1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. **Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА №2**

«**Исследование принципов разработки средств защиты»**

по дисциплине «Теория и системы управления информационной безопасностью**»**

1. Выполнили

студенты гр. 3651003/50801 Д.И. Кожушок

А.А. Корольков

Ю.В. Романюк

* 1. Руководитель Е.В. Жуковский

1. Санкт-Петербург
2. 2019

**1. Цель работы**

Изучение принципов разработки средств антивирусной защиты. Изучение технологий и механизмов ОС Windows, используемых для реализации функций защиты. Получение навыков проектирования и разработки многокомпонентного программного обеспечения.

**2. Формулировка задания**

Лабораторная работа выполняется в группах по 3 человека. В ходе выполнения лабораторной работы необходимо спроектировать и разработать программное обеспечение (ПО), реализующее основные функции, выполняемые современным антивирусным программном обеспечением.

Разработать следующие подсистемы:

1. Графический интерфейс. Отображение списка защищаемых файлов ПО, а также других настроек возможность его изменения.
2. Подсистема фильтрации. Драйвер-фильтр файловой системы и реестра, блокирующий доступ к заданным в реестре объектам системы.
3. Подсистема статического анализа. Сканирование файлов с использованием базы сигнатур ClamAV/ClamWin и известных YARA-правил.
4. Подсистема защиты реального времени. Реализовать DLL, которая перехватывала вызовы опасных функций. И блокировала опасные операции (на основе MITRE ATT&CK), например, попытки внедрения кода в чужие процессы (WriteProcessMemory, CreateThread).
5. Подсистема сетевой защиты. Сетевой фильтр-драйвер, осуществляющий логирование пакетов на основе заданных правил в конфигурационном файле: отправитель получатель, протокол, порты, содержимое.
6. Подсистема обновления и облачного сканирования. Клиент-серверное приложение, с возможностью отправки файлов и их скачивания. Анализ на сервере загруженного файла в песочнице и отправка клиенту вердикта о его безопасности на основе анализа. Логирование всей информации.

**3. Результаты работы**

1. Графический интерфейс.

В ходе работы был разработан графический интерфейс, который позволяет добавлять и удалять защищаемые файлы и папки в реестр, для последующего их чтения подсистемами защиты и фильтрации.

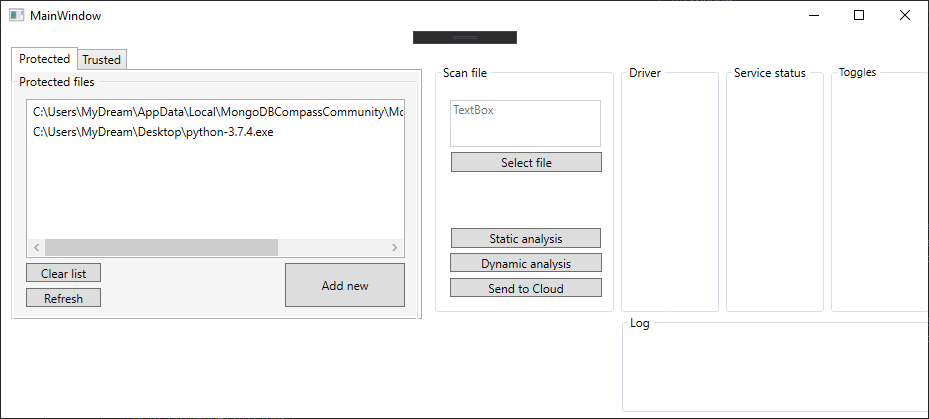


Рисунок 1 - Пользовательский интерфейс

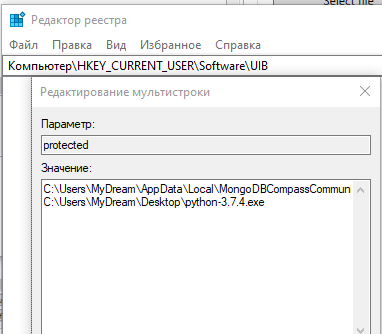


Рисунок 2 – Хранение списка защищаемых файлов

1. Подсистема фильтрации.

Был разработан модуль самозащиты. Защита от модификации указанных файлов, директорий и ключей реестра (включая, используемые разработанной программой). Информация о защищаемых файлах считывается из реестра.

Драйвер мини-фильтра может фильтровать операции ввода-вывода на основе IRP. Для каждой операции ввода-вывода, которую он выбирает для фильтрации, драйвер мини-фильтра может зарегистрировать процедуру обратного вызова перед операцией, процедура обратного вызова после операции, или и то, и другое.

Так как необходимо защищать файлы и директории от модификации, отслеживается операция записи:

const FLT\_OPERATION\_REGISTRATION Callbacks[] = {

{ IRP\_MJ\_WRITE, 0, MiniPreWrite, NULL },

{ IRP\_MJ\_OPERATION\_END }

};

В разработанном драйвере выполняется обработка до выполнения операции.

В драйвере роль точки входа выполняет функция DriverEntry, которая получает на вход указатель на структуру DriverObject, а также указатель на строку реестра, соответствующую загружаемому драйверу.

const FLT\_REGISTRATION FilterRegistration = {

sizeof(FLT\_REGISTRATION), // Size

FLT\_REGISTRATION\_VERSION, // Version

0, // Flags

NULL, // Context

Callbacks, // Operation callbacks

MiniUnload, // FilterUnload

NULL, // InstanceSetup

NULL, // InstanceQueryTeardown

NULL, // InstanceTeardownStart

NULL, // InstanceTeardownComplete

NULL, // GenerateFileName

NULL, // GenerateDestinationFileName

NULL, // NormalizeNameComponent

NULL

};

Структура DriverObject содержит множество полей, которые определяют поведение драйвера. Наиболее ключевые из них — это указатели на callback функции, то есть функции, которые будут вызываться при наступлении определенного события.

Одну из таких функций была определена: это функция MiniUnload. Указатель на данную функцию помещается в поле FilterUnload. Таким образом, при выгрузке драйвера сначала будет вызвана функция MiniUnload.

Регистрация драйвера осуществляется посредством вызова FltRegisterFilter(DriverObject, &FilterRegistration, &g\_FilterHandle). Для запуска процесса фильтрации: FltStartFiltering(g\_FilterHandle).

Для того чтобы получить список объектов из реестра устанавливается соединение с пользовательским приложением по communication port.

FltCreateCommunicationPort(g\_FilterHandle, &g\_Port, &oa, NULL, MiniConnect, MiniDisconnect, MiniMessageNotifier, 1);

Полученные объекты добавляются в UNICODE\_STRING\* g\_FoldersBlockModify, UNICODE\_STRING\* g\_FilesBlockModify, UNICODE\_STRING\* g\_KeysBlockModify.

В MiniPreWrite осуществляется блокировка действий. Выполняется сравнение имени файла, к которому применяется операция, со структурой g\_FilesBlockModify.

RtlCompareUnicodeStrings(

CurrentFileName.Buffer, g\_FilesBlockModify[i].Length / 2,

g\_FilesBlockModify [i].Buffer, g\_FilesBlockModify[i].Length / 2, TRUE );

Если имя файла присутствует в структуре, то выполняется блокировка:

if (retValue == 0) {

DbgPrint("[\*] Write file: %ws blocked", CurrentFileName.Buffer);

Data->IoStatus.Status = STATUS\_ACCESS\_DENIED; …}

Запрет на модификацию директорий осуществляется аналогично.

Защита ключей реестра осуществляется несколько иначе. Необходимо сделать так, чтобы данные, записанные в ключ реестра, который используется антивирусом, были недоступны. Таким образом, осуществляется не защита от модификации ключа реестра, а защита от открытия.

Изначально был зарегистрирован callback на открытие ключа реестра:

g\_RegistryCallbackTable[RegNtPreOpenKeyEx] = (PEX\_CALLBACK\_FUNCTION)RfPreOpenKeyEx;

status = CmRegisterCallbackEx(RfRegistryCallback, &AltitudeString, DriverObject, NULL, &g\_CmCookie, NULL);

Далее все выполняется аналогично запрету модификации файлов и директорий. Получаем полное имя открываемого ключа и проверяем на наличие его в списке g\_KeysBlockModify и блокируем открытие: status = STATUS\_ACCESS\_DENIED.

Модуль логирования операций. Сбор информации об опасных операциях и попытках доступа к защищаемым объектам ОС. Сохранение ее в БД.

После того, как драйвер определил попытку доступа к защищаемому объекту операционной системы, выполняется формирование сообщения для communication port и выполняется отправка сообщения в пользовательское приложение.

LogMessage.Code = DIRECTORY\_CODE;

LogMessage.Length = CurrentFileName.Length;

RtlCopyMemory(LogMessage.Buffer, CurrentFileName.Buffer, CurrentFileName.Length);

timeout.QuadPart = 100;

status = FltSendMessage(g\_FilterHandle, &g\_ClientPort, &LogMessage, sizeof(LogMessage), NULL, NULL, &timeout);

Пользовательское приложение получает сообщение от драйвера:

res = commPortClient->getMessage(message, code);

Формирует строку для базы данных и записывает в файл:

std::string logMessage;

if (code == Code::CodeDir)

logMessage = "Access to file " + message + "in the specified directory has been blocked";

if (code == Code::CodeFile)

logMessage = "Access to the specified file " + message + " has been blocked";

if (code == Code::CodeRegKey)

logMessage = "Access to the specified reg key " + message + " has been blocked";

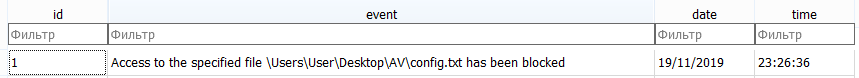
log.event = logMessage;

log.date = std::to\_string(now.tm\_mday) + "/" + std::to\_string(now.tm\_mon + 1) + "/" + std::to\_string(now.tm\_year + 1900);

log.time = std::to\_string(now.tm\_hour) + ":" + std::to\_string(now.tm\_min) + ":" + std::to\_string(now.tm\_sec);

bool res = db.insertLogEntry(log);

Пример записи в базе данных:



1. Подсистема статического анализа.

Для осуществления сканирования с использованием базы сигнатур ClamAV (база сигнатур должна храниться в директории rules) был разработан модуль на языке Python:

def YaraAnalyze(sample):

directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

path = directory + '\\rules\\'

for sig in os.listdir(path):

rules= yara.compile(filepath=path+sig)

matches = rules.match(data=sample)

for match in matches:

return True

return False

В качестве входных параметров передается уже считанный сканируемый файл.

Для ведения журнала анализы была создана таблица в базе данных:

CREATE TABLE check\_log (

md5 VARCHAR (32) NOT NULL,

sha1 TEXT NOT NULL,

sha256 VARCHAR (64) NOT NULL,

size INT NOT NULL,

detect\_sign BOOL NOT NULL,

detect\_ML BOOL

);

1. Подсистема защиты реального времени.

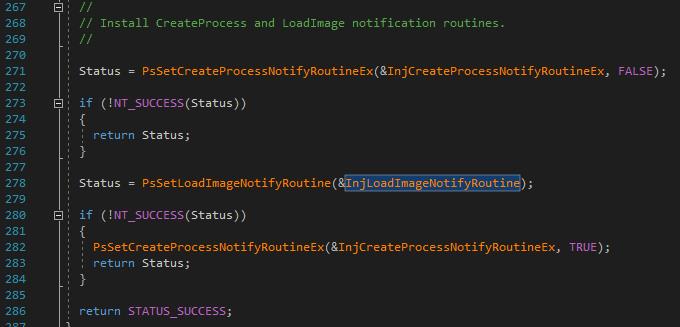
Для защиты в реальном времени было разработано средство динамического анализа, состоящее из dll библиотеки использующей DetoursNT, драйвер, выполняющий инъекцию разработанной dll в каждый запускаемый процесс, подменяя оригинальную ntdll.dll. 

Рисунок 3 - Инъекция кода в драйвере

В коде разработанной dll находятся обработчики вызываемых функций. При использовании отслеживаемой функции из ntdll.dll DetoursNT выполняет написанную нами обертку функции, которая отправляет полученную информацию о вызове в управляющее приложение, которое должно принимать решение о вредоносности процесса.

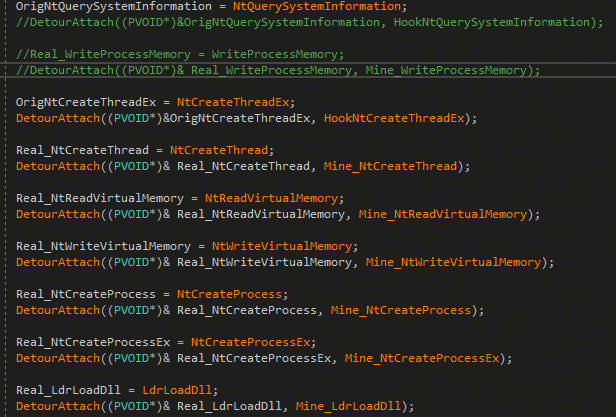


Рисунок 4 - Отслеживаемые функции из ntdll.dll

Выбор отслеживаемых функций проводилось, ориентируясь на рекомендации MITRE Attack.

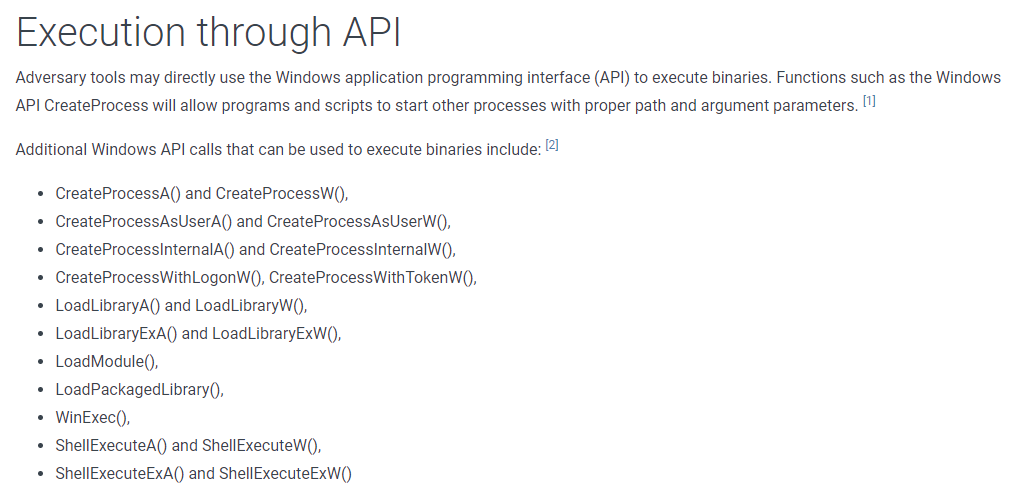


Рисунок 5 – Рекомендации MITRE к потенциально опасным функциям

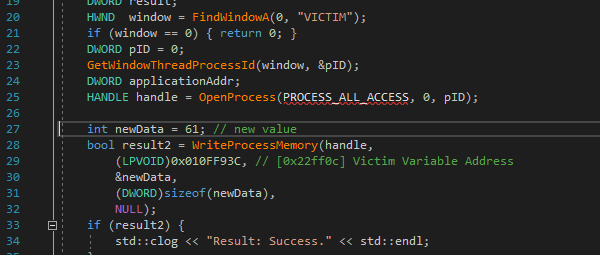
Так же было разработаны тестовые приложения, совершающие потенциально вредоносные действия. Первое из них совершало запись в память атакуемого приложения, изменяя значение переменной.  


Рисунок 6 - Код атакующего приложения

Второе приложение совершает запуск команд cmd, которые мы должны перехватить нашей dll.

При установке и запуске драйвера производится логирование операций следующего вида.

[PID:1952][TID:1984] Arch: x86, CommandLine: '"C:\Users\MyDream\Desktop\ShellTest.exe" '

[PID:1952][TID:1984] LdrLoadDll(DllName api-ms-win-core-synch-l1-2-0, DllHandle 0012F434)

[PID:1952][TID:1984] LdrLoadDll(DllName api-ms-win-core-fibers-l1-1-1, DllHandle 0012F444)

[PID:1952][TID:1984] LdrLoadDll(DllName kernel32, DllHandle 0012F444)

[PID:1952][TID:1984] LdrLoadDll(DllName api-ms-win-core-fibers-l1-1-1, DllHandle 0012F448)

[PID:1952][TID:1984] LdrLoadDll(DllName kernel32, DllHandle 0012F448)

[PID:1952][TID:1984] LdrLoadDll(DllName api-ms-win-core-synch-l1-2-0, DllHandle 0012F448)

[PID:1952][TID:1984] LdrLoadDll(DllName api-ms-win-core-localization-l1-2-1, DllHandle 0012EA64)

[PID:1952][TID:1984] LdrLoadDll(DllName kernel32, DllHandle 0012F46C)

[PID:1952][TID:1984] LdrLoadDll(DllName C:\Windows\system32\apphelp.dll, DllHandle 0012EF40)

[PID:1952][TID:1984] NtWriteVirtualMemory(ProcessHandle 00000028, BaseAddress00050000, 327711,BufferSize 32,Buffer )

[PID:1952][TID:1984] NtWriteVirtualMemory(ProcessHandle 00000028, BaseAddress00050020, 327763,BufferSize 52,Buffer ??5?]?O?-?D%?:??5?]?O?-?D%?:)???(

[PID:1952][TID:1984] NtWriteVirtualMemory(ProcessHandle 00000028, BaseAddress7FFD8238, 2147320379,BufferSize 4,Buffer )

[PID:1952][TID:1984] NtWriteVirtualMemory(ProcessHandle 00000028, BaseAddress00150000, 1376287,BufferSize 32,Buffer )

[PID:1952][TID:1984] NtWriteVirtualMemory(ProcessHandle 00000028, BaseAddress00150020, 1376339,BufferSize 52,Buffer ??5?]?O?-?D%?:??5?]?O?-?D%?:???(

[PID:1952][TID:1984] NtWriteVirtualMemory(ProcessHandle 00000028, BaseAddress7FFD5238, 2147308091,BufferSize 4,Buffer )

[PID:3304][TID:2752] Arch: x86, CommandLine: 'C:\Windows\system32\cmd.exe /c ping localhost'

[PID:2660][TID:2596] Arch: x86, CommandLine: 'ping 8.8.8.8'

1. Подсистема сетевой защиты.

Сетевой фильтр-драйвер, осуществляющий анализ пакетов. Для анализа используются следующие поля пакеты: IP-адрес отправителя, порт отправителя, IP-адрес получателя, порт получателя, протокол.

Возможные действия при классификации пакетов: разрешить, блокировать и вызвать callout. Callout – набор функций в драйвере, которые проводят инспекцию пакетов. Они имеют специальную функцию, выполняющую классификацию пакетов. Эта функция может принят следующее решение:

* разрешить (FWP\_ACTION\_PERMIT);
* блокировать (FWP\_ACTION\_BLOCK);
* продолжить обработку.

Также есть фильтры (filters) – правила, указывающие, в каких случаях вызывается тот или иной callout.

Регистрация callout:

NTSTATUS WfpRegisterCallout()

{

FWPS\_CALLOUT Callout = { 0 };

Callout.calloutKey = WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID;

Callout.flags = 0;

Callout.classifyFn = FilterCallback;

Callout.notifyFn = NotifyCallback;

Callout.flowDeleteFn = FlowDeleteCallback;

return FwpsCalloutRegister(DeviceObject, &Callout, &RegCalloutId);

}

classifyFn – классифицирующая функция, notifyFn – функция добавления\удаления фильтра, flowDeleteFn – закрытие обрабатываемого потока.

После регистрации callout его необходимо присоединить к определенному уровню:

NTSTATUS WfpAddCallout() {

FWPM\_CALLOUT callout = { 0 };

callout.flags = 0;

callout.displayData.name = L"EstablishedCalloutName";

callout.displayData.description = L"EstablishedCalloutName";

callout.calloutKey = WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID;

callout.applicableLayer = FWPM\_LAYER\_ALE\_AUTH\_CONNECT\_V4;

return FwpmCalloutAdd(EngineHandle, &callout, NULL, &AddCalloutId);

}

applicableLayer — уникальный идентификатор уровня, на который добавляется callout. "v4" в названии идентификатора означает версию протокола Ipv4. CONNECT в названии означает, что мы контролируем только установку соединения, то есть будем разрешать\блокировать соединение на этапе установки.

После того, как callout успешно добавлен в систему, нужно создать фильтр, то есть указать, в каких случаях будет вызываться наш callout, а именно - его классифицирующая функция. Новый фильтр создается функцией FwpmFilterAdd.

В FWPM\_FILTER есть структура FWPM\_FILTER\_CONDITION. Поле layerKey заполняется GUID’ом уровня (layer), к которому мы хотим присоединиться. В нашем случае необходимо указать FWPM\_LAYER\_ALE\_AUTH\_CONNECT\_V4.

В поле fieldKey нужно указать, что хотим контролировать — порт, адрес, приложение. Указываем WPM\_CONDITION\_IP\_REMOTE\_ADDRESS, что нас интересует IP-адрес. Значение fieldKey определяет, значения какого типа будут в структуре FWP\_CONDITION\_VALUE, входящей в FWPM\_FILTER\_CONDITION. В данном случае в ней содержится ipv4-адрес. Поле matchType определяет, каким образом будет производиться сравнение значения в FWP\_CONDITION\_VALUE с тем, что пришло по сети.

NTSTATUS WfpAddFilter()

{

FWPM\_FILTER filter = { 0 };

FWPM\_FILTER\_CONDITION condition[1] = { 0 };

FWP\_V4\_ADDR\_AND\_MASK AddrandMask = { 0 };

filter.displayData.name = L"filterCalloutName";

filter.displayData.description = L"filterCalloutName";

filter.layerKey = FWPM\_LAYER\_ALE\_AUTH\_CONNECT\_V4;

filter.subLayerKey = WFP\_SAMPLE\_SUB\_LAYER\_GUID;

filter.weight.type = FWP\_EMPTY;

filter.numFilterConditions = 1;

filter.filterCondition = condition;

filter.action.type = FWP\_ACTION\_CALLOUT\_TERMINATING;

filter.action.calloutKey = WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID;

condition[0].fieldKey = FWPM\_CONDITION\_IP\_REMOTE\_ADDRESS;

condition[0].matchType = FWP\_MATCH\_EQUAL;

condition[0].conditionValue.type = FWP\_V4\_ADDR\_MASK;

condition[0].conditionValue.v4AddrMask = &AddrandMask;

return FwpmFilterAdd(EngineHandle, &filter, NULL, &filterId);

}

В качестве классифицирующей функции была указана FilterCallback. Рассмотрим, что она выполняет. Функция принимает структуру FWPS\_INCOMING\_VALUES0\* Values, в которой содержится вся необходимая информация о пакете: IP-адрес отправителя и получателя, порты и протокол.

Для получения необходимых полей пакета выполняются следующие действия:

LocalIp = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_LOCAL\_ADDRESS].value.uint32;

RemoteIp = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_REMOTE\_ADDRESS].value.uint32;

LocalPort = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_LOCAL\_PORT].value.uint8;

RemotePort = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_REMOTE\_PORT].value.uint8;

Protocol = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_PROTOCOL].value.uint8;

В последующем будет осуществляться блокировка пакетов на основе правил фильтрации. Пример кода:

if (targetIp == RemoteIp)

{

DbgPrint("block\n");

classifyout->actionType = FWP\_ACTION\_BLOCK;

classifyout->rights &= ~FWPS\_RIGHT\_ACTION\_WRITE;

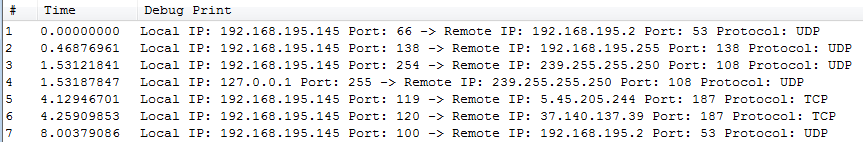
return;

}

else

classifyout->actionType = FWP\_ACTION\_PERMIT;

Пример логирования:



1. Подсистема обновления и облачного сканирования.

Было разработано клиент-серверное приложение.

Клиент считывает сканируемый файл и отправляет его на сервер с помощью сокетов по протоколу TCP.

Сервер получает файл от клиента и проводит анализ в песочнице Cuckoo Sandbox 2.0.6. Для этого был разработан модуль на языке Python:

def Analyze\_cuckoo(SAMPLE\_FILE):

REST\_URL = "http://localhost:8090/tasks/create/file"

with open(SAMPLE\_FILE, "rb") as sample:

files = {"file": ("temp\_file\_name", sample)}

r = requests.post(REST\_URL, headers=HEADERS, files=files)

task\_id = r.json()["task\_id"]

REST\_URL = "http://localhost:8090/tasks/report/"+str(task\_id)

time.sleep(60)

while(1):

r = requests.get(REST\_URL, headers=HEADERS)

if r.status\_code == 200:

time.sleep(30)

r = requests.get(REST\_URL, headers=HEADERS)

break

time.sleep(60)

todos=json.loads(r.text)

for todo in todos["behavior"]["apistats"]:

for t in todos["behavior"]["apistats"][str(todo)]:

count=todos["behavior"]["apistats"][str(todo)][str(t)]

return task\_id, r.text

Общение с Cuckoo Sandbox происходит с помощью REST API. Для старта анализа отправляется POST запрос с файлом. В ответ получаем идентификатор задачи. С помощью GET запроса и указанием идентификатора задачи происходит отслеживание выполнение анализа. Если возвращаемый код равен 200, то анализ прошел успешно и сформирован отчет в формате json. В отчете во вложенном объекте apistats хранятся вызываемые функции и их количество, что требуется для дальнейшего построения классификатора.

Для логирования была создана таблица в базе данных:

CREATE TABLE Check\_file (

id INTEGER NOT NULL,

IP TEXT NOT NULL,

File\_Name TEXT NOT NULL,

Vuln TEXT

);

**4. Выводы**

В ходе выполнения лабораторной работы были изучены принципы разработки средств антивирусной защиты. Были разработаны компоненты антивирусного приложения. В последующих лабораторных работах будет выполнено дорабатывание компонент – добавления функционала и объединение компонент в единое антивирусное приложение.