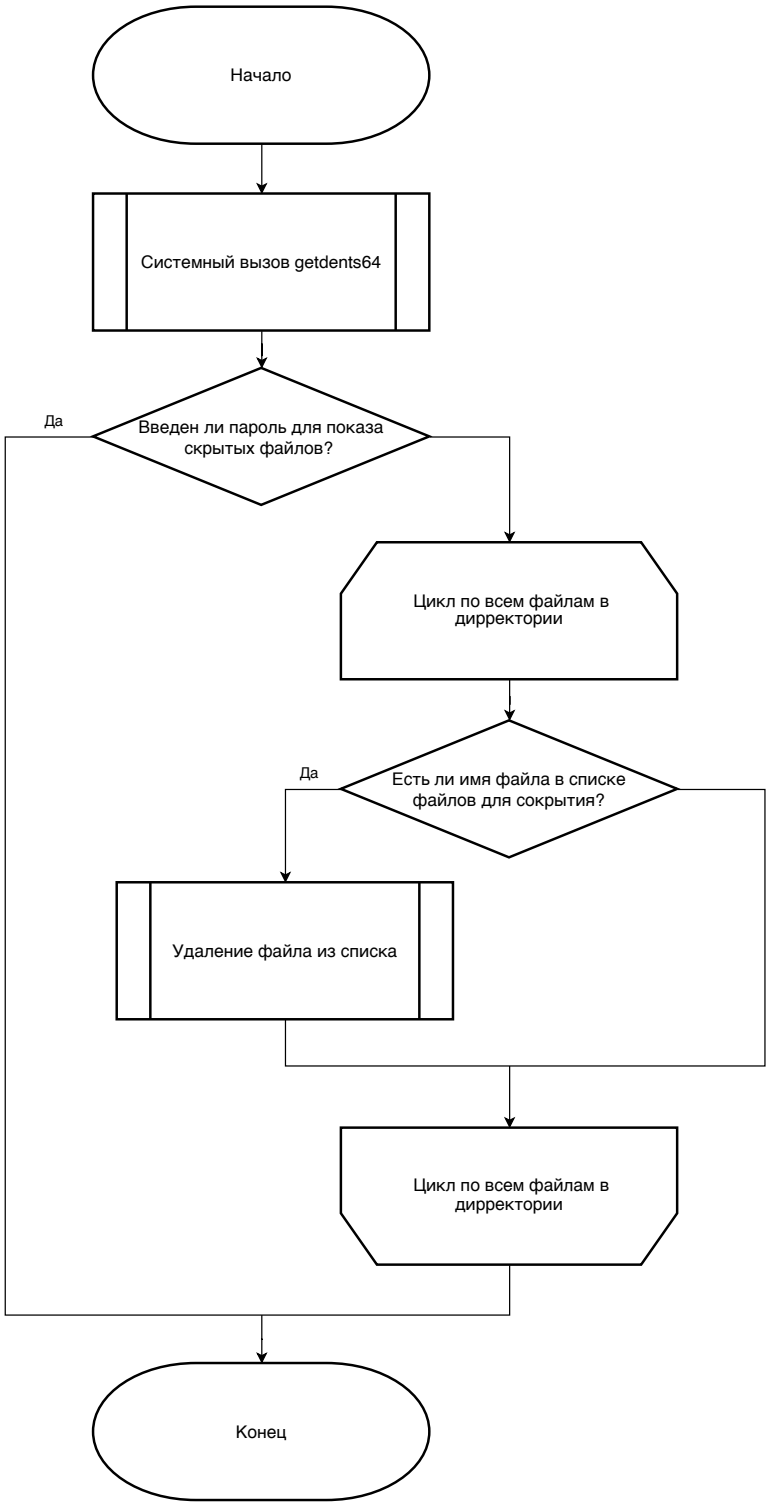

Рисунок 2.3 — Алгоритм проверки необходимости сокрытия файла

2.3 Алгоритм проверки разрешения на удаление файла

На рисунке 2.4 приведена схема алгоритма проверки разрешения на удаление файла.

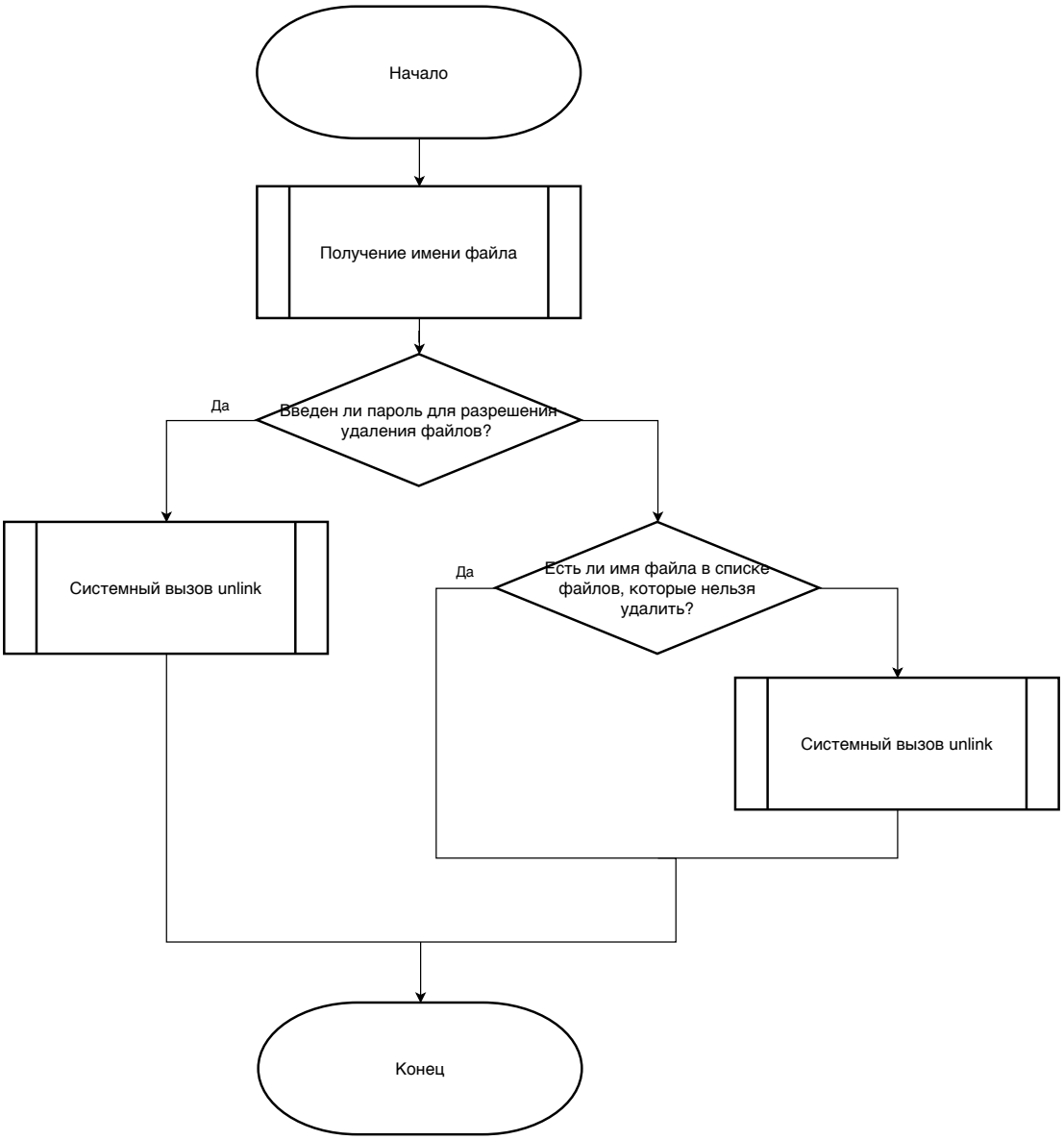

Рисунок 2.4 — Алгоритм проверки разрешения на удаления файла

2.4 Алгоритм проверки разрешения на запись в файл

На рисунке 2.5 приведена схема алгоритма проверки разрешения на запись в файл.

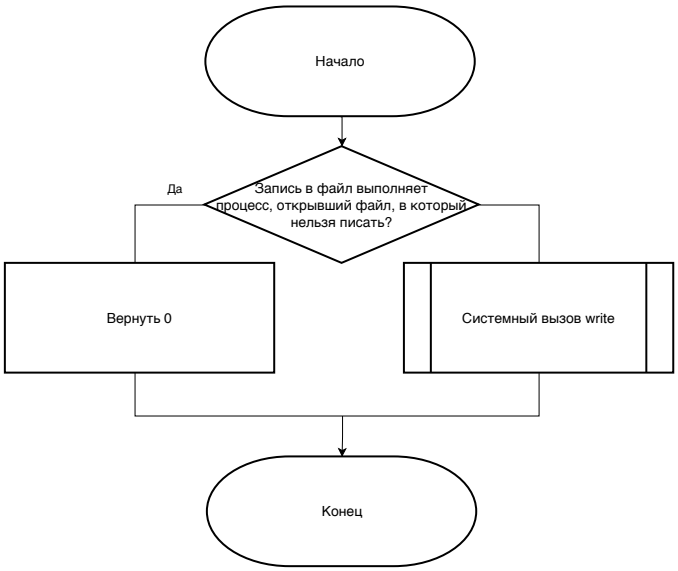

Рисунок 2.5 — Алгоритм проверки разрешения на запись в файл

2.5 Алгоритм проверки разрешения на чтение из файла

На рисунке 2.6 приведена схема алгоритма проверки разрешения на чтение из файла.

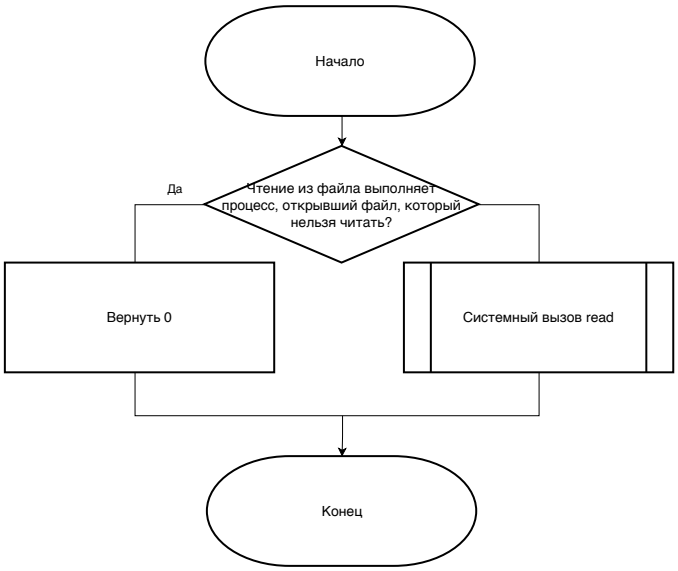

Рисунок 2.6 — Алгоритм проверки разрешения на чтение из файла

2.6 Структура программного обеспечения

На рисунке 2.7 представлена структура программного обеспечения.

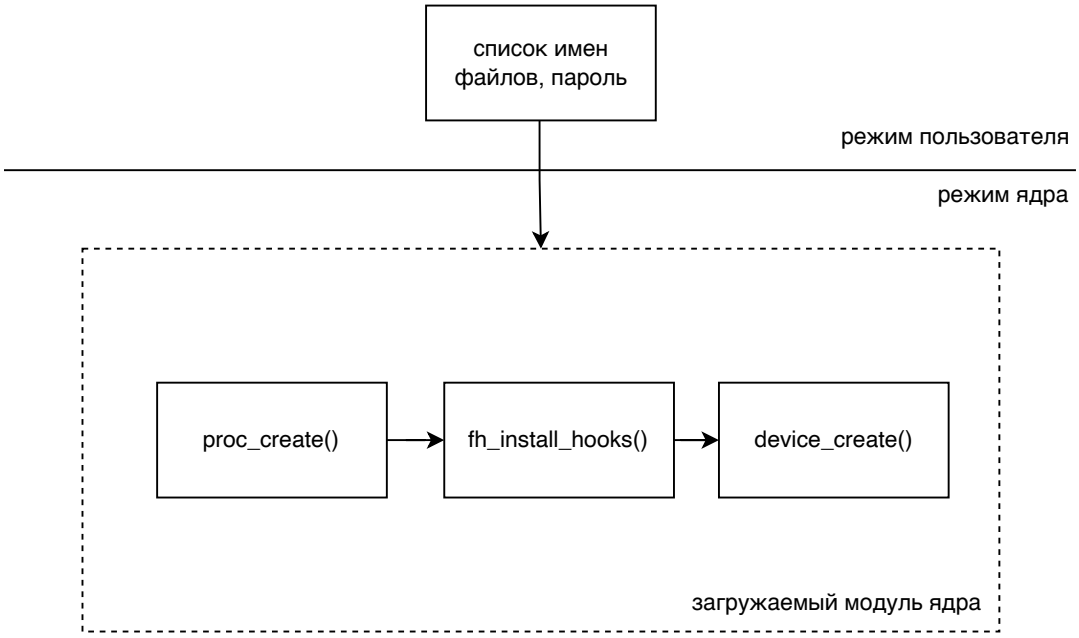

Рисунок 2.7 — Структура программного обеспечения

3 Технологический раздел

3.1 Выбор языка и среды программирования

В качестве языка программирования был выбран язык Си. Для сборки модуля использовалась утилита make. В качестве среды программирования был выбран VSCode.

3.2 Реализация алгоритма проверки необходимости сокрытия файла

В листинге 3.1 приведена реализация алгоритма проверки необходимости сокрытия файл.

Листинг 3.1 — Реализация алгоритма проверки необходимости сокрытия файл

```
1  static asmlinkage int fh_sys_getdents64(const struct pt_regs *regs)
2  {
3      struct linux_dirent64 __user *dirent = (struct linux_dirent64
           *)regs->si;
4      struct linux_dirent64 *previous_dir, *current_dir, *dirent_ker = NULL;
5      unsigned long offset = 0;
6      int ret = real_sys_getdents64(regs);
7
8      dirent_ker = kzalloc(ret, GFP_KERNEL);
9
10     if ((ret <= 0) || (dirent_ker == NULL))
11     {
12         return ret;
13     }
14
15     copy_from_user(dirent_ker, dirent, ret);
16
17     while (offset < ret)
18     {
19         current_dir = (void *)dirent_ker + offset;
20
21         if (check_fs_hidelist(current_dir->d_name))
```

```
22          {
23              if (current_dir == dirent_ker)
24              {
25                  ret -= current_dir->d_reclen;
26                  memmove(current_dir, (void *)current_dir +
                        current_dir->d_reclen, ret);
27                  continue;
28              }
29
30              previous_dir->d_reclen += current_dir->d_reclen;
31          }
32          else
33          {
34              previous_dir = current_dir;
35          }
36
37          offset += current_dir->d_reclen;
38      }
39
40      copy_to_user(dirent, dirent_ker, ret);
41
42      kfree(dirent_ker);
43      return ret;
44  }
```

3.3 Реализация алгоритма проверки разрешения на удаление файла

Реализация алгоритма проверки разрешения на удаление файла представлена в листинге 3.2.

Листинг 3.2 — Реализация алгоритма проверки разрешения на удаление файла

```
1  static asmlinkage long fh_sys_unlink(struct pt_regs *regs)
2  {
3      long ret=0;
4      char *kernel_filename = get_filename((void*) regs->si);
5
6      if (check_fs_blocklist(kernel_filename))
```

```
 7       {
 8           ret=0;
 9           kfree(kernel_filename);
10           return ret;
11       }
12
13       kfree(kernel_filename);
14       ret = real_sys_unlink(regs);
15
16       return ret;
17   }
```

3.4 Реализация алгоритма проверки разрешения на чтение из файла

Реализация алгоритма проверки разрешения на чтение из файла представлена в листинге 3.3.

Листинг 3.3 — Реализация алгоритма проверки разрешения на чтение из файла

```
 1  static asmlinkage long fh_sys_read(struct pt_regs *regs)
 2  {
 3      long ret = 0;
 4      struct task_struct *task = current;
 5      if (task->pid == target_pid && regs->si == target_fd)
 6      {
 7          return 0;
 8      }
 9      ret = real_sys_read(regs);
10      return ret;
11  }
```

3.5 Реализация алгоритма проверки разрешения на запись в файл

Реализация алгоритма проверки разрешения на запись в файл представлена в листинге 3.4.

Листинг 3.4 — Реализация алгоритма проверки разрешения на запись в файл

```
1  static asmlinkage long fh_sys_write(struct pt_regs *regs)
2  {
3      long ret = 0;
4      struct task_struct *task = current;
5      if (task->pid == target_pid && regs->si == target_fd)
6      {
7          return 0;
8      }
9      ret = real_sys_write(regs);
10     return ret;
11 }
```

3.6 Инициализация полей структуры ftrace_hook

Инициализация полей структуры ftrace_hook представлена в листинге 3.5.

Листинг 3.5 — Инициализация полей структуры ftrace_hook

```
1  static struct ftrace_hook demo_hooks[] = {
2      HOOK("sys_write", fh_sys_write, &real_sys_write),
3      HOOK("sys_read", fh_sys_read, &real_sys_read),
4      HOOK("sys_open", fh_sys_open, &real_sys_open),
5      HOOK("sys_unlink", fh_sys_unlink, &real_sys_unlink),
6      HOOK("sys_getdents64", fh_sys_getdents64, &real_sys_getdents64)
7  };
```

3.7 Makefile

В листинге 3.6 представлен Makefile.

Листинг 3.6 — Makefile

```
1  CONFIG_MODULE_SIG=n
2  PWD := $(shell pwd)
3  CC := gcc
4  KERNEL_PATH ?= /lib/modules/$(shell uname -r)/build
5  ccflags-y    += -Wall -Wdeclaration-after-statement
6
7  obj-m += my_module.o
8  casperfs-objs := main.o hooked.o
9
10 all:
11 make -C $(KERNEL_PATH) M=$(PWD) modules
12
13 clean:
14 make -C $(KERNEL_PATH) M=$(PWD) clean
```

4 Исследовательский раздел

Программное обеспечение было реализовано на дистрибутиве Ubuntu 20.04, ядро версии 5.19.0.

4.1 Пример работы разработанного программного обеспечения

Пусть содержимое рассматриваемой директории имеет вид, изображенный на рисунке 4.1.

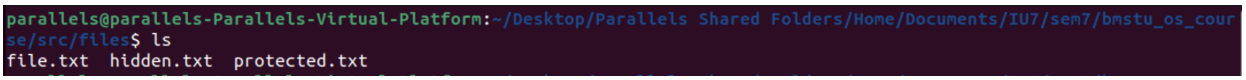

Рисунок 4.1 — Содержимое папки до загрузки модуля

Файлы, содержащие списки контроля доступа, находятся в директории /proc. На рисунке 4.2 изображен пример формирования таких списков.

Рисунок 4.2 — Создание файлов, содержащих списки контроля доступа

Файл hidden содержит имена файлов, которые необходимо скрыть полностью, файл protected — имена файлов, которые нельзя открывать, изменять, удалять.

После загрузки модуля содержимое рассматриваемой директории выглядит следующим образом (рисунок 4.3).

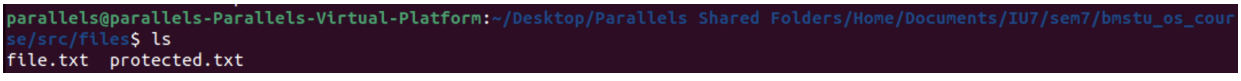

Рисунок 4.3 — Содержимое папки после загрузки модуля

Результат выполнения команды ls не содержит файл hidden.txt.

После ввода пароля (рисунок 4.4) файл hidden.txt перестает быть скрытым (рисунок 4.5).

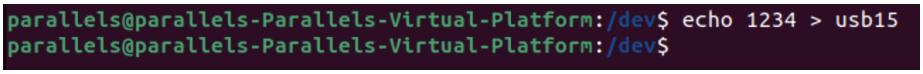

Рисунок 4.4 — Ввод пароля

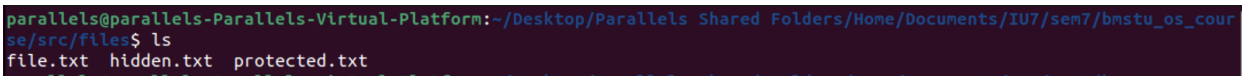

Рисунок 4.5 — Результат работы команды ls после ввода пароля

При попытке удалить файл protected.txt (рисунок 4.6) или вывести его содержимое с помощью cat ничего не происходит.

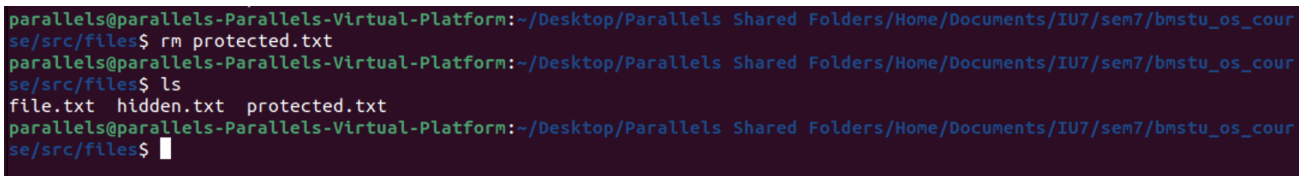

Рисунок 4.6 — Удаление файла protected.txt

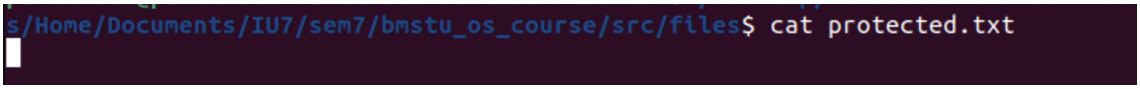

Рисунок 4.7 — Вывод содержимого файла protected.txt

После ввода пароля (рисунок 4.8) операции над файлом protected.txt становятся возможными (рисунок 4.9).

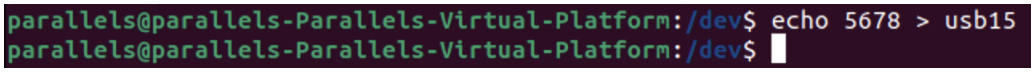

Рисунок 4.8 — Ввод пароля

Рисунок 4.9 — Вывод содержимого файла protected.txt после ввода пароля

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был определен способ перехвата системных вызовов — путем регистрации функций перехвата с использованием ftrace, так как он позволяет перехватывать любые функции ядра и не требует его перекомпиляции.

Для сокрытия файла была перехвачена функция getdents64, для запрета чтения из файла и записи в файл — функции open, read и write, удаления — unlink.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Код ядра Linux [Электронный ресурс]. — Режим доступа: `https://elixir.bootlin.com/linux/latest/source` (дата обращения: 10.12.2023).

2. Встраивание в ядро Linux: перехват функций [Электронный ресурс]. — Режим доступа: `https://habr.com/ru/companies/securitycode/articles/237089/` (дата обращения: 12.12.2023).

3. Трассировка ядра с ftrace [Электронный ресурс]. — Режим доступа: `https://habr.com/ru/companies/selectel/articles/280322/` (дата обращения: 12.12.2023).

4. Стивенс Раго. UNIX. Профессиональное программирование. — Питер, 2018. — 944 с.

ПРИЛОЖЕНИЕ А

Листинг А.1 — Файл main.c

```
1  #include <linux/module.h>
2  #include <linux/kallsyms.h>
3  #include <linux/skbuff.h>
4  #include <linux/init.h>
5  #include <linux/fs.h>
6  #include <linux/device.h>
7  #include <linux/cdev.h>
8  #include <linux/proc_fs.h>
9  #include <linux/string.h>
10
11 #include "hooked.h"
12
13 MODULE_DESCRIPTION("os_course");
14 MODULE_AUTHOR("Darya Tatarinova");
15 MODULE_LICENSE("GPL");
16
17 #define PROC_FILE_NAME_HIDDEN "hidden"
18 #define PROC_FILE_NAME_PROTECTED "protected"
19
20 static char *buffer[MAX_BUF_SIZE];
21 char tmp_buffer[MAX_BUF_SIZE];
22 char hidden_files[100][9];
23 int hidden_index = 0;
24 char protected_files[100][9];
25 int protected_index = 0;
26
27 static ssize_t my_proc_write(struct file *file, const char __user *buf,
       size_t len, loff_t *ppos)
28 {
29     DMSG("my_proc_write called");
30
31     if (len > MAX_BUF_SIZE - write_index + 1)
32     {
33         DMSG("buffer overflow");
34         return -ENOSPC;
35     }
36
```

```
37      if (copy_from_user(&buffer[write_index], buf, len) != 0)
38      {
39          DMSG("copy_from_user fail");
40          return -EFAULT;
41      }
42
43      write_index += len;
44      buffer[write_index - 1] = '\0';
45
46      if (strcmp(file->f_path.dentry->d_iname, PROC_FILE_NAME_HIDDEN) == 0)
47      {
48          snprintf(hidden_files[hidden_index], len, "%s", &buffer[write_index
                - len]);
49          hidden_index++;
50          DMSG("file written to hidden %s", hidden_files[hidden_index - 1]);
51      }
52      if (strcmp(file->f_path.dentry->d_iname, PROC_FILE_NAME_PROTECTED) == 0)
53      {
54          snprintf(protected_files[protected_index], len, "%s",
                &buffer[write_index - len]);
55          protected_index++;
56          DMSG("file written to protected %s",
                protected_files[protected_index - 1]);
57      }
58      else
59      {
60          DMSG("Unknown file->f_path.dentry->d_iname");
61      }
62      return len;
63  }
64
65  static ssize_t my_proc_read(struct file *file, char __user *buf, size_t
        len, loff_t *f_pos)
66  {
67      DMSG("my_proc_read called.\n");
68
69      if (*f_pos > 0 || write_index == 0)
70      return 0;
71
72      if (read_index >= write_index)
73      read_index = 0;
```

```
74
75      int read_len = snprintf(tmp_buffer, MAX_BUF_SIZE, "%s\n",
            &buffer[read_index]);
76      if (copy_to_user(buf, tmp_buffer, read_len) != 0)
77      {
78          DMSG("copy_to_user error.\n");
79          return -EFAULT;
80      }
81
82      read_index += read_len;
83      *f_pos += read_len;
84
85      return read_len;
86  }
87
88  static const struct proc_ops fops =
89  {
90      proc_read: my_proc_read,
91      proc_write: my_proc_write
92  };
93
94
95  static int fh_init(void)
96  {
97      proc_file_hidden = proc_create(PROC_FILE_NAME_HIDDEN, S_IRUGO |
            S_IWUGO, NULL, &fops);
98      if (!proc_file_hidden)
99      {
100         DMSG("call proc_create_data() fail");
101         return -ENOMEM;
102     }
103     proc_file_protected = proc_create(PROC_FILE_NAME_PROTECTED, S_IRUGO |
            S_IWUGO, NULL, &fops);
104     if (!proc_file_protected)
105     {
106         remove_proc_entry(PROC_FILE_NAME_HIDDEN, NULL);
107         DMSG("call proc_create_data() fail");
108         return -ENOMEM;
109     }
110     DMSG("proc file created");
111
```

```
112     struct device *fake_device;
113     int error = 0;
114     dev_t devt = 0;
115
116     error = start_hook_resources();
117     if (error)
118     {
119         remove_proc_entry(PROC_FILE_NAME_HIDDEN, NULL);
120         remove_proc_entry(PROC_FILE_NAME_PROTECTED, NULL);
121         DMSG("Problem in hook functions");
122         return error;
123     }
124
125     error = alloc_chrdev_region(&devt, 0, 1, "usb15");
126
127     if (error < 0)
128     {
129         remove_proc_entry(PROC_FILE_NAME_HIDDEN, NULL);
130         remove_proc_entry(PROC_FILE_NAME_PROTECTED, NULL);
131         return error;
132     }
133
134     major = MAJOR(devt);
135     minor = MINOR(devt);
136
137     fake_class = class_create(THIS_MODULE, "custom_char_class");
138
139     if (IS_ERR(fake_class)) {
140         remove_proc_entry(PROC_FILE_NAME_HIDDEN, NULL);
141         remove_proc_entry(PROC_FILE_NAME_PROTECTED, NULL);
142         unregister_chrdev_region(MKDEV(major, minor), 1);
143         return PTR_ERR(fake_class);
144     }
145
146     cdev_init(&fake_cdev, &fake_fops);
147     fake_cdev.owner = THIS_MODULE;
148     cdev_add(&fake_cdev, devt, 1);
149
150     fake_device = device_create(fake_class,
151             NULL,   /* no parent device */
152             devt,    /* associated dev_t */
```

```
153             NULL,   /* no additional data */
154             "usb15");  /* device name */
155
156     if (IS_ERR(fake_device))
157     {
158         remove_proc_entry(PROC_FILE_NAME_HIDDEN, NULL);
159         remove_proc_entry(PROC_FILE_NAME_PROTECTED, NULL);
160         class_destroy(fake_class);
161         unregister_chrdev_region(devt, 1);
162         return -1;
163     }
164
165     return 0;
166 }
167
168
169 static void fh_exit(void)
170 {
171     if (proc_file_hidden)
172     {
173         remove_proc_entry(PROC_FILE_NAME_HIDDEN, NULL);
174         DMSG("proc file removed (hidden)");
175     }
176
177     if (proc_file_protected)
178     {
179         remove_proc_entry(PROC_FILE_NAME_PROTECTED, NULL);
180         DMSG("proc file removed (prtotected)");
181     }
182
183     fh_remove_hooks(demo_hooks, ARRAY_SIZE(demo_hooks));
184     unregister_chrdev_region(MKDEV(major, 0), 1);
185     device_destroy(fake_class, MKDEV(major, 0));
186     cdev_del(&fake_cdev);
187     class_destroy(fake_class);
188 }
189
190 module_init(fh_init);
191 module_exit(fh_exit);
```

Листинг А.2 — Файл hook.h

```
1  #include <linux/ftrace.h>
2  #include <linux/kallsyms.h>
3  #include <linux/syscalls.h>
4  #include <linux/kernel.h>
5  #include <linux/version.h>
6  #include <linux/kprobes.h>
7  #include <linux/delay.h>
8  #include <linux/kthread.h>
9  #include <linux/kernel.h>
10 #include <asm/signal.h>
11 #include <linux/delay.h>
12 #include <linux/fcntl.h>
13 #include <linux/types.h>
14 #include <linux/dirent.h>
15 #include <linux/device.h>
16 #include <linux/cdev.h>
17 #include <linux/module.h>
18 #include <linux/init.h>
19 #include <linux/fs.h>
20 #include <linux/proc_fs.h>
21
22 MODULE_DESCRIPTION("os_course");
23 MODULE_AUTHOR("Darya Tatarinova");
24
25 #define FILE_NAME (strrchr(__FILE__, '/') ? strrchr(__FILE__, '/') + 1 :
      __FILE__)
26 #define DMSG(msg_fmt, msg_args...) \
27 printk(KERN_INFO "OS: %s(%04u): " msg_fmt "\n", FILE_NAME, __LINE__,
      ##msg_args)
28
29 #define MAX_BUF_SIZE 1000
30
31 static struct proc_dir_entry *proc_file_hidden;
32 static struct proc_dir_entry *proc_file_protected;
33
34 extern char hidden_files[100][9];
35 extern int hidden_index;
36 extern char protected_files[100][9];
37 extern int protected_index;
```

```
38
39 static int read_index = 0;
40 static int write_index = 0;
41
42 static unsigned int major;
43 static unsigned int minor;
44 static struct class *fake_class;
45 static struct cdev fake_cdev;
46
47 static short fs_hidden = 1;
48 static short fs_protect = 1;
49
50 ssize_t fake_write(struct file * filp, const char __user * buf, size_t
       count,loff_t * offset);
51
52
53 static struct file_operations fake_fops = {
54     write: fake_write,
55 };
56
57 int check_fs_blocklist(char *input);
58 int check_fs_hidelist(char *input);
59
60 static unsigned int target_fd = 0;
61 static unsigned int target_pid = 0;
62
63 static unsigned long lookup_name(const char *name)
64 {
65     struct kprobe kp = {
66         .symbol_name = name
67     };
68     unsigned long retval;
69
70     if (register_kprobe(&kp) < 0)
71     {
72         return 0;
73     }
74     retval = (unsigned long) kp.addr;
75     unregister_kprobe(&kp);
76     return retval;
77 }
```

```
78
79  struct ftrace_hook {
80      const char *name;
81      void *function;
82      void *original;
83
84      unsigned long address;
85      struct ftrace_ops ops;
86  };
87
88  static int fh_resolve_hook_address(struct ftrace_hook *hook)
89  {
90      hook->address = lookup_name(hook->name);
91
92      if (!hook->address) {
93          pr_debug("unresolved symbol: %s\n", hook->name);
94          return -ENOENT;
95      }
96
97      *((unsigned long*) hook->original) = hook->address + MCOUNT_INSN_SIZE;
98
99      return 0;
100 }
101
102 static void notrace fh_ftrace_thunk(unsigned long ip, unsigned long
        parent_ip,
103 struct ftrace_ops *ops, struct ftrace_regs *fregs)
104 {
105     struct pt_regs *regs = ftrace_get_regs(fregs);
106     struct ftrace_hook *hook = container_of(ops, struct ftrace_hook, ops);
107
108     regs->ip = (unsigned long)hook->function;
109 }
110
111 int fh_install_hook(struct ftrace_hook *hook);
112 void fh_remove_hook(struct ftrace_hook *hook);
113 int fh_install_hooks(struct ftrace_hook *hooks, size_t count);
114 void fh_remove_hooks(struct ftrace_hook *hooks, size_t count);
115
116 static char *get_filename(const char __user *filename)
117 {
```

```
118     char *kernel_filename=NULL;
119
120     kernel_filename = kmalloc(4096, GFP_KERNEL);
121     if (!kernel_filename)
122     return NULL;
123
124     if (strncpy_from_user(kernel_filename, filename, 4096) < 0) {
125         kfree(kernel_filename);
126         return NULL;
127     }
128
129     return kernel_filename;
130 }
131
132 static asmlinkage long (*real_sys_write)(struct pt_regs *regs);
133 static asmlinkage long fh_sys_write(struct pt_regs *regs)
134 {
135     long ret=0;
136     struct task_struct *task;
137     task = current;
138     if (task->pid == target_pid && regs->di == target_fd)
139     {
140         return 0;
141     }
142     ret = real_sys_write(regs);
143     return ret;
144 }
145
146
147 static asmlinkage long (*real_sys_open)(struct pt_regs *regs);
148 static asmlinkage long fh_sys_open(struct pt_regs *regs)
149 {
150     long ret;
151     char *kernel_filename;
152     struct task_struct *task;
153     task = current;
154     kernel_filename = get_filename((void*) regs->si);
155
156     if (check_fs_blocklist(kernel_filename))
157     {
158         DMSG("our file is opened by process with id: %d\n", task->pid);
```

```
159         DMSG("opened file : %s\n", kernel_filename);
160         kfree(kernel_filename);
161         ret = real_sys_open(regs);
162         DMSG("fd returned is %ld\n", ret);
163         target_fd = ret;
164         target_pid = task->pid;
165         return 0;
166
167     }
168
169     kfree(kernel_filename);
170     ret = real_sys_open(regs);
171
172     return ret;
173 }
174
175
176 static asmlinkage long (*real_sys_unlink) (struct pt_regs *regs);
177 static asmlinkage long fh_sys_unlink(struct pt_regs *regs)
178 {
179     long ret=0;
180     char *kernel_filename = get_filename((void*) regs->si);
181     if (check_fs_blocklist(kernel_filename))
182     {
183         ret=0;
184         kfree(kernel_filename);
185         return ret;
186     }
187     kfree(kernel_filename);
188     ret = real_sys_unlinkat(regs);
189     return ret;
190 }
191
192 static asmlinkage long (*real_sys_getdents64)(const struct pt_regs *);
193 static asmlinkage int fh_sys_getdents64(const struct pt_regs *regs)
194 {
195     struct linux_dirent64 __user *dirent = (struct linux_dirent64
        *)regs->si;
196     struct linux_dirent64 *previous_dir, *current_dir, *dirent_ker = NULL;
197     unsigned long offset = 0;
198     int ret = real_sys_getdents64(regs);
```

```
199
200     dirent_ker = kzalloc(ret, GFP_KERNEL);
201
202     if ((ret <= 0) || (dirent_ker == NULL))
203     {
204         return ret;
205     }
206
207     copy_from_user(dirent_ker, dirent, ret);
208
209     while (offset < ret)
210     {
211         current_dir = (void *)dirent_ker + offset;
212
213         if (check_fs_hidelist(current_dir->d_name))
214         {
215             if (current_dir == dirent_ker )
216             {
217                 ret -= current_dir->d_reclen;
218                 memmove(current_dir, (void *)current_dir +
                        current_dir->d_reclen, ret);
219                 continue;
220             }
221
222             previous_dir->d_reclen += current_dir->d_reclen;
223         }
224         else
225         {
226             previous_dir = current_dir;
227         }
228
229         offset += current_dir->d_reclen;
230     }
231
232     copy_to_user(dirent, dirent_ker, ret);
233
234     kfree(dirent_ker);
235     return ret;
236 }
237
238 #define SYSCALL_NAME(name) ("__x64_" name)
```

```
239
240 #define HOOK(_name, _function, _original)   \
241 {                       \
242     .name = SYSCALL_NAME(_name),    \
243     .function = (_function),    \
244     .original = (_original),    \
245 }
246
247 static struct ftrace_hook demo_hooks[] = {
248     HOOK("sys_write", fh_sys_write, &real_sys_write),
249     HOOK("sys_read", fh_sys_read, &real_sys_read),
250     HOOK("sys_open", fh_sys_open, &real_sys_open),
251     HOOK("sys_unlink", fh_sys_unlink, &real_sys_unlink),
252     HOOK("sys_getdents64", fh_sys_getdents64, &real_sys_getdents64)
253 };
254
255 static int start_hook_resources(void)
256 {
257     int err;
258     err = fh_install_hooks(demo_hooks, ARRAY_SIZE(demo_hooks));
259     if (err)
260     {
261         return err;
262     }
263     return 0;
264 }
```

Листинг А.3 — Файл hook.c

```
1 #include "hooked.h"
2
3 ssize_t fake_write(struct file * filp, const char __user * buf, size_t
      count,
4 loff_t * offset)
5 {
6     char message[128];
7     memset(message, 0, 127);
8
9     if(copy_from_user(message,buf,127) != 0)
10    {
11        return EFAULT;
```

```
12      }
13
14      if(strstr(message, "1234") != NULL)
15      {
16          fs_hidden = fs_hidden ? 0 : 1;
17      }
18
19      if(strstr(message, "5678") != NULL)
20      {
21          fs_protect = fs_protect ? 0 : 1;
22      }
23
24      return count;
25  }
26
27
28  int check_fs_blocklist(char *input)
29  {
30      int i = 0;
31
32      if (fs_protect==0)
33      {
34          return 0;
35      }
36
37      if (strlen(protected_files[0]) <= 2)
38      {
39          return 0;
40      }
41
42      while (i != protected_index)
43      {
44          if(strstr(input, protected_files[i]) != NULL)
45          return 1;
46          i++;
47      }
48
49      return 0;
50  }
51
52  int check_fs_hidelist(char *input)
```

```
53 {
54     int i = 0;
55     if (fs_hidden == 0)
56     {
57         return 0;
58     }
59
60     if (strlen(hidden_files[0]) <= 2)
61     {
62         return 0;
63     }
64
65     while (i != hidden_index)
66     {
67         if(strstr(input, hidden_files[i]) != NULL)
68         return 1;
69         i++;
70     }
71
72     return 0;
73 }
74
75 int fh_install_hook(struct ftrace_hook *hook)
76 {
77     int error;
78
79     error = fh_resolve_hook_address(hook);
80     if (error)
81     {
82         return error;
83     }
84
85     hook->ops.func = fh_ftrace_thunk;
86     hook->ops.flags = FTRACE_OPS_FL_SAVE_REGS
87     | FTRACE_OPS_FL_RECURSION
88     | FTRACE_OPS_FL_IPMODIFY;
89
90     error = ftrace_set_filter_ip(&hook->ops, hook->address, 0, 0);
91     if (error)
92     {
93         DMSG("ftrace_set_filter_ip() failed: %d\n", error);
```

```
94          return error;
95      }
96
97      error = register_ftrace_function(&hook->ops);
98      if (error)
99      {
100         DMSG("register_ftrace_function() failed: %d\n", error);
101         ftrace_set_filter_ip(&hook->ops, hook->address, 1, 0);
102         return error;
103     }
104
105     return 0;
106 }
107
108
109 void fh_remove_hook(struct ftrace_hook *hook)
110 {
111     int err;
112
113     err = unregister_ftrace_function(&hook->ops);
114     if (err)
115     {
116         DMSG("unregister_ftrace_function() failed: %d\n", err);
117     }
118
119     err = ftrace_set_filter_ip(&hook->ops, hook->address, 1, 0);
120     if (err)
121     {
122         DMSG("ftrace_set_filter_ip() failed: %d\n", err);
123     }
124 }
125
126
127 int fh_install_hooks(struct ftrace_hook *hooks, size_t count)
128 {
129     int err;
130     size_t i;
131
132     for (i = 0; i < count; i++)
133     {
134         err = fh_install_hook(&hooks[i]);
```

```
135             if (err)
136             {
137                 while (i != 0)
138                 {
139                     fh_remove_hook(&hooks[--i]);
140                 }
141                 return err;
142             }
143         }
144 
145         return 0;
146 }
147 
148 
149 void fh_remove_hooks(struct ftrace_hook *hooks, size_t count)
150 {
151         size_t i;
152 
153         for (i = 0; i < count; i++)
154         {
155             fh_remove_hook(&hooks[i]);
156         }
157 }
```