**Документация по коду**

**1. Модуль ядра**

* static int \_\_init mod\_init(void)

Точка входа в модуль, инициализирует внутренний список и патчит вызов stub\_execve.

* static void \_\_exit mod\_exit(void)

Вызывается при выгрузке модуля. Освобождает выделенные ресурсы, а так же возвращает stub\_execve в изначальное состояние.

* static int changeMemMode(unsigned long \*address, TMemMode mode)

Изменяет права доступа к странице виртуальной памяти с адресом address на read only или read write (параметр mode). Тип TMemMode описывается следующим образом:

typedef enum

{

MODE\_RW,

MODE\_RO

}TMemMode;

MODE\_RW — делает страницу доступной для чтения и записи;

MODE\_RO — делает страницу доступной только для чтения.

Аргументы:

address — виртуальный адрес страницы;

mode — устанавливаемые права доступа;

Возвращаемое значение

При ошибке возвращает -1, при успехе — 0.

* static int patchStubExecve(void)

Ищет в системном вызове stub\_execve команду call (код 0xE8), которая вызывает sys\_execve() и подменяет ее аргумент на адрес функции fakeExecve(). Изначальный адрес сохраняет в переменной sysExecve.

Возвращаемое значение:

При ошибке возвращает -1, при успехе — 0.

* static void unpatchStubExecve(void)

Возвращает системный вызов stub\_execve() в его изначальное состояние.

Возвращаемое значение:

При ошибке возвращает -1, при успехе — 0.

* static int changeSysCall(void)

Находит адрес таблицы системных вызовов, делает доступной для записи виртуальную страницу памяти, в которой находится системный вызов stub\_execve(), патчит его и делает страницу опять доступной только для чтения.

Возвращаемое значение:

При ошибке возвращает -1, при успехе — 0.

* static int returnSysCall(void)

Находит адрес таблицы системных вызовов, делает доступной для записи виртуальную страницу памяти, в которой находится системный вызов stub\_execve(), откатывает сделанные в ней изменения и делает страницу опять доступной только для чтения.

Возвращаемое значение:

При ошибке возвращает -1, при успехе — 0.

* asmlinkage long fakeExecve(const char \_\_user \*filename,

const char \_\_user \*const \_\_user \*argv,

const char \_\_user \*const \_\_user \*envp)

Функция, которой подменяется системный вызов sys\_execve(). Ее работа заключается в том, что она проверяет свой первый аргумент на наличие в черном списке и в зависимости от результата блокирует запуск процесса или вызывает стандартные обработчик.

Возвращаемое значение:

Если процесс находится в черном списке возвращает -EACCES, иначе — вызывает стандартный обработчик и возвращает то, что тот вернет.

* int makeHash(const char \*path\_to\_file, char \*hash\_result)

Считает хэш md5 от файла, путь к которому указан в первом аргументе и записывает результат во второй аргумент. Память под результат должна быть выделена до вызова.

Аргументы:

path\_to\_file — путь к файлу, хэш которого необходимо посчитать;

hash\_result — хэш файла. Память для результата должна быть выделена до вызова.

Возвращаемое значение:

При ошибке возвращает -1, при успехе — 0.

2. Модуль для работы с /proc

* **void initBlackList(void);**

Метод вызывается при загрузке модуля. В этом методе происходит создание каталога , используется функция

struct proc\_dir\_entry **\*proc\_mkdir**( const char \*name,

struct proc\_dir\_entry \*parent );

и файла в виртуальной файловой системе procfs. Инициализируется список.

struct proc\_dir\_entry \***proc\_create**( const char \*name,

mode\_t mode,

struct proc\_dir\_entry \*base,

read\_proc\_t \*read\_proc);

Последний аргумент функции – это структура, которая содержит в себе реализацию операций для файла в procfs.

static const struct file\_operations proc\_file\_fops = {

.owner = THIS\_MODULE,

.read = seq\_read,

.write = write\_callback,

.open=open\_callback,

};

*Возвращаемое значение функции initBlackList:*

При ошибке возвращает - ENOMEM, при успехе — 0.

* **size\_t write\_callback (struct file\* file, const char \_\_user\* buffer, size\_t count, int\* pos)**

Функция ***write\_callback*** вызывается автоматически при записи данных в файл в procfs. Добавляет путь к файлу и хэш этого файла в два отдельных списка.

В зависимости от записанной строки в этой функции может быть вызвана функция удаления из списка.

*struct file\* file* – структура для файла в который производится запись

*const char \_\_user\* buffer* – полученный буфер

*size\_t count*, - количество символов

*int\* pos* – позиция в файле для вставки

*Возвращаемое значение функции write\_callback:*

При ошибке возвращает -1, при успехе — количество записанных символов

* **ssize\_t read\_callback (struct seq\_file \*m, void \*v)**

Функция производит вывод содержимого файла /proc. Выводит содержимое списка.

*Возвращаемое значение функции write\_callback:*

Возвращает 0

* **releaseBlackList(void);**

Метод, который будет вызван при выгрузке модуля. В этом методе очищается содержимое списка. Удаляется файл и каталог.

*Возвращаемое значение функции*

При ошибке возвращает -1, при успехе — 1

* **findProcInBlackList(const char\* name);**

Метод поиска процесса в черном списке. В параметры этой функции передается имя процесса, который необходимо отыскать, считается хэш, далее происходит значение полученного хэша и хэша из списка.

*const char\* name –* имя процесса, которое необходимо найти

*Возвращаемое значение функции write\_callback:*

Если значение не надено возвращает -1, найдено — возвращает 0

Для работы со списком использовалась стандартная библиотека <linux/list.h>

* **int findProcAndDel(long pos,TBlackList mlist)**

Находит элемент в заданном списке по номеру и удаляет его.

*long pos –* номер элемента для удаления

*TBlackList mlist –* список в котором будет произведен поиск

*Возвращаемое значение функции write\_callback:*

Если значение не надено возвращает (-1), найдено — возвращает 0

**2. GUI**

class AFFL\_GUIFrame : public wxFrame

Класс главного окна приложения. Содержит в себе весь интерфейс и обработку колбэков.

**Конструкторы:**

AFFL\_GUIFrame(wxWindow\* parent,wxWindowID id = -1)

Создает новый объект дочерним для объекта parent. Если parent = 0, то окно не имеет родителя. Инициализирует все свои дочерние объекты, подгружает модуль ядра, подключает обработчики колбэков, инициализирует списки.

*Аргументы:*

parent — указатель на родительское окно;

id — идентификатор окна;

**Колбэки:**

void OnbtnKillClick(wxCommandEvent& event)

Убработчик кнопки btnKill. Убивает выделенный процесс.

void OnbtnAddClick(wxCommandEvent& event)

Обработчик кнопки btnAdd. Добавляет выделенный процесс в черный список.

void OnbtnAddPathClick(wxCommandEvent& event)

Обработчик кнопки btnAddPath. Позволяет добавить незапущенный процесс в черный список по полному пути к исполняемому файлу с помощью файлового диалога fdAddByPath.

void OnbtnDelClick(wxCommandEvent& event)

Обработчик кнопки btnDel. Удаляет выбраный процесс из черного списка.

void OntRefreshTrigger(wxTimerEvent& event)

Обработчик срабатывания таймера tRefresh. Обновляет список запущеных процессов один раз в секунду.

void OnKillFocus(wxFocusEvent& event)

Обработчик вызываемый при потере окном фокуса. Отключает таймер tRefresh чтобы не тратить ресурсы системы на обновление списка процессов, когда это не нужно.

void OnSetFocus(wxFocusEvent& event)

Обработчик вызываемый при получении окном фокуса. Включает таймер tRefresh, чтобы при работе с приложением список запущеных процессов обновлялся.

void OnClose(wxCloseEvent& event)

Обработчик вызываемый при закрытии приложения. Выгружает модуль ядра.

**Приватные методы:**

bool initKernelModule()

Подгружает модуль ядра при запуске программы.

//=======================================================

Class OpenListBox : public wxListBox

Определен для того чтобы сделать публичным метод DoDeleteOneItem родительского класса.

Публичные методы:

void removeSelected()

Удаляет выделенную строчку из списка.

//=======================================================

class BlackList

Является классом черного списка. Имеет в себе объект класса OpenListBox, который является виджетом черного списка, объект класса wxTextFile который предоставляет интерфейс для работы с физическим файлом и объект класса wxFile, который предоставляет интерфейс для работы с файлом из файловой системы procfs. В последнем случае выбран именно такой класс в связи со спецификой файла, который он представляет.

**Конструктор:**

BlackList(OpenListBox \*black\_list,

const wxString &procfs\_file\_name = DEFAULT\_PROCFS\_FILE\_NAME,

const wxString &phys\_file\_name = DEFAULT\_PHIS\_FILE\_NAME)

Создает объект класса OpenListBox, который являет собой виджет отображения для черного списка и инициализирует объекты для работы с файлами.

*Аргументы:*

black\_list — указатель на объект OpenListBox;

procfs\_file\_name — путь к виртуальному файлу, интерфейсу для работы с модулем ядра;

phys\_file\_name — путь к физическому файлу, в котором хранится черный список.

**Приватные методы:**

void init()

Открывает физический и виртуальный файлы черного списка, инициализирует виджет с черным списком, а так же черный список внутри модуля ядра. При этом невалидные записи удаляются.

**Публичные методы:**

void addByPath(const wxString &path)

Добавляет исполняемый файл по заданому пути в виджет черного списка, считает хэш-сумму, добавляет соответствующую запись в физический файл с черным списком и отправляет информацию о добавлении в модуль ядра.

*Аргументы:*

path — путь к исполняемому файлу.

void delSelected()

Удаляет выделенный исполняемый файл из черного списка и шлет модулю ядра соответствующее сообщение.

bool IsHashExist(const wxString &hash\_value)

Проверяет есть ли указанный хэш в файле с черным списком.

*Аргументы:*

hash\_value — проверяемый хэш.

//============================================================

class ProcList

Предоставляет собой класс для работы со списком процессов. Содержит в себе следующие объекты:

m\_proc\_list — объект класса wxGrid, который являет собой виджет, в котором отображается список запущеных процессов, для которых известен путь к исполняемому файлу;

m\_black\_list — объект класса BlackList, который являет собой интерфейс для работы с черным списком.

**Используемая структура:**

struct ProcInfo

{

wxString id;

wxString name;

wxString path;

};

Хранит информацию о процессе.

id — pid процесса;

name — имя исполняемого файла;

path — путь к исполняемому файлу вместе с именем.

**Конструктор:**

ProcList(wxGrid \*proc\_list, BlackList \*black\_list)

: m\_proc\_list(proc\_list), m\_black\_list(black\_list)

Инициализирует виджет для отображения списка процессов.

**Приватные методы:**

ProcInfo getProcInfo(wxString &pid)

Получает информацию о процессе с pid this\_name.

*Аргументы:*

pid — идентификатор процесса (имя каталога в папке /proc).

ProcInfo getSelectedProcInfo()

Возвращает информацию о выделенном процессе.

void addProcInfoLine(const ProcInfo &pi)

Добавляет информацию о процессе в виджет.

*Аргументы:*

pi — структура с информацией о процессе.

**Публичные методы:**

void killSelected()

Убивает выделенный процесс, посылая ему сигнал SIGKILL и обновляет список процессов.

void addSelectedToBlacklist()

Добавляет выделенный процесс в черный список и убивает его.

void update()

Очищает виджет со списком процессов и заполняет его заново.