

Министерство образования и науки Российской Федерации  
Санкт-Петербургский Политехнический Университет Петра Великого

—  
Институт компьютерных наук и кибербезопасности

## **ЛАБОРАТОРНАЯ РАБОТА № 1**

**«Реализация моделей безопасности в ОС Windows»**  
по дисциплине «Модели безопасности компьютерных систем»

Выполнила

студентка гр. 5131001/10302

Куковякина Д. А.

*<подпись>*

Преподаватель

Овасапян Т. Д.

*<подпись>*

Санкт-Петербург

2024

## **ПОСТАНОВКА ЗАДАЧИ**

Исследовать принципы разработки моделей безопасности, функционирующих на уровне ядра ОС Windows. Разработать драйвер-фильтр для ОС Windows 7/8/8.1/10, позволяющий ограничить права доступа процессов к объектам файловой системы. Разграничение должно осуществляться по правилам ролевой модели доступа.

## **ХОД РАБОТЫ**

### **Теоретические сведения**

Драйвер — это программа, которая работает как инструкция для операционной системы. Можно выделить следующие типы драйверов по их назначению:

- Драйвера физических устройств. Они необходимы для работы обычных устройств. Например принтеров, сканеров и другого оборудования.
- Фильтры файловой системы. Необходимы, например, для создания программных RAID (технология объединения двух и более накопителей в единый логический элемент с целью повышения производительности и отказоустойчивости) или шифрования дисков.
- Сетевые перенаправители. Это драйверы файловой системы, которые передают запросы по сети на другую машину. В качестве клиента в сетевой операции ввода/вывода отправляет запросы на сервер и обрабатывает ответы. Как сервер получает запросы ввода/вывода и обрабатывает их. Таким образом они позволяют приложению получать доступ к ресурсам на удаленных серверах и управлять ими, как если бы они находились на локальном компьютере.
- Драйвера протоколов. Они реализуют управление передачей в соответствии с принятым протоколом передачи, например, TCP/IP.

Еще можно разделить их на работающие в пользовательском режиме и в режиме ядра. В пользовательском режиме работают драйверы принтеров, они переводят аппаратно-независимые запросы в понятные принтеру команды. В режиме ядра работают, например, драйверы фильтры файловой системы.

Фильтры файловой системы перехватывают операции ввода-вывода (IRP-пакет — структура данных ядра Windows, обеспечивающая обмен данными между приложениями и драйвером, а также между драйверами) до того, как они будут выполнены в файловой системе. Это позволяет осуществлять мониторинг, перехват, управление, манипуляцию, и даже

прием/отклонение операций ввода-вывода до того, как их увидит файловая система. Тип фильтра файловой системы знакомый большинству — это антивирусный фильтр. Этот фильтр обычно перехватывает операции открытия файла, приостанавливает их и сканирует открываемый файл на предмет вредоносности. Если файл определяется как вирус, операция открытия может быть отменена. Если же нет, открытие файла происходит успешно.

### Мини-фильтр

Основная модель фильтров файловой системы в Windows – модель мини-фильтра, в которой используется поддержка менеджера фильтров. На рисунке 1 представлена схема функционирования Filter Manager.

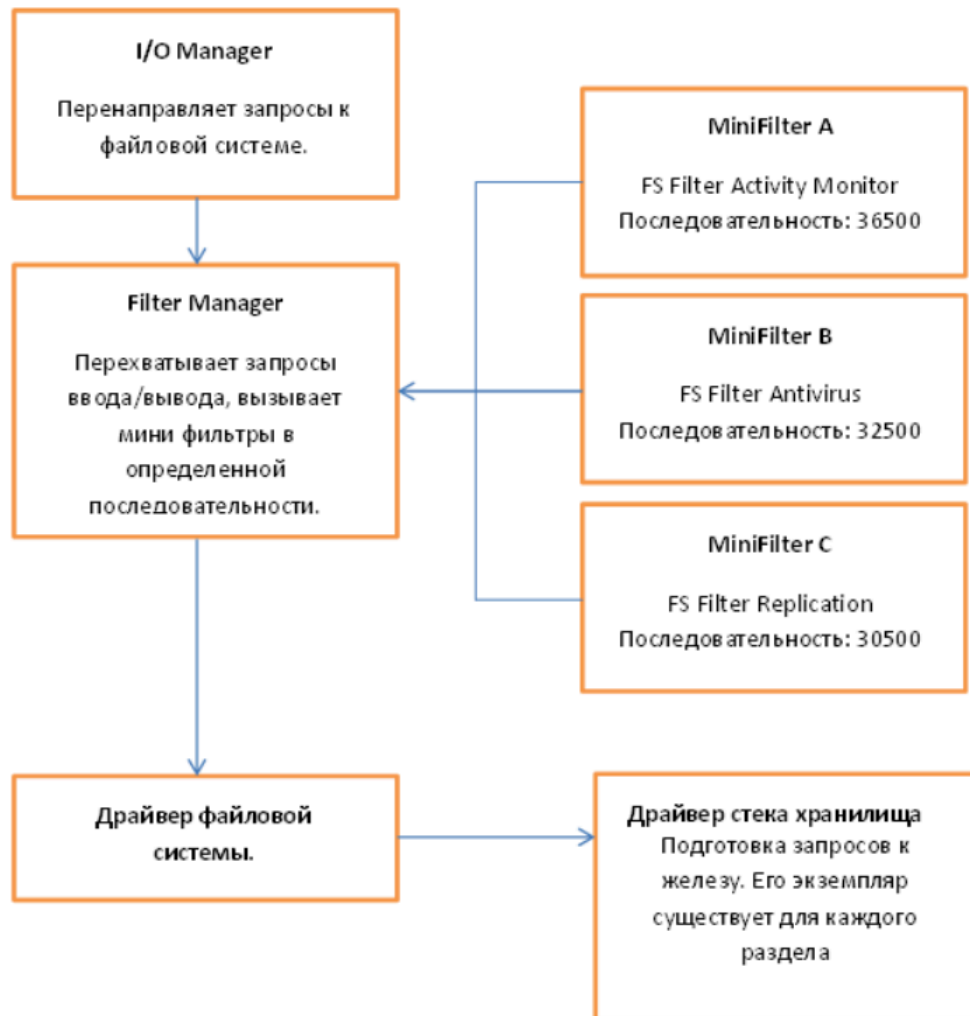


Рисунок 1 – Схема функционирования Filter Manager

Когда мини-фильтр регистрируется менеджером фильтров, он может выбрать получать ли функции обратного вызова PreOperation/PostOperation для определенных операций ввода-вывода.

Функции обратного вызова PreOperation - вызываются перед каждой операцией ввода-вывода указанного типа, которые выполняются в фильтруемой файловой системе.

Функции обратного вызова PostOperation - вызываются после того, как файловая система (и любой мини-фильтр “ниже”) обработала операцию ввода-вывода определенного типа.

### INF-файл

INF-файл для драйвера фильтра файловой системы обычно содержит следующие разделы:

- Версия – содержит класс, номер версии, издателя и другие данные (рисунок 2).

```
[Version]
Signature   = "$WINDOWS NT$"
Class       = "ActivityMonitor"
ClassGuid   = {b86dff51-a31e-4bac-b3cf-e8cfe75c9fc2}
Provider    = %Msft%
DriverVer   = 10/09/2001,1.0.0.0
CatalogFile =
PnpLockdown = 1
```

Рисунок 2 – Заполнение раздела Версия

- DestinationDirs – указываются каталоги, в которые будут скопированы файлы драйверов и приложений мини-фильтра, значение 12 относится к каталогу Drivers (%windir%\system32\drivers).
- DefaultInstall –директива CopyFiles копирует файлы драйверов и пользовательских приложений драйвера минифильтра в места назначения, указанные в разделе DestinationDirs.
- DefaultInstall.Services – содержит директиву AddService, которая определяет, как и когда загружаются службы определенного драйвера.

- `ServiceInstall` – содержит сведения, используемые для загрузки службы драйверов, например, тип службы (мини-фильтр – 2), когда её следует запускать (3 – диспетчером), зависимости (`FltMgr`— имя службы диспетчера фильтров), ссылка на раздел `AddRegistry`. Имя раздела `ServiceInstall` должно отображаться в директиве `AddService` в разделе `DefaultInstall.Services`.

- `AddRegistry` – добавляет ключи и значения в реестр. Драйверы минифильтра используют раздел `AddRegistry` для определения экземпляров минифильтра и указания экземпляра по умолчанию.

- Строки – определяет каждый токен `%strkey%`, используемый в INF-файле.

- `DefaultUninstall` – содержит директивы `DelFiles` и `DelReg` для удаления файлов и записей реестра.

- `DefaultUninstall.Services` – содержит директивы `DelService` для удаления служб драйвера минифильтра. Директива `DelService` всегда указывает флаг `SPSVCINST_STOPSERVICE` (0x00000200), чтобы остановить службу перед ее удалением.

Фильтр файловой системы обязан иметь уникальный идентификатор, называемый `altitude`, который определяет позицию по отношению к другим фильтрам в стеке файловой системы. Он также задается в `inf` файле, в разделе строки.

Менеджер фильтров реализован в виде устаревшего фильтра файловой системы и фильтрует все экземпляры файловой системы. Так как у экземпляра файловой системы могут существовать несколько фильтров (стандартная комплектация Windows включает в себя не менее девяти стандартных мини-фильтров), менеджер фильтров предоставляет системе набор уникальных идентификаторов, которые позволяют понять, в какое место, в иерархии фильтров, должен быть установлен мини-фильтр. Например, есть два мини-фильтра, которые могут быть установлены для условной файловой системы: антивирусный мини-фильтр и мини-фильтр

прозрачного шифрования данных. Чтобы антивирусный мини-фильтр выполнял свою работу, ему нужен доступ к расшифрованному содержимому файла. Поэтому, антивирусный мини-фильтр должен иметь более “высокий” уникальный идентификатор, чем мини-фильтр шифрования.

Для мониторов активности, которые включают в себя драйверы фильтров, которые отслеживают и сообщают о файловом вводе-выводе, задан диапазон 360000–389999.

### **Реализация**

В качестве основы был взят фильтр passThrough, предоставляющий пустые обработчики каждой операции над файлами.

В таблице 1 представлены основные функции и их назначение.

Таблица 1 – Функции

| Функция/структура                      | Назначение                                                                                                                                                                                                                                           |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FLT_REGISTRATION FilterRegistration    | Регистрация фильтра. Основные поля: Callbacks – ссылка на структуру, определяющую, что и при помощи каких функций обрабатывать; FilterUnload – функция, которая будет вызвана при отключении фильтра.                                                |
| FLT_OPERATION_REGISTRATION Callbacks[] | Указываются операции, которые необходимо перехватить, а также функции, которые будут вызываться, соответственно, до и после выполнения операции над файлом.                                                                                          |
| DriverEntry                            | Вызывается при загрузке драйвера для инициализации, которая будет применяться ко всем экземплярам драйвера. В рамках DriverEntry вызывается FltRegisterFilter для регистрации процедур обратного вызова в менеджере фильтров и FltStartFiltering для |

|                          |                                                                                                                                                                                                                                                                      |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                          | уведомления менеджера фильтров о том, что драйвер готов начать подключение к томам и фильтрацию запросов ввода-вывода.                                                                                                                                               |
| InstanceSetup            | Вызывается всякий раз, когда на томе создается новый экземпляр.                                                                                                                                                                                                      |
| InstanceTeardownStart    | Вызывается в начале процесса удаления. В этой процедуре драйвер должен завершить все ожидающие операции, отменить или завершить другую работу, такую как запросы ввода-вывода, сгенерированные драйвером, и прекратить постановку в очередь новых рабочих элементов. |
| InstanceTeardownComplete | Вызывается после завершения всех незавершенных операций ввода-вывода. В этой процедуре драйвер закрывает все файлы, которые все еще открыты.                                                                                                                         |
| Unload                   | Вызывается, когда мини-фильтр удаляется. Мы можем отклонить этот запрос на выгрузку, если это не обязательная выгрузка, обозначенная флагами параметр.                                                                                                               |
| InstanceQueryTeardown    | Вызывается, когда экземпляр удаляется вручную с помощью вызова FltDetachVolume или FilterDetach.                                                                                                                                                                     |
| OperationPassThrough     | Эта процедура является процедурой предварительной обработки для данного мини-фильтра.                                                                                                                                                                                |
| OperationStatusCallback  | Эта процедура вызывается, когда I/O операция возвращается из вызова в IoCallDriver. Это полезно для операций, где STATUS_PENDING                                                                                                                                     |



|                          |                                                                                                                                                                      |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                          | означает, что операция была успешно поставлена в очередь.                                                                                                            |
| PostOperation            | Эта процедура является процедурой постобработки для данного мини-фильтра.                                                                                            |
| DoRequestOperationStatus | Это определяет те операции, для которых мы хотим получить статус операции. Эти обычно это операции, которые возвращают STATUS_PENDING как обычный статус завершения. |

### Реализация функционала

Сперва была написана небольшая программа, осуществляющая запись и чтение файла.

Модель разграничения доступа – ролевая. Для этого были реализованы два конфигурационных файла. Первый содержит имена процессов и группы, к которым они принадлежат, записи имеют вид <имя процесса> <группа>. Второй содержит запреты для данных групп, записи имеют вид <группа> <директория> <write/read>.

Для реализации функционала драйвера была создана структура Groups, хранящая имя группы; процессы, входящие в группу, и их количество; директории с запретом на чтение и их количество; директории с запретом на запись и их количество.

В функцию DriverEntry был добавлен вызов функции parseGroups, отвечающей за обработку конфигурационных файлов и сохранение данных в структуру. В функцию OperationPassThrough была добавлена проверка прав. Сначала проверяется наличие данного процесса в одной из групп, далее, в зависимости от действия, проверяется, находится ли запрошенный файл в одной из запрещенных директорий. Если доступ к файлу должен быть закрыт – функция возвращает FLT\_PREOP\_DISALLOW\_FASTIO. В данном случае FltMgr не будет отправлять операцию ввода-вывода ни в какие драйверы

мини-фильтры ниже вызывающего или в файловую систему. На рисунке 3 представлена реализация предварительной обработки.

```
if (Data->Iopb->TargetFileObject->FileName.Length > 0)
{
    ACCESS_MASK desA = Data->Iopb->Parameters.Create.SecurityContext->DesiredAccess;

    if (Data->Iopb->MajorFunction == IRP_MJ_WRITE || Data->Iopb->MajorFunction == IRP_MJ_READ ||
        (Data->Iopb->MajorFunction == IRP_MJ_CREATE && ((desA & FILE_WRITE_DATA) || (desA & FILE_APPEND_DATA))))
    {
        //getting process name
        PEPROCESS peprocess;
        char* processName;
        char filename[MAX_NAME];
        wcstombs(filename, Data->Iopb->TargetFileObject->FileName.Buffer, MAX_NAME);

        HANDLE pid = PsGetCurrentProcessId();
        status = PsLookupProcessByProcessId(pid, &peprocess);
        if ((NT_SUCCESS(status)) || (!peprocess))
        {
            return FALSE;
        }
        processName = (CHAR*)gGetProcessImageFileName(peprocess);

        int i = 0, j = 0;
        for (; i < MAX_GROUPS; i++) {
            for (j = 0; j < groups[i].countProc; j++) {
                if (strcmp(processName, groups[i].processes[j]) == 0) {
                    DbgPrint("find in group %s process name: %s", groups[i].name, groups[i].processes[j]);

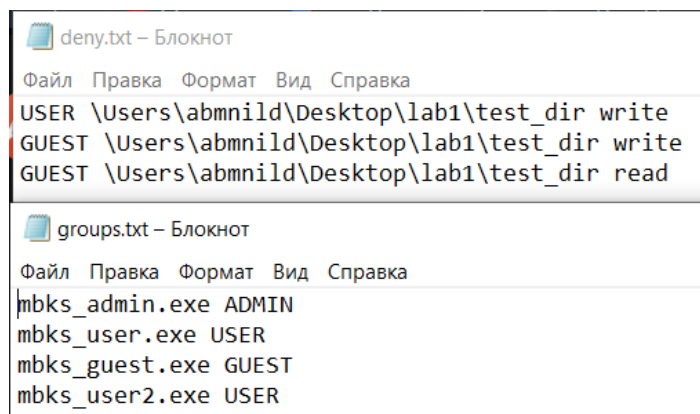
                    if (!checkAccess(i, Data->Iopb->MajorFunction, filename)) {
                        return FLT_PREOP_DISALLOW_FASTIO;
                    }
                    else {
                        return FLT_PREOP_SUCCESS_WITH_CALLBACK;
                    }
                }
            }
        }
    }
}

return FLT_PREOP_SUCCESS_WITH_CALLBACK;
```

Рисунок 3 – Предварительная обработка

В функцию PostOperation была добавлена обработка закрытия конфигурационных файлов для обновления групп и их прав без перезагрузки драйвера.

Далее представлены результаты работы программы. Для тестирования были заданы три группы в отношении директории test\_dir: ADMIN – всё разрешено, USER – разрешено только чтение, GUEST – всё запрещено. Конфигурационные файлы представлены на рисунке 4.



The image shows two Notepad windows. The top window, titled 'deny.txt - Блокнот', contains the following text:

```
USER \Users\abmnild\Desktop\lab1\test_dir write
GUEST \Users\abmnild\Desktop\lab1\test_dir write
GUEST \Users\abmnild\Desktop\lab1\test_dir read
```

The bottom window, titled 'groups.txt - Блокнот', contains the following text:

```
mbks_admin.exe ADMIN
mbks_user.exe USER
mbks_guest.exe GUEST
mbks_user2.exe USER
```

Рисунок 4 – Конфигурационные файлы

После запуска драйвера он выводит диагностические сообщения о полученных из конфигурационных файлов группах. Пример вывода представлен на рисунке 5.

```
Groups:
ADMIN
mbks_admin.exe
write:
read:
USER
mbks_user.exe
mbks_user2.exe
write:
\Users\abmnild\Desktop\lab1\test_dir
read:
GUEST
mbks_guest.exe
write:
\Users\abmnild\Desktop\lab1\test_dir
read:
\Users\abmnild\Desktop\lab1\test_dir
```

Далее представлены диагностические сообщения и результаты работы программы для всех трех пользователей (рисунок 6–8).

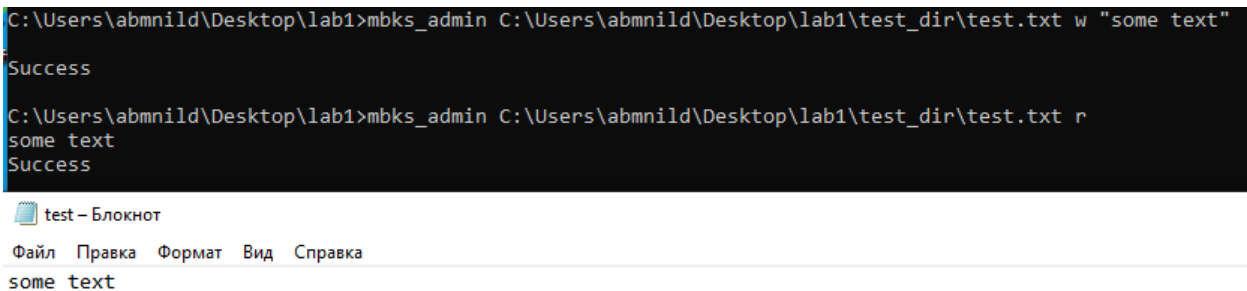


Рисунок 5 – Группа ADMIN



Рисунок 6 – Группа USER



Рисунок 7 – Группа GUEST

## **ВЫВОД**

В ходе работы были исследованы принципы разработки моделей безопасности, функционирующих на уровне ядра ОС Windows. Был разработан драйвер – фильтр, позволяющий ограничить права доступа процессов к объектам файловой системы по правилам ролевой модели доступа.