Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет математики, механики и компьютерных наук
Кафедра дифференциальных и стохастических уравнений

РАБОТА ПРОВЕРЕНА
Рецензент, руководитель
группы аналитики ГП «Рифарм»,
канд. техн. наук
_____ / Б.М. Кувшинов /
«___» _____2016 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой дифференциальных и
стохастических уравнений, ЮУрГУ,
доктор физ-мат. наук, доцент
_____ / С.А. Загребина /
«___» _____2016 г.

РАЗРАБОТКА СИСТЕМЫ ЗАЩИТЫ ПРИЛОЖЕНИЙ НА ОСНОВЕ ПРОФИЛЯ АКТИВНОСТИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2016.129-125.ВКР

Руководитель, канд. физ.-мат. наук,
доцент
_____ / С.М. Елсаков /
«___» _____2016 г.

Автор, студент группы ММиКН-293
_____ / Н.О. Артес /
«30» мая 2016 г.

Нормоконтролер, канд. физ.-мат. наук,
доцент
_____ / М.А. Сагадеева /
«___» _____2016 г.

Челябинск 2016

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет математики, механики и компьютерных наук
Кафедра дифференциальных и стохастических уравнений

З А Д А Н И Е

студенту группы ММиКН-293
Артесу Никите Олеговичу
на выпускную квалификационную работу
по направлению 09.04.04 – ПРОГРАММНАЯ ИНЖЕНЕРИЯ

1. **Тема диссертации:** «Разработка системы защиты приложений на основе профиля активности».
   (Утверждена приказом по университету от «15» апреля 2016г. № 661)
2. **Перечень подлежащих исследованию вопросов**
   2.1. Литературный обзор, выбор технологии противодействия вирусной активности
   2.2. Постановка задачи
   2.3. Разработка прототипа модуля активной защиты
3. **Календарный план подготовки выпускной квалификационной работы**

Наименование этапов дипломной работы	Срок выполнения этапов работы	Отметка о выполнении
1. Обзор литературы	01.02.16 – 14.02.16	
2. Выбор технологии противодействия вирусной активности	15.02.16 – 28.02.15	
3. Разработка прототипа модуля активной защиты	29.02.16 – 15.04.16	
4. Подготовка текста выпускной квалификационной работы	15.04.16 – 15.05.16	
5. Проверка и рецензирование работы руководителем, исправление замечаний	15.05.16 – 25.05.16	
6. Подготовка доклада и текста выступления	26.05.16 – 10.06.16	
7. Внешнее рецензирование	20.05.16 – 05.06.16	
8. Защита дипломной работы	10.06.16 – 20.06.16	

4. **Дата выдачи задания** «01» февраля 2016 г.

Руководитель работы
канд. ф.- м. наук, доцент _____ (подпись) ***Елсаков С.М.***

Задание принял к исполнению _____ (подпись) ***Артес Н.О.***

УДК - 4.056.5

Артес Н.О.

Разработка системы защиты приложений на основе профиля активности. /Н.О. Артес. – Челябинск, 2016. - 43 с.

Данная работа посвящена разработке прототипа модуля активной защиты против вредоносных программ для приложений под операционной системой «Windows». Прототип модуля реализован с помощью MS Visual Studio 2015 Community с использованием .Net Framework 4.6 на языке C#.

Библиографический список – 29 наим., 14 иллюстраций.

ОГЛАВЛЕНИЕ

Введение

Современные вредоносные программы — это хорошо организованный криминальный бизнес, вовлекающий в свою преступную деятельность высококвалифицированных системных и прикладных разработчиков ПО.[4]

Основные направления коммерческого кибермошенничества.

Компрометация компьютерных систем — с целью присоединения их к бот-сетям для слежения за жертвой, хищения хранящейся в системе информации, организации отказа в обслуживании и DDOS-атак.

Хищение средств аутентификации к системам дистанционного банковского обслуживания и платежным онлайн-системам — с целью дальнейшего хищения денежных средств.

Хищение данных банковских карт — с целью дальнейшего хищения денежных средств.

Причины роста хищений, совершаемых с помощью вредоносных компьютерных программ: рост количества вредоносных программ, изобретение вирусописателями еще более успешных новых угроз, использование вирусами уязвимостей, еще не закрытых производителями программное обеспечение (ПО), неправильные настройки безопасности (в том числе антивируса).[5;19;25]

Для атак на компьютерные системы предприятий кибермошенники успешно эксплуатируют: недостатки построения антивирусных систем защиты всех узлов корпоративной сети или полное отсутствие антивирусной системы защиты (речь не об использовании антивирусов, а именно о системах антивирусной защиты), недостатки или полное отсутствие на предприятиях политик информационной безопасности(ИБ), несоблюдение сотрудниками предприятий политик ИБ по причинам неграмотности в вопросах основ ИБ, неосознания проблемы, халатности,[6;23]

Для организации эффективной антивирусной системы защиты локальной сети ИБ-специалистам компании важно знать актуальные пути проникновения вредоносных программ в локальную сеть. Наиболее распространенными на сегодняшний день путями являются: уязвимости программного обеспечения, интернет и веб сайты, съемные устройства, электронная почта.[3;4]

Компьютерный вирус или компьютерный червь — это вредоносные программы, которые способны воспроизводить себя на компьютерах или через компьютерные сети. При этом пользователь не подозревает о заражении своего компьютера. Так как каждая последующая копия вируса или компьютерного червя также способна к самовоспроизведению, заражение распространяется очень быстро. Существует очень

много различных типов компьютерных вирусов и компьютерных червей, большинство которых обладают высокой способностью к разрушению.[26]

По данным «Лаборатории Касперского» в течении 2015 года было детектировано 121 262 075 уникальных вредоносных объектов (скрипты, эксплойты, исполняемые файлы и т.д.), программы-вымогатели обнаружены на 753 684 компьютерах уникальных пользователей, при этом программами-шифровальщиками было атаковано 179 209 компьютеров, 34% компьютеров пользователей интернета в течение года хотя бы раз подвергались вирусной атаке.[25]

Данная статистика позволяет сделать вывод, что риск заражения компьютера или рабочей станции достаточно велик и потому обеспечение информационной безопасности является значимым вопросом для обеспечения защиты и сохранности данных, особенной в закрытых организациях, например, в банках.

Данная работа посвящена разработке прототипа модуля активной защиты против вредоносных программ для приложений под операционной системой «Windows».

1. Обзор существующих угроз и типов защиты на платформе Windows

1.1. Обзор взаимодействия разных видов вредоносных программ на систему «Windows»

Известно большое количество типов вредоносных программ. Но каждый тип состоит из огромного количества образцов, также отличающихся друг от друга. Для борьбы со всеми ними нужно уметь однозначно классифицировать любую вредоносную программу и легко отличить ее от других таких же программ[1]. Однако у всех у них можно выявить общие особенности, рассмотрим для этого несколько видов вредоносных программ. Так как только лишь известных компьютерных вирусов в мире имеется огромное количество(в течении 2015 года было детектировано 121 262 075 уникальных вредоносных объектов[25]), рассмотрим топ 10 самых часто детектируемых вирусов по данным лаборатории Касперского.

1. Cryptolocker

В большинстве случаев троянская программа CryptoLocker попадает в систему через фишинговое сообщение электронной почты под видом отчета в формате PDF. Вместо файла в формате PDF в письме лежит исполняемый файл с именем, например, FORM.pdf.exe и с иконкой очень похожей на иконку PDF файла.[10]

Первое детектирование данного трояна было 5 сентября 2013 года.

Инфицирование.

После первого запуска Cryptolocker собирает в реестре различную информацию о системе и на ее основе генерирует для себя уникальное имя файла, представляющее собой набор латинских букв, и далее сохраняет себя на жестком диске жертвы в каталоге %AppData% (или %LocalAppData% в Win XP) с атрибутом скрытого файла.

Троян создает в реестре, в ветках автозапуска, два ключа, в которых прописан путь к Criptolocker'у на диске:

HKEY_CURRENT_USER\Software\Microsoft\Window\CurrentVersion\Run\CryptoLocker

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Once*CryptoLocker

Ключ с символом звездочки позволяет запускать CryptoLocker при старте системы в режиме Safe Mode.

Троян не скрывает себя в диспетчере задач, а для предотвращения остановки своего процесса запускает самого себя в качестве второго дублирующего процесса, который контролирует наличие основного и в случае необходимости повторно его запускает.

Работа с системой.

После запуска Cryptolocker генерирует доменное имя и пытается подсоединиться к этому доменному имени. В случае неудачи, генерация доменного имени и попытки подключиться продолжаются.

В качестве исходных данных для алгоритма генерации доменного имени используется системное время, получаемое путем вызова API GetSystemTime.

Доменное имя поочередно генерируется в следующих доменных зонах: .org, .co.ua, .info, .com, .net, .biz, .ru

В случае успешного подсоединения на сервере генерируется пара ключей для алгоритма шифрования RSA–2048. Закрытый ключ, предназначенный для расшифровки, остается храниться на сервере, а открытый, предназначенный для шифрования, передается на компьютер жертвы, где сохраняется в реестре в специально созданном разделе с именем параметра PublicKey.

В этом же разделе создается параметр с именем VersionInfo, в котором содержится информация о текущей версии трояна, IP-адрес командного сервера и время установки.

После успешного внедрения в систему и получения ключа для шифрования троян начинает поиск нужных файлов на всех доступных дисках поочередно с применением функций FindFirstFile и FindNextFile по определенным маскам.[18]

CryptoLocker шифрует выбранные файлы с помощью библиотеки Microsoft CryptoAPI и стандартного криптопровайдера Microsoft Enhanced RSA and AES Cryptographic Provider.

Содержимое файла криптуется с помощью алгоритма AES–256, ключ для которого генерирует API-функция CryptGenKey с параметром CALG _ AES _ 256. После шифрования файла этот AES-ключ шифруется алгоритмом RSA–2048 с помощью открытого ключа, полученного с командного сервера (и заблаговременно сохраненного в реестре), далее с открытого RSA-ключа берется SHA-хеш длиной 20 байт[7].

После того как файл зашифрован, Cryptolocker сохраняет путь и имя зашифрованного файла в реестр, в ветке HKEY_CURRENT_USER\SoftwareCryptoLocker_0388Files.

После шифрования всех доступных файлов на диске, CryptoLocker показывает предупреждающее окно с обратным отсчетом времени в 72 часа и требованием оплаты за получение ключа для расшифровки файлов.

Вариантов оплаты предлагается всего два — битокоинами и с помощью платежной системы MoneyPak. В случае истечения времени оплаты, Cryptolocker грозит удалением приватного ключа с сервера и дальнейшей невозможностью расшифровки файлов.

2. Net-Worm.Win32.Kido.ih

Net-Worm.Win32.Kido.ih - Сетевой червь, распространяющийся через локальную сеть и при помощи съемных носителей информации. Программа является динамической библиотекой Windows. Размер компонентов около 165 КБ.[11]

Первый раз червь был обнаружен 2 апреля 2009 года.

Инфицирование.

Червь копирует свой исполняемый файл в следующие папки со случайным именем:

%System%\<random>,

%Program Files%\Internet Explorer\<random>.dll,

%Program Files%\Movie Maker\<random>.dll,

%All Users Application Data%\<random>.dll,

%Temp%\<random>.dll,

%Temp%\<random>.tmp,

где <random> - случайная последовательность символов.

Для автозапуска при перезагрузке системы вредоносная программа создает службу, которая запускает исполняемый файл червя при каждой последующей загрузке Windows. При этом создается следующий ключ реестра:

[HKLM\SYSTEM\CurrentControlSet\Services\netsvcs]

Также червь изменяет значение следующего ключа реестра:

[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost]

netsvcs= <оригинальное значение> %System%\<rnd>.dll

При заражении компьютера вредоносная программа запускает HTTP сервер на случайном TCP порту, который используется для загрузки исполняемого файла вредоносной программы на другие компьютеры.

Червь получает список IP-адресов компьютеров, находящихся в сетевом окружении зараженного компьютера и производит на них атаку, использующую уязвимость переполнения буфера MS08-067 в сервисе «Сервер». Для этого вредоносная программа отсылает удаленной машине определенным образом сформированный RPC-запрос, вызывающий переполнение буфера при вызове функции wcscpy_s в системной библиотеке netapi32.dll, из-за этого запускается специальный загрузочный код, скачивающий с зараженного компьютера исполняемый файл червя и запускает его. После этого происходит инсталляция червя на атакуемом компьютере.[8]

Для того, чтобы воспользоваться вышеуказанной уязвимостью, программа пытается подключиться к удаленному компьютеру используя учетную запись администратора при этом перебирая различные комбинации паролей.

Червь копирует собственный исполняемый файл на все съемные диски со следующим именем:

<X>:\RECYCLER\S-<%d%>-<%d%>-<%d%>-<%d%>-<%d%>-<%d%>-<%chislo%>\<random>.vmx, где random – случайная последовательность строчных букв, chislo – произвольное число, X – буква съемного диска.

Одновременно с собственным исполняемым файлом программа помещает в корень каждого диска сопровождающий файл:

<X>:\autorun.inf

Этот файл запускает исполняемый файл червя каждый раз, когда пользователь открывает зараженный раздел при помощи программы "Проводник".

При запуске вредоносная программа внедряет свой код в адресное пространство одного из запущенных системных процессов "svchost.exe". Этот код выполняет следующий функционал:

1) отключает следующие службы: wuauserv, BITS

2) блокирует доступ к адресам, содержащим определенные строки.

Также червь может скачивать файлы по ссылкам вида:

http://<URL>/search?q=<%random%>

где random – случайное число, URL – ссылка, сформированная по специальному алгоритму в зависимости от текущей даты. Текущую дату червь запрашивает с одного из следующих сайтов:[18]

http://www.w3.org,

http://www.ask.com,

http://www.msn.com,

http://www.yahoo.com,

http://www.google.com,

http://www.baidu.com.

Файлы, скачанные червем, помещаются в каталог Windows (%System%) с оригинальными именами.

3. Virus.Win32.Virut.ce

Virus.Win32.Virut.ce - Файловый вирус, заражающий исполняемые файлы Windows. Является вредоносным кодом, который содержится в исполняемых файлах Windows.

Впервые данная программа была обнаружена 9 января 2013 года.[7]

Инфицирование.

Вирус внедряет свой код в адресное пространство всех запущенных в системе процессов. Внедренный код перехватывает следующие системные функции в библиотеке

ntdll.dll: NtCreateFile, NtCreateProcess, NtCreateProcessEx, NtOpenFile, NtQueryInformationProcess

при помощи которых следит за открываемыми файлами и запускаемыми приложениями. При обнаружении запуска нового процесса или открытия исполняемого файла вирус производит его заражение. Заражаются файлы с расширениями .EXE и .SCR. Вирус не заражает файлы, которые содержат в своем имени следующие строки : “WINC”, “WCUN”, “WC32”, “PSTO”. При заражении вирус расширяет последнюю PE-секцию зараженного файла и записывает туда свое полиморфное тело, после чего перенаправляет точку входа в программу на себя.

Вирус добавляет исполняемый файл процесса, в котором работает в список доверенных приложений Windows Firewall.

Отключает функцию восстановления системных файлов.

Вирус пытается соединиться со следующими IRC серверами: prox*****ircgalaxy.pl, irc*****ef.pl

если ему это удается, посылает серверу следующие команды: NICK dewxxpyi, USER b, JOIN #.<rnd1>, где rnd1 – случайное число.

Работа с системой.

После этого вирус переходит в режим приема команд от IRC сервера злоумышленников и выполняет их.

Вирус поддерживает следующие команды:

- !get - загрузка из интернет вредоносного кода и внедрение его в процессы на машине пользователя.

- !hosu - открытие указанных URL с компьютера пользователя.

Так же вирус сканирует жесткий диск компьютера в поисках файлов с расширениями:

HTM, PHP, ASP

и если находит вставляет в них следующую строку:

<iframe src="http://****.pl/rc/" width=1 height=1 style="border:0"></iframe>

4. Email-Worm.Win32.Runouce.b

Email-Worm.Win32.Runouce.b - Червь, распространяющийся через вложения в письма электронной почты. Программа является исполняемым файлом написанном на C++.

Впервые червь был обнаружен 16 января 2013 года.[12]

Инфицирование.

При первом запуске червь копирует самого себя в системный каталог Windows под именем: %System%\runouce.exe

Для автозапуска червь добавляет ссылку на свою копию в ключ автозапуска системного реестра: [HKLM\Software\Microsoft\Windows\CurrentVersion\Run] Runonce=%System%\runouce.exe

Работа с системой

Этот червь заражает исполняемые файлы и веб страницы и при запуске выполняет следующие действия:

1) ищет на локальном диске все исполняемые файлы и заражает их записывая самого себя в конец заражаемых файлов и перенаправлении точки входа.

2) проверяет все документы, текстовые файлы и файлы электронных таблиц на наличие адресов электронной почты

3) ищет все файлы веб-страниц, модифицирует их содержимое и копирует в их каталог файл почтового сообщения, содержащий во вложении копию червя. Таким образом, при открытии зараженной страницы на собранные адреса будут отосланы подготовленные червем сообщения. Сообщения имеют следующий вид:

От кого: TEST***3E9@yahoo.com

Тема письма: TEST***3E9 is comming!

Тело письма отсутствует.

В приложении файл с именем «pp.exe»

5. TROJAN-SPY:W32/ZBOT

W32 / Zbot фокусируется на воровстве онлайн банковской информации.

Впервые троян был обнаружен 9 мая 2010 года. Различные виды данного вируса детектируются до сих пор.[9]

Инфицирование.

Zbot создает в папке% WinDir% \ system32 \ wsnpoem папку, в которой он размещает два файла, video.dll и audio.dll. Эти файлы используются для хранения информации, украденный в зараженной системе, а также зашифрованный файл конфигурации, который троян скачивает из заданного местоположения. Папка wsnpoem и его содержание, как правило, скрыты[9].

В Zbot троян копирует себя в% WINDIR% \ system32 \ ntos.exe (или, в некоторых вариантах, ... \ oembios.exe).

Работа с системой.

Во время установки Zbot троян проверяет запущенные программы брандмауэра, связанных с процессами, такими как outpost.exe или zlclient.exe. Если какой-либо из этих процессов работает, троян только копирует себя в папку system32.

Zbot-троян открывает подключение к удаленному серверу и загружает зашифрованный конфигурационный файл. Этот файл содержит адрес, где троян позже загружает информацию, которую он украл, адрес, по которому он может загрузить новую версию самого себя и адрес другого файла конфигурации. Этот файл также определяет на какие сайты троян будет загружать украденную информацию.

Если жертва вводит учетные данные на интернет-сайте банка, троян перехватывает данные в веб-форме и загружает их на сервер определенный в конфигурационном файле трояна.

6. Email-Worm.Win32.Blare

Вирус-червь. Распространяется через интернет в виде файлов, прикрепленных к зараженным письмам и через IRC-каналы. Червь является приложением Windows (PE EXE-файл), имеет размер около 18KB, написан на Visual Basic. Упакован UPX (размер распакованного файла около 83KB).[13]

Инфицирование.

При запуске червь копирует себя с именем ACCOUNT_DETAILS.DOC.exe в каталог c:\progra~1\ и регистрирует этот файл в ключе автозапуска системного реестра Windows:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run Windows Task Manager = c:\progra~1\ACCOUNT_DETAILS.DOC.exe

Работа с системой:

Создает в корневом каталоге диска C: файл с именем WIN32.SORT-IT-OUT-BLAIR.TXT, в который записывает текстовую строку "Infected by the WIN32.SORT-IT-OUT-BLAIR Virus!", после чего пытается скопировать этот файл в следующие каталоги:

c:\inetpub\wwwroot\default.asp,

c:\inetpub\wwwroot\default.htm,

c:\inetpub\wwwroot\default.html,

c:\inetpub\wwwroot\index.asp,

c:\inetpub\wwwroot\index.htm,

c:\inetpub\wwwroot\index.html.

7. Trojan-Clicker.Win32.Mobs

Троянская программа, открывающая различные URL без ведома пользователя. Является приложением Windows (PE-EXE файл). Имеет размер 26 624 байта. Написана на Visual Basic.[14]

Инфицирование.

При запуске троянская программа копирует свой исполняемый файл в системный каталог Windows: %System%\service.exe

Для автоматического запуска при каждом последующем старте Windows троян добавляет ссылку на собственный исполняемый файл в ключ автозапуска системного реестра: [HKCU\Software\Microsoft\Windows\CurrentVersion\Run]"MyApp" = "%System%\service.exe"

Работа с системой.

Троян изменяет значения следующих ключей реестра: [HKCU\Software\Microsoft\Internet Explorer\Main], [HKCU\.DEFAULT\Software\Microsoft\Internet Explorer\Main], [HKEY_USERS\S-1-5-21-606747145-1060284298-839522115- 1003\.DEFAULT\Software], [HKEY_USERS\S-1-5-21-606747145-1060284298-839522115-1003\Software\Microsoft\ Internet Explorer\Main]

Периодически данная вредоносная программа открывает следующие ссылки в окне Internet Explorer:

http://www.countering.de/***2000/click.exe?a200639+1,

http://213.221.***.59/in.php?id=Daniel20gera,

http://213.221.***.42/rankem.cgi?id=daniel20,

http://520009810531-****.bei.t-online.de/index.htm,

http://www.countering.de/***2000/counter.exe?a200639+1

8. Virus.Win32.Glyn

Файловый вирус. Представляет собой Windows PE EXE-файл. Имеет размер 8192 байта.

При запуске вирус заражает все EXE-файлы в текущем каталоге и во всех родительских каталогах вплоть до корневого.[7;17]

Инфицирование.

После запуска вирус ищет файлы с расширением .exe в текущем и во всех родительских каталогах. При заражении использует следующую технику:

создаёт свою копию в текущем каталоге с именем GLYVEN.$$$;

записывает по смещению 0x2000 байт в GLYVEN.$$$ заражаемый EXE-файл и при этом шифрует его;

удаляет заражаемый файл, а GLYVEN.$$$ переименовывает в него.

Вирус не заражает файлы, размер которых меньше 8192 байт.

Работа с системой.

После инфицирования вирус создает в корневом каталоге скрытый файл с именем GLYVxxxx.exe, где xxxx — число, вычисляемое по специальному алгоритму. Извлекает из инфицированного файла оригинальный EXE-файл, расшифровывает его и запускает этот файл на исполнение.

После завершения работы файл GLYVxxxx.exe не удаляется. В результате чего в корневом каталоге может скопиться множество скрытых файлов с именами вида GLYVxxxx.exe.

Данный вирус завершает работу компьютера каждый день в определенное время.

9. IM-Worm.Win32.Funner

Trojan:W32/Dllpatcher изменяет файл dnsapi.dll, модуль является точкой для создания новых host файлов, которые содержат дополнительные имена хостов и IP-адреса.[15]

Инфицирование.

DLL патчер включается в файл установщика, который чаще всего загружен и сохранен в папку Темп, используя имя файла extensionupdate.exe

Затем загруженный файл выполняется. Файл extensionupdate.exe также имеет следующие файлы компонентов: libnspr4.dll, libplc4.dll, libplds4.dll, nss3.dll, nssutil3.dll, smime3.dll

Файл extensionupdate.exe создает файл новые хосты в % WINDIR% /system32 / например: - C: \ WINDOWS \ system32 \ neb \ okhb \vobg.dat

Длина пути новых хост файлов, однако должна соответствовать длине пути старых хост файлов. Новый файл использует '.dat' расширение и будет содержать следующие данные:

107.178.255.88 www.google-analytics.com,

107.178.255.88 ssl.google-analytics.com,

107.178.255.88 partner.googleadservices.com,

107.178.255.88 google-analytics.com,

107.178.248.130 static.doubleclick.net,

107.178.247.130 connect.facebook.net.

Затем DLL патчер размещает файл Dnsapi.dll (либо в% WINDIR% \ system32 \ или% Windir% \ SysWOW64 \) и изменяет его для использования новых хост файлов, которые он создал.

10. Trojan-Spy.Win32.Stonari

Троянская программа-шпион. Похищает конфиденциальную информацию. Является приложением Windows (PE EXE-файл). Написана на Borland Delphi. Имеет размер 289280 байт. Упакована при помощи Aspack. Размер распакованного файла — около 704 КБ.[16]

Инфицирование.

Троянская программа создает следующие ключи в системном реестре: [HKCR\Software\Microsoft\microware], [HKLM\Software\stonari]

Для автоматического запуска при каждом последующем старте системы вирус добавляет ссылку на свой исполняемый файл в ключи автозапуска системного реестра:

[HKLM\Software\Microsoft\Windows\CurrentVersion\Run],
[HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices]

Также изменяется значение ключа системного реестра: [HKCR\exefile\shell\open\command]

Работа с системой.

Программа-шпион выполняет снимки с экрана компьютера и следит за нажатиями на кнопки мыши и клавиатурными операциями.

Собранную информацию троянская программа записывает в файл %System%\setpass.dat и отсылает на электронный почтовый ящик злоумышленника.

Для отправки похищенных сведений используется следующий smtp-сервер: smtp.163.net

Выводы

Представим краткое описание взаимодействия с системой рассмотренных вредоносных программ в виде таблицы.(рисунок 1)

При анализе данной таблицы можно сделать вывод, что различные вредоносные программы, после проникновения в систему, в основном взаимодействуют с разделами реестра, например, SYSTEM и SOFTWARE, так же они копируют себя в системную директорию или папку, например, AppData, WinDir, Temp. Так же вредоносные программы работают с различными исполняемыми или веб файлами, а также с библиотеками dll. Так же можно отметить, что любая программа взаимодействует с «Windows» с помощью WIN API функций системы.

Название вредоносной программы	Инфицирование				Взаимодействие программы с системой			
	Работа с реестром	Работа с директориями и системными папками	Работа с файлами и библиотеками DLL	Работа с процессами	Работа с реестром	Работа с директориями и системными папками	Работа с файлами и библиотеками DLL	Работа с процессами
Cryptolocker	+				+	+		

Net-Worm.Win32.Kido.ih		+		+	+	+	+	
Virus.Win32.Virut.ce		+		+			+	
Email-Worm.Win32.Runouce.b	+	+			+	+	+	
TROJAN-SPY:W32/ZBOT		+	+		+	+		
Email-Worm.Win32.Blare	+	+				+		
Trojan-Clicker.Win32.Mobs	+	+			+			+
Virus.Win32.Glyn			+			+	+	+
IM-Worm.Win32.Funner			+	+		+		+
Trojan-Spy.Win32.Stonari	+					+	+	+

Рис 1. Описание взаимодействия с системой рассмотренных вредоносных программ

1.2. Обзор методов противодействия вирусной активности

При использовании любой информационной технологии следует обращать внимание на наличие средств защиты данных, программ, компьютерных систем.

Безопасность данных включает обеспечение достоверности данных и защиту данных и программ от несанкционированного доступа, копирования, изменения. Защита данных и программ от несанкционированного доступа, копирования, изменения реализуется программно-аппаратными методами и технологическими приемами. Одним из способов защиты данных является использование антивирусных программ.[20;23]

Антивирусная программа — специализированная программа для обнаружения компьютерных вирусов, а также нежелательных (считающихся вредоносными) программ вообще и восстановления зараженных (модифицированных) такими программами файлов, а также для профилактики — предотвращения заражения (модификации) файлов или операционной системы вредоносным кодом.[20;23]

Антивирусные программы по объектам защиты подразделяют на:[25;26]

1. Антивирусные продукты для защиты персональных компьютеров,
2. Антивирусные продукты для защиты рабочих станций,
3. Антивирусные продукты для защиты файловых и терминальных серверов,
4. Антивирусные продукты для защиты почтовых и Интернет-шлюзов,
5. Антивирусные продукты для защиты серверов виртуализации и др.

Так же антивирусные программы классифицируют по технологии защиты: [25;26]

1. Классические антивирусные продукты (продукты, применяющие только сигнатурный метод детектирования),
2. Продукты проактивной антивирусной защиты (продукты, применяющие только проактивные технологии антивирусной защиты),
3. Комбинированные продукты (продукты, применяющие как классические, сигнатурные методы защиты, так и проактивные).

Сигнатурный метод обнаружения

Сигнатурный метод обнаружения — метод работы антивирусов и систем обнаружения вторжений, при котором программа, просматривая файл или пакет, обращается к словарю с известными вирусами, составленному авторами программы. Обычно сигнатурой антивируса является MD5-хэш (16 байт) сгенерированный на основе тела известного вируса. В случае соответствия какого-либо участка кода просматриваемой программы известному коду (сигнатуре) вируса в словаре, программа антивирус может заняться выполнением одного из следующих действий:[24;25]

1. удалить инфицированный файл,
2. отправить файл в «карантин» (то есть сделать его недоступным для выполнения, с целью недопущения дальнейшего распространения вируса),
3. попытаться восстановить файл, удалив сам вирус из тела файла.

Такой вид сканирования позволяет определить вид атаки с высокой долей вероятности, без ложных срабатываний, а также несомненным достоинством являются малые ресурсные требования.[17]

Недостатком является неспособность выявить новые вирусы, которые отсутствуют в базе. Это требует постоянного обновления баз сигнатур, что в свою очередь требует доступа к интернету.

Проактивная защита

Проактивные технологии – совокупность технологий и методов, используемых в антивирусном программном обеспечении, основной целью которых, в отличие от реактивных (сигнатурных) технологий, является предотвращение заражения системы пользователя путем контроля действий приложения, а не поиск уже известного вредоносного программного обеспечения в системе.[17]

Проактивные технологии начали развиваться практически одновременно с классическими (сигнатурными) технологиями. Однако, первые реализации проактивных технологий антивирусной защиты требовали высокий уровень квалификации пользователя, т.е. не были рассчитаны на массовое использование простыми пользователями персональных компьютеров. Спустя десятилетие антивирусной индустрии стало очевидно, что сигнатурные методы обнаружения уже не могут обеспечить эффективную защиту пользователей. Этот факт и подтолкнул к возрождению проактивных технологий.

Технологии проактивной защиты ПО и приложений: эвристический анализ, поведенческий анализ, песочница[24;25]

Описание технологии эвристического анализа

Эвристический анализ — это совокупность функций антивируса, нацеленных на обнаружение неизвестных вирусным базам вредоносных программ, но в то же время этот же термин обозначает один из конкретных способов. [24;25]

Технология эвристического анализа позволяет на основе анализа кода выполняемого приложения, скрипта или макроса обнаружить участки кода, отвечающие за вредоносную активность.

Практически все современные антивирусные средства применяют технологию эвристического анализа программного кода. Эвристический анализ нередко используется

совместно с сигнатурным сканированием для поиска сложных шифрующихся и полиморфных вирусов. Методика эвристического анализа позволяет обнаруживать ранее неизвестные инфекции, однако, лечение в таких случаях практически всегда оказывается невозможным. В таком случае, как правило, требуется дополнительное обновление антивирусных баз для получения последних сигнатур и алгоритмов лечения, которые, возможно, содержат информацию о ранее неизвестном вирусе. В противном случае, файл передается для исследования антивирусным аналитикам или авторам антивирусных программ.[28;29]

Чрезмерная подозрительность эвристического анализатора может вызывать ложные срабатывания при наличии в программе фрагментов кода, выполняющего действия и/или последовательности, в том числе и свойственные некоторым вирусам. В частности, распаковщик в файлах, запакованных PE-упаковщиком (Win)Upack вызывает ложные срабатывания целого ряда антивирусных средств, не признающих такой проблемы.[27]

Наличие простых методик обмана эвристического анализатора. Как правило, прежде чем распространять вредоносную программу (вирус), её разработчики исследуют существующие распространенные антивирусные продукты, различными методами избегая её детектирование при эвристическом сканировании. К примеру, видоизменяя код, используя элементы, выполнение которых не поддерживается эмулятором кода данных антивирусов, используя шифрование части кода и др.[21]

Даже при успешном определении, лечение неизвестного вируса практически всегда является невозможным. Как исключение, некоторыми продуктами возможно лечение однотипных и ряда полиморфных, шифрующихся вирусов, не имеющих постоянного вирусного тела, но использующих единую методику внедрения. В таком случае, для лечения десятков и сотен вирусов может существовать одна запись в вирусной базе, как это реализовано, к примеру, в антивирусе AVAST.[2]

Описание технологии песочницы

Песочница - одна из самых молодых технологий в антивирусах. Технология песочницы работает по принципу ограничения активности потенциально вредоносных приложений таким образом, чтобы они не могли нанести вреда системе пользователя.[27]

Ограничение активности достигается за счет выполнения неизвестных приложений в ограниченной среде – собственно песочнице, откуда приложение не имеет прав доступа к критическим системным файлам, веткам реестра и другой важной информации. Технология ограничения привилегий выполнения является эффективной технологией противодействия

современным угрозам, но, следует понимать, что пользователь должен обладать знаниями, необходимыми для правильной оценки неизвестного приложения.

Как правило, песочницы используют для запуска непроверенного кода из неизвестных источников, как средство проактивной защиты от вредоносного кода, а также для обнаружения и анализа вредоносных программ. Также зачастую песочницы используются в процессе разработки программного обеспечения для запуска «сырого» кода, который может случайно повредить систему или испортить сложную конфигурацию. Такие песочницы копируют основные элементы среды, для которой пишется код, и позволяют разработчикам быстро и безболезненно экспериментировать с неотлаженным кодом.[22]

Описание технологии защиты на основе поведенческого анализа.

Технология анализа поведения основывается на перехвате всех важных системных функций или установке т.н. мини-фильтров, что позволяет отслеживать всю активность в системе пользователя. Самой распространенной технологией поведенческого анализа является инжектирование dll.[23]

Инжектирование dll или «хуки» — это технология перехвата вызовов функций в сторонних процессах. «Хуки», как и любая достаточно мощная технология, могут быть использованы как в благих целях (снифферы, аудио\видеограбберы, расширения функционала закрытого ПО, логирование, багфиксинг) так и со злым умыслом (трояны, кряки, кейлоггеры).[23]

При запуске любого приложения — операционная система создаёт его процесс затем exe-файл копируется в память, далее определяется какие именно библиотеки (dll-файлы) ему нужны для работы (эта информация записана в начале каждого exe-файла), эти библиотеки ищутся и загружаются в память процесса. Потом определяется, какие именно функции библиотек использует программа и где они находятся (в какой библиотеке и где именно в этой библиотеке). Строится таблица импорта функций вида «функция SomeFunction1() — библиотека SomeLibrary1.dll — %адрес_функции_SomeFunction1()%». Когда программе понадобиться вызвать эту функцию — она найдет в своей памяти нужную библиотеку, отсчитает нужный адрес и передаст туда управление(рисунок 2).[28;29]

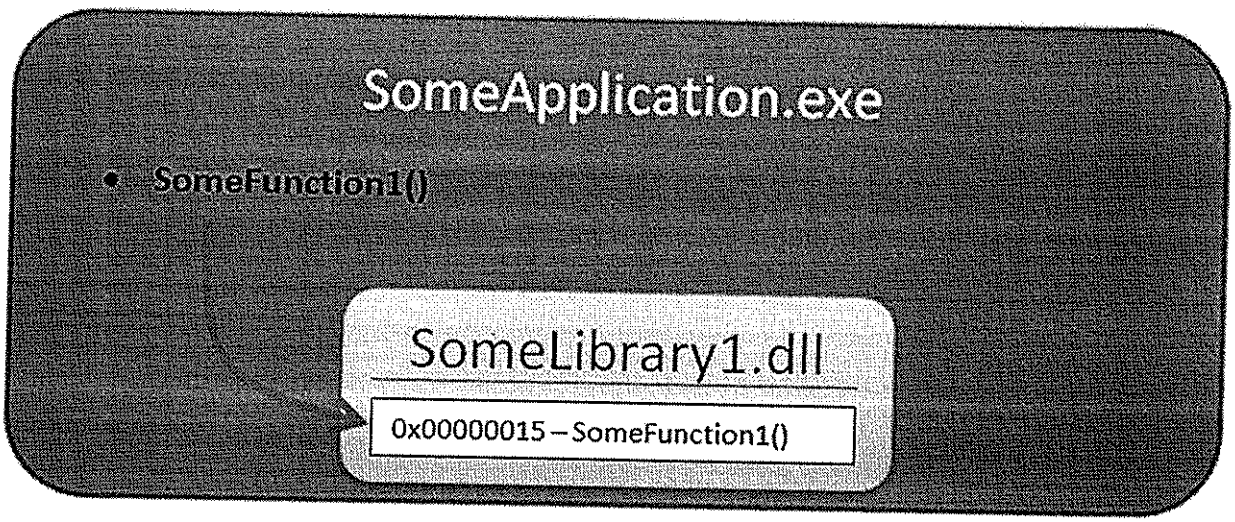

Рис 2. Фрагмент таблицы импорта функций

Суть хукинга — подмена адреса функции в таблице импорта функций на адрес инжектируемой функции(Рисунок 3).

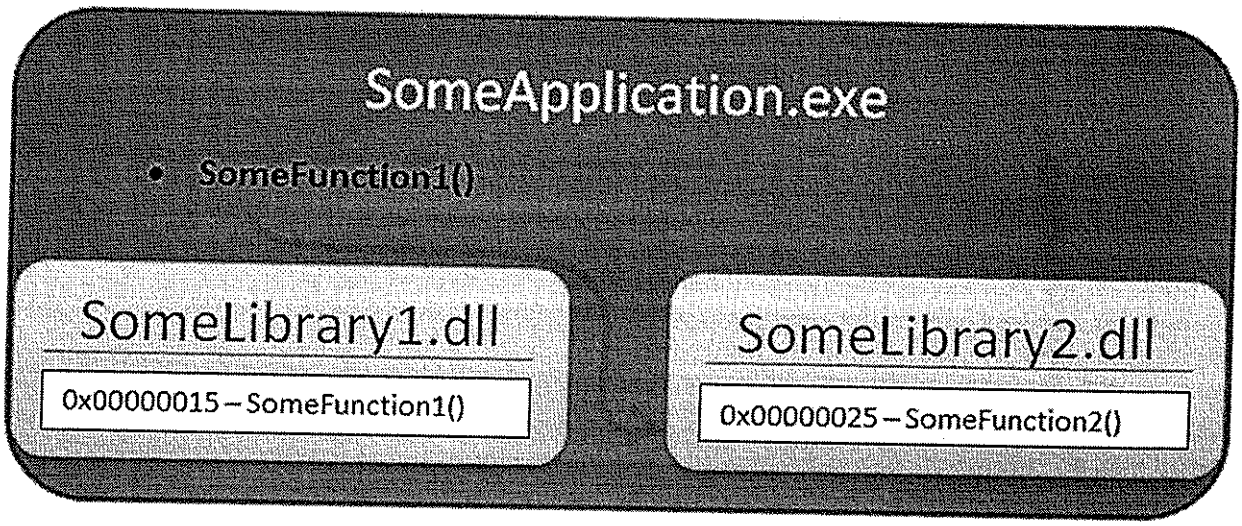

Рис 3. Измененная таблица импорта функций

Делается это таким образом — пишется своя библиотека SomeLibrary2.dll, в которой будет находится функция SomeFunction2(). Далее эта библиотека загружается в память чужого процесса и изменяется таблица импорта функций так, чтобы теперь она содержала запись «функция SomeFunction1() — библиотека SomeLibrary2.dll — %адрес _функции_SomeFunction2()%».

1.3. Постановка задачи

Дана рабочая станция, в которой важно настроить защиту системы от вредоносного ПО наиболее эффективно, желательно так, чтобы вредоносное ПО вообще не попало в систему. Примером может быть выделенный компьютер оператора, настроенный для работы в приложении «Клиент-банк».

В предыдущих разделах мы рассмотрели топ 10 вредоносных программ и отметили, что любая такая программа взаимодействует с системой с помощью WIN API функций. Так же мы рассмотрели несколько способов противодействия вирусам. Разобрав все способы противодействия вирусной активности можно сделать вывод, что проактивная защита является самой простой в реализации и не требующая дальнейших обновлений, так же данный тип защиты предполагает обнаружение раннее неизвестных вирусов. Было отмечено, что самой распространенной технологией поведенческого анализа является инжектирование dll и перехват WIN API функций системы. Учитывая это и то, что вредоносные программы взаимодействуют с системой с помощью таких же WIN API функций, можно утверждать, что инжектирование dll при перехвате всех системных WIN API функций обеспечивает максимальную защиту компьютера от вирусной активности.

Из всего этого можно сделать вывод, что для нашей задачи создания модуля, потиводействующего вирусным угрозам, больше всего подходит способ поведенческого анализа.

Так как в системе Windows имеется большое количество WIN API функций, то ставится задача разработки прототипа программы, имеющей ограниченную библиотеку перехватываемых WIN API функций, выполняющий поведенческий анализ действий пользователя в любом приложении и на основе этих действий осуществляющей защиту системы.

Модуль программы должен быть легко поддерживаемым – так как на платформе Windows процессы в системе могут взаимодействовать с большим количеством WIN API функций, архитектура модуля должна предполагать для программиста простой способ добавления новой API функции для перехвата.

Модуль должен автоматически создавать файлы белого списка для каждого процесса Windows. Количество процессов, защищаемых нашим модулем, должно определяться пользователем модуля. Так же в модуле должно быть реализовано простое переключение с режима сбора сведений на режим активной защиты.

2. Разработка прототипа модуля защиты системы на основе действий приложения

2.1. Обзор библиотек для инжектирования dll.

Microsoft Detours

Microsoft Detours — проект, разрабатываемый в лабораториях Microsoft Research, позволяющий перехватывать Win32 API-вызовы. Win64 API-вызовы так же поддерживаются, однако эта версия Microsoft Detours является коммерческой. Другая проблема состоит в том, что последнее обновление этого продукта датируется декабрем 2006 года, а значит более сей продукт не поддерживается.

Итак, сравнительная характеристика; достоинства:

1. Разработка от производителя операционной системы,
2. Бесплатна для Win32 API-вызовов,
3. Не требует знания ассемблера.

Недостатки:

1. Платна для коммерческого использования или x64-архитектуры,
2. Продукт давно не поддерживается.

madCodeHook

Первая её версия вышла в 2000-ом году, предназначалась для использования под Delphi. Тем ни менее за последующие годы автор весьма неплохо её развивал: сделал SDK для C++, внедрил поддержку 64-битных систем, всех версий Windows от 9x до Win 8.1, реализовал драйвер для внедрения хуков во все созданные процессы. Однако, данный продукт является коммерческим для всех типов архитектур windows.

Достоинства:

1. стабильная поддержка всех известных видов windows,
2. стабильная работа во всех типах архитектур.

Недостатки:

1. библиотека madCHook.dll должна быть предварительно скопирована в System32,
2. должно быть вручную запущено входящее в комплект приложение mchEvaluation.exe,
3. продукт полностью коммерческий.

EasyHook:

Библиотека с открытым исходным кодом, доступном на GitHub. Написана на C# и, как следствие, требует конкретную версию . NET Framework, хотя начиная с Windows 7 последние версии .NET Framework поставляются вместе с установщиком Windows. На данный момент для работы EasyHook нужна версия .NET Framework 3.5 / 4.0. Работает на

всех типах Windows, начиная с XP SP2. Так же стабильно работает на всех типах архитектур.

Достоинства:

1. полностью бесплатный продукт,
2. работа с любым типом архитектур,
3. стабильная работа начиная с win xp sp2.

Недостатки:

1. возможна нестабильная работа в последней версии EasyHook.

2.2. Описание архитектуры и разработки прототипа приложения, выполняющей защиту системы на основе действий пользователя.

Диаграмма вариантов использования

Диаграмма вариантов использования модуля активной защиты представлена на рисунке 4.

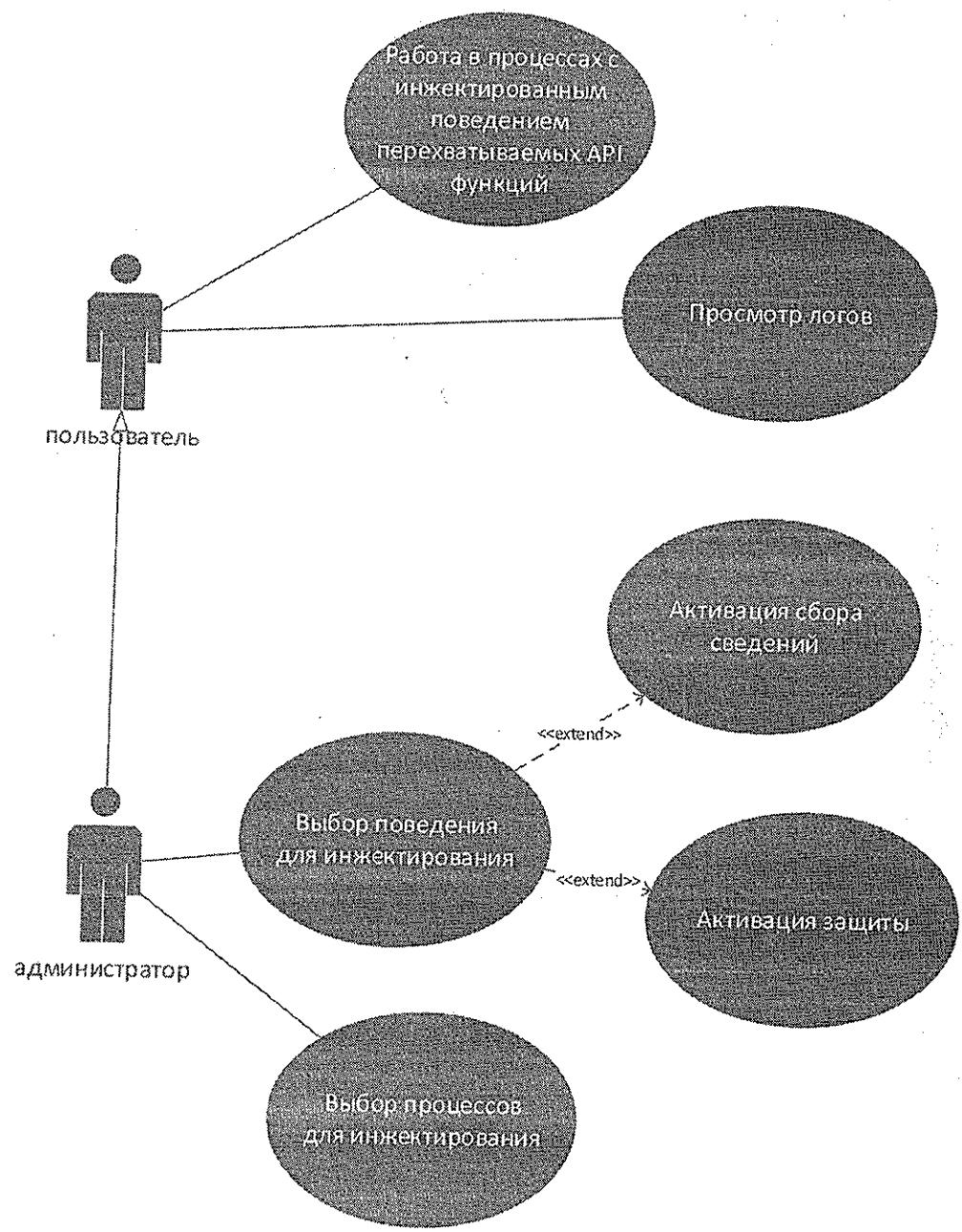

Рис 4. Диаграмма вариантов использования

На данной диаграмме вариантов использования представлено несколько «акторов», то есть внешних, по отношению к моделируемой подсистеме сущностей, которые взаимодействуют с системой и используют ее функциональные возможности для достижения определенных целей или решения частных задач. При этом акторы служат для

обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой.

Рассмотрим каждого актора в подробнее.

1. «Администратор» - это пользователь, который имеет право выбирать процессы для инжектирования, запущенные на рабочей станции, инжектирование поведения сбора сведений и активной защиты. Так же администратор может просматривать файл белого списка WIN API функций и просматривать файл логов, в котором прописываются системные сообщения программы, а также директории или ключи системного реестра не входящие в белый список, но в которые пытались получить доступ перехватываемые WIN API функции.
2. «Пользователь» - отличается от администратора тем, что не может инжектировать различные поведения в любой из процессов на компьютере, однако он может просматривать файл логов и работать в процессах с инжектированным поведением перехватываемых WIN API функций.

Рассмотрим каждое действие подробнее:

1. Работа в процессах с инжектированным поведением – это возможность пользователю и администратору работать в процессах к которому подключен наш модуль.
2. Просмотр логов – возможность просматривать системные сообщения и записанные директории или ключи реестра, к которым перехваченные WIN API функции не санкционировано пытались попасть
3. Выбор процессов для инжектирования – это действие, связанное с выбором нужного процесса из списка текущих процессов на данном компьютере для инжектирования определенного поведения.
4. Выбор поведения для инжектирования – это действие, которое состоит из двух под действий:

   4.1. Активация сбора сведений – инжектирование в выбранный процесс поведения перехвата определенных WIN API функций и составление профиля активности из так называемых ключей белого списка, которыми могут быть ключами реестра, директориями, IP-адресами, к которым перехваченные WIN API функции пытались получить допуск. Выбор ключа белого списка или ключа профиля активности определяется программистом и выбирается с учетом определенной перехватываемой WIN API функции. Для каждой перехватываемой WIN API функции создается собственный файл белого списка – профиль активности приложения.

4.2. Активация защиты – это инжектирование в выбранный процесс поведения перехвата тех же WIN API функций и установки валидности доступа к ключу профиля активности, к которым эти WIN API функции пытались произвести допуск. Допуск к ключу считается не валидным, если его нет в файле белого списка. В этом случае этот ключ записывается в файл логов с пометкой какая WIN API функция пыталась произвести доступ.

Архитектура и диаграмма классов модуля активной защиты

Архитектура модуля активной защиты имеет многоуровневый шаблон и разделена на две части – слой инжектирования и слой поведения перехватываемых WIN API функций в выбранном процессе. Слой поведения модуля активной защиты имеет два поведения - режим сбора сведений и режим защиты. Архитектура модуля представлена на рис 5.

На слое перехвата модуль инжектирует в выбранных процесс одно из двух поведений, описанное на слое поведения.

В режиме сбора сведений модуль активной защиты реализует поведение, заставляющее определенные перехватываемые WIN API функции добавлять в файл белого списка посещенные процессом директории, ключ реестра и т.д. В режиме защиты модуль так же инжектирует поведение, которое уже сверяет деятельность тех же перехватываемых WIN API функций с составленным ранее профилем активности. Если же деятельность

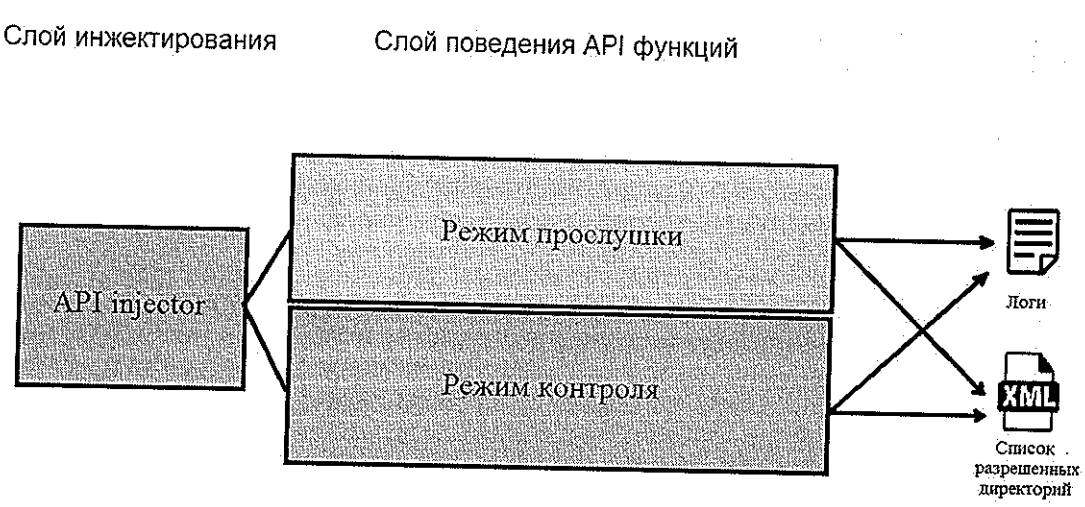

Рис 5. Архитектура модуля активной защиты

перехватываемой WIN API функции не валидна, модуль создает запись в логе и не дает завершить работу WIN API функции.

На рисунке 6 изображена диаграмма классов уровня перехвата и инжектирования модуля активной защиты.

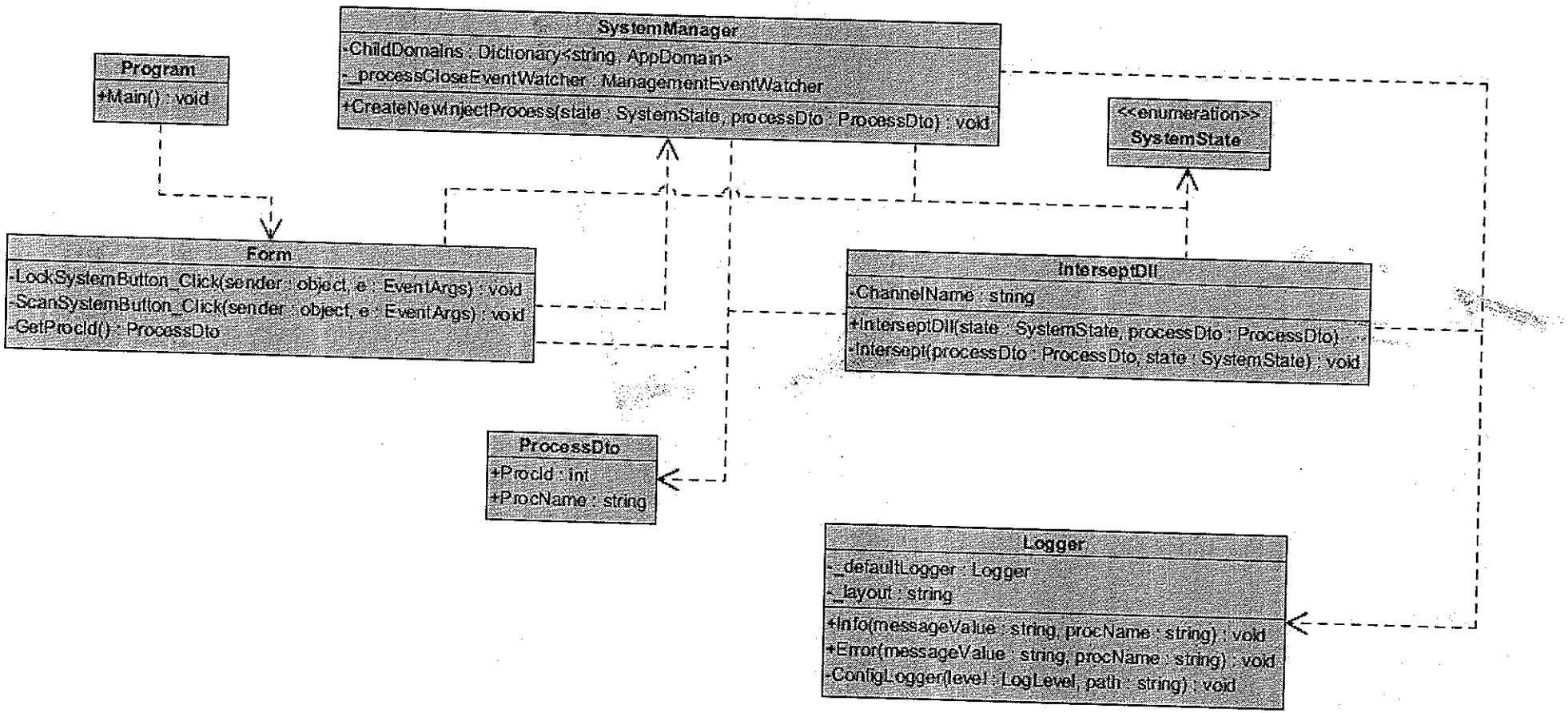

Рис 6. Диаграмма классов уровня перехвата и инжектирования

Разберем каждый класс подробнее.

Класс Program является точкой входа приложения. В нем создается класс Form, которое реализует главное окно программы. В классе Form события LockSystemButton_Click и ScanSystemButton_Click инициализируют инжектирование в выбранный процесс поведение сбора сведений и активной защиты соответственно.

Класс SystemManger - статический класс, который отвечает за инжектирование выбранного поведения в нужный процесс. Для каждого такого инжектирование система создает отдельный домен приложения, который хранится в словаре ChildDomains ключами которого являются имена процессов. Поле _processCloseEventWotcher следит за событиями закрытия всех процессов компьютера и если в такой процесс инжектировано какое-либо поведение, дочерний домен с поведением выгружается.

InterceptDll – класс, находящийся в отдельной сборке, которая в свою очередь загружается в дочерний домен приложения и реализует инжектирование нужного поведения.

Класс Loger – статический класс, который осуществляет логирование системных сообщений, а также сообщений о несанкционированном доступе перехватываемой WIN API функцией по ключу не состоящем в белом списке. В классе есть два метода – Info и Error. Метод Info записывает системные сообщения или же сообщения о несанкционированном доступе. Метод Error записывает сообщения о системных ошибках. Метод ConfigLogger задает настройки логгера и путь записи лог файла. Для каждого процесса создается отдельная папка с названием равным имени процесса, в которой хранятся файлы логов и белые списки.

Класс ProcessDto создает объекты для передачи данных о процессе между классами и уровнями приложения, но не содержащими какого-либо поведения.

Перечисление SystemState нужно для передачи информации о состоянии системы между классами и уровнями приложения.

Рассмотрим уровень поведения перехватываемых WIN API функций в выбранном процессе. Диаграмма классов этого уровня показана на рисунке 7.

Рассмотрим каждый класс подробнее.

Точка входа является класс Main, в который из внешнего домена приложения приходит класс BehaviorsWrapper, который содержит список всех поведений, перехватываемых WIN API функций, соответствующих текущему режиму системы.

Основные классы данного уровня являются абстрактный класс FunctionBehavior и все его производные классы, абстрактный класс FunctionInjected и все его производные классы, а также атрибут AttachedTypeAttribute, который связывается с классом поведения и хранит

в себе тип создаваемого объекта наследника класса FunctionInjected, который в свою очередь реализовывает перехват определенной API функции.

Итак, рассмотрим архитектуру модуля активной защиты в разрезе двух поведений системы - режим сбора сведений и режим защиты.

Класс FunctionBehavior наследуется абстрактным классом FunctionBehaviorForXml от которого уже наследуются поведения двух API функций – DeleteFile и CreateFile описанных классами DeleteFileFunctionBehaviorForXml и CreateFileFunctionBehaviorForXml соответственно. От абстрактного класса FunctionBehaviorForXml наследуются классы, реализующие поведение сбора сведений и составления белых списков. Для добавления новой функции для перехвата с таким же типом поведения сбора сведений нужно просто наследоваться от класса FunctionBehaviorForXml. Класс XmlLoggerManager – статический класс, реализующий создание белых списков и сверения данных в нем. Так же используется описанный выше класс Logger для логирования системных сообщений и ошибок.

Поведение активной защиты система реализовывает с помощью класса FunctionBehaviorForNLog от которого так же наследуются классы поведений отдельных функций – тех, же, которые были описанны выше. Для проверки доступа API функции к ключу белого списка используется класс FunctionBehaviorForXml, описанный выше.

Как и говорилось выше, классы поведений API функций связанны с классами, реализующими их перехват с помощью атрибута AttachedTypeAttribute. Создание класса перехвата функции для определенного поведения этой функции реализовано с помощью метода расширения CreateAttachedTypeOfFunctionInjected.

Класс ProcessDto, как и на уровне перехвата и инжектирования является объектом, передающим информацию о процессе, но не реализующим логику.

Как уже говорилось выше, абстрактный класс FunctionInjected является базовым для всех классов, реализующих перехват WIN API функций. Для добавления в систему новой API функции, как и в случае с классами поведения, нужно так же добавить наследуемый от FunctionInjected, класс, реализующий перехват новой функции.

Как и говорилось выше, точка входа является класс Main, в который из внешнего домена приложения приходит класс BehaviorsWrapper, который содержит список всех поведений, перехватываемых API функций, соответствующих текущему режиму системы. Модуль просто пробегает по всем классам поведения наследованным от FunctionBehavior и с помощью метода расширения CreateAttachedTypeOfFunctionInjected создает объект класса, наследуемого от FunctionInjected, тип которого хранится в атрибуте AttachedTypeAttribute класса данного поведения.

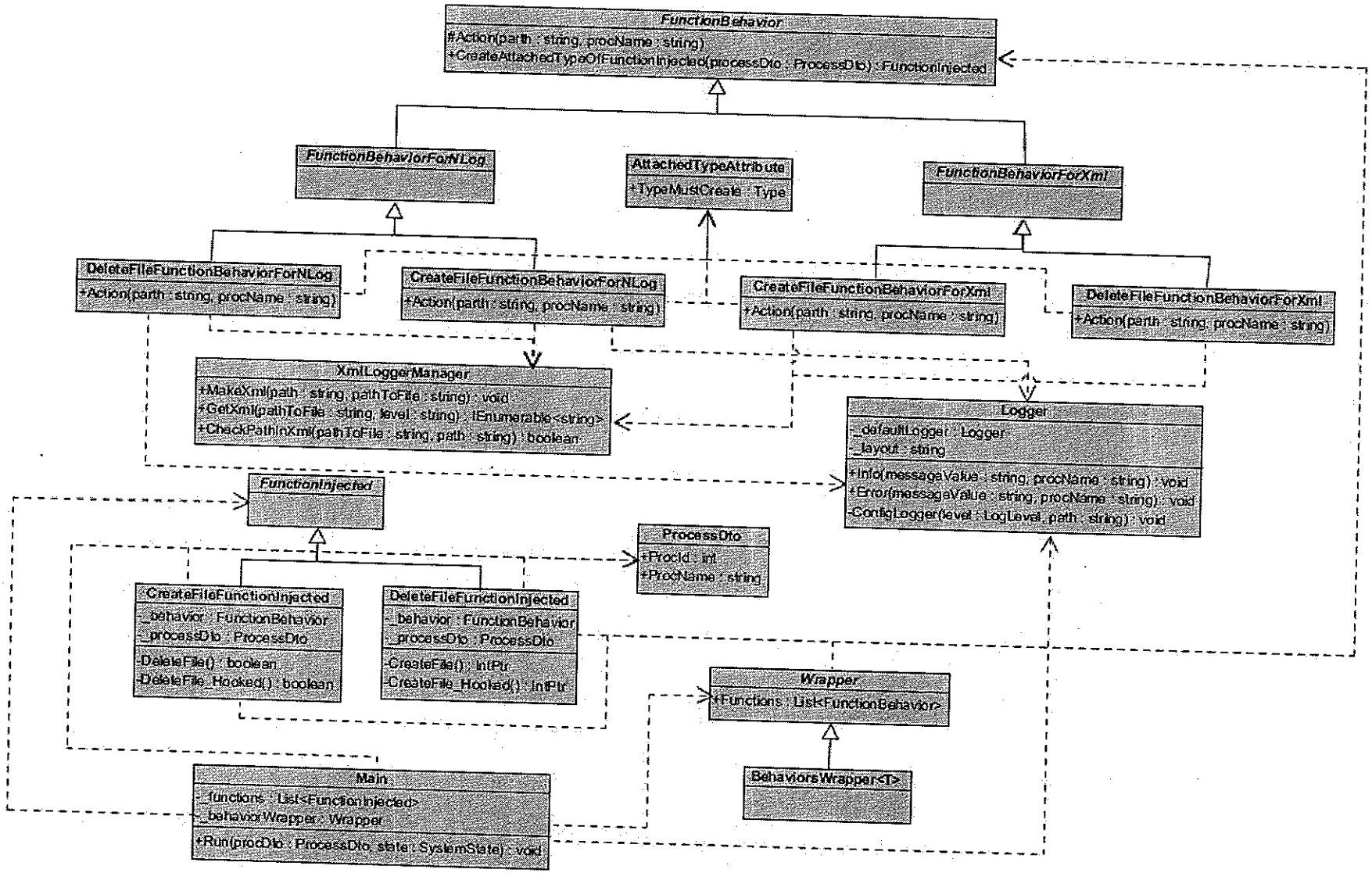

Рис 7. Диаграмма классов поведения перехватываемых API функций в выбранном процессе

Для того, чтобы программист смог добавить новую API функцию в модуль для перехвата, нужно создать класс, наследуемый от абстрактного класса FunctionInjected. В этом классе нужно определить делегат, который должен называться так же, как и перехватываемая функция, количество аргументов делегата и их типы должны соответствовать перехватываемой функции. Этому делегату должен быть приставлен атрибут DllImport, который берет ссылку на функцию из библиотеки, в которой находится перехватываемая API функция. Так же нужно реализовать функцию, которая будет воспроизводить заданное поведение. Так же эта функция должна иметь точно такой набор аргументов, что и перехватываемая функция. С помощью библиотеки EasyHook, в конструкторе заменяем ссылку на перехватываемую WIN API функцию на описанную выше функцию, реализующую заданное поведение. Чтобы добавить нужное поведение, реализованное в подмененных функциях, нужно реализовать класс, наследуемый от абстрактного класса FunctionBehavior. Так как в нашем модуле предполагается два поведения перехватываемых функций, то от абстрактного класса FunctionBehavior уже отнаследованы классы-маркеры FunctionBehaviorForNlog и FunctionBehaviorGorXml – для поведения сбора сведений и поведения защиты соответственно. В связи с этим для добавления нужного поведения нужно отнаследоваться от соответствующего класса-маркера и реализовать класс поведения для конкретной WIN API функции, либо создать отдельный вектор поведения и наследовать уже конкретное поведение конкретных функций от него. Каждый конкретный класс поведения определенной api функции должен иметь атрибут AttachedTypeAttribute, который хранит в себе класс-наследник FunctionInjected для соответствующей WIN API функции.

Для примера, если добавить в модуль новую WIN API функцию RegCreateKeyEx для перехвата, то диаграмма классов поведения перехватываемых WIN API функций в выбранном процессе будет выглядеть как на рисунке 8.

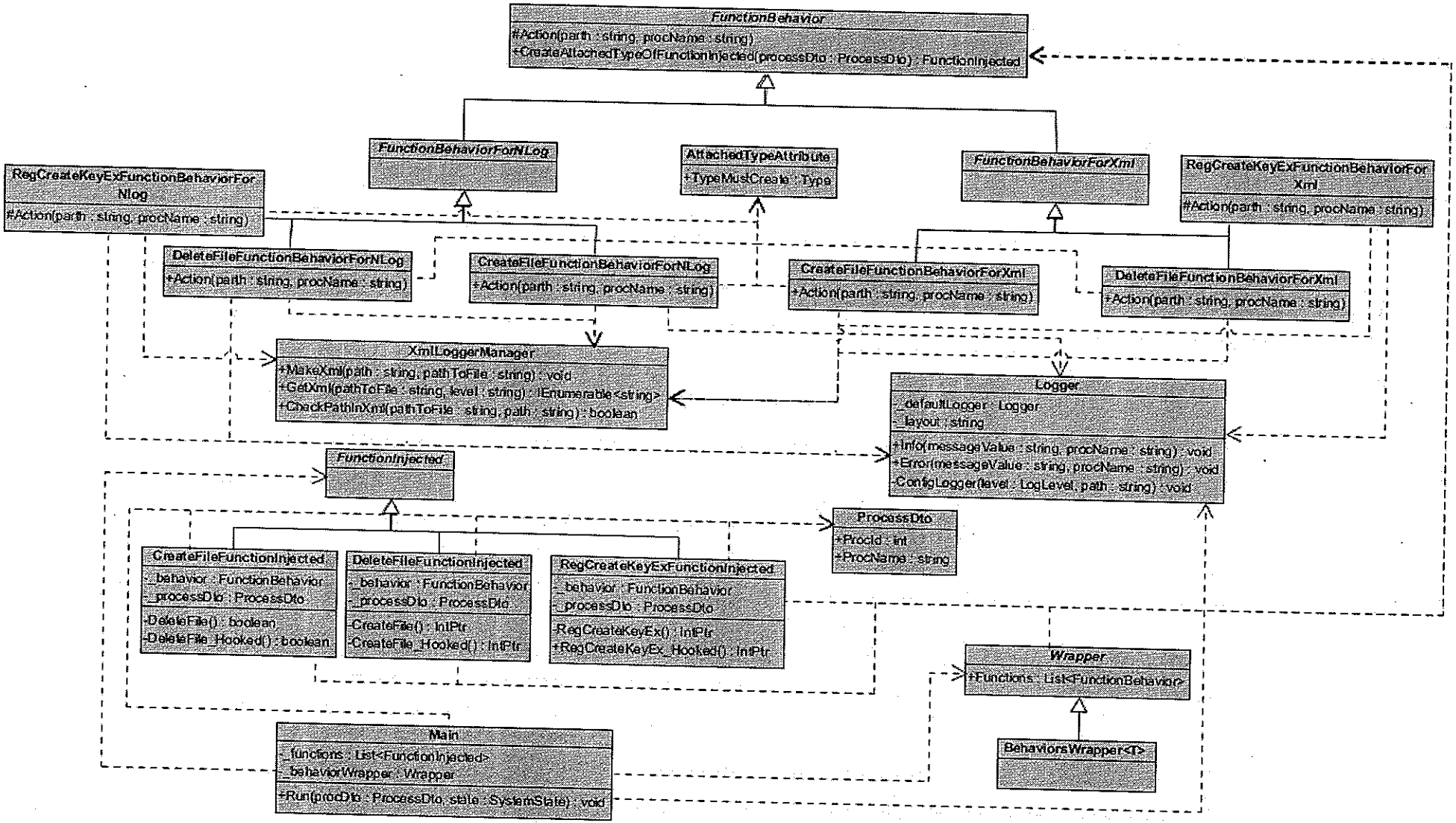

Рис 8. Диаграмма классов поведения с добавленной API функцией

2.3. Алгоритмы работы модуля

Общий алгоритм работы главного окна

При запуске модуля активной защиты появляется окно модуля отображающая все процессы, запущенные на компьютере. Далее модуль ожидает выбора пользователя каких-либо существующих элементов управления.

1. При выборе пользовательского элемента управления «Активировать сбор сведений» происходит инжектирование поведения, реализующее сбор сведений в перехватываемых функциях процесса, выбранного в списке всех запущенных на компьютере процессов.
2. При выборе пользовательского элемента управления «Активировать защиту» происходит инжектирование поведения, реализующее проверку посещаемых перехватываемыми WIN API функциями директорий, ключей реестра и т.д. по белому списку, составленному в режиме программы сбора сведений.
3. При выборе «выход» приложение закрывается.

Общий алгоритм работы главного окна представлен на рисунке 9.

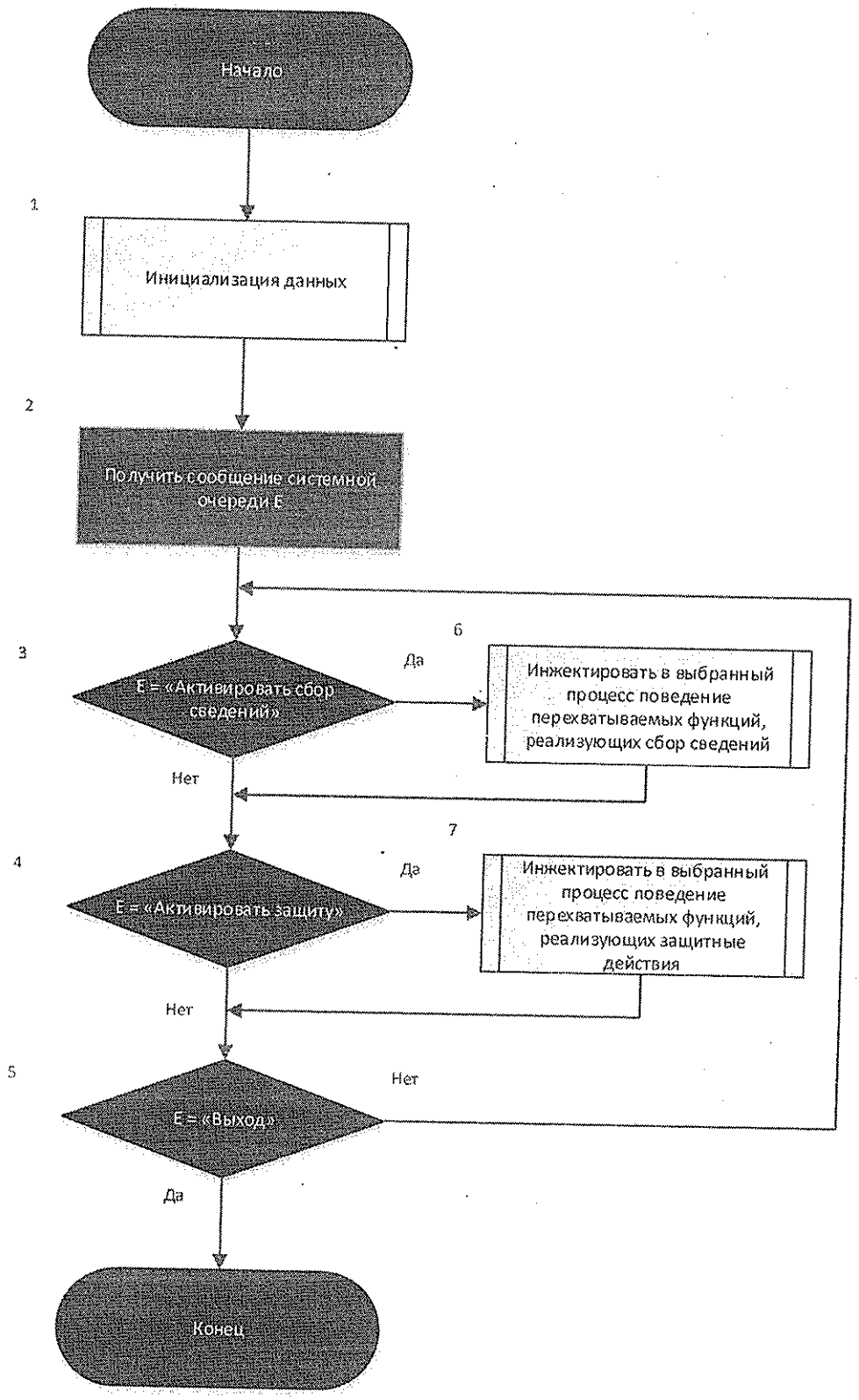

Рис 9. Общий алгоритм главного окна приложения

Алгоритм создания дочернего домена

При инжектировании поведения в процесс происходит создание нового дочернего домена приложения и помещения в этот домен сборки, реализующей инжектирование выбранного поведения в заданный процесс. Если же дочерний домен уже существует, то он выгружается и в него заново подгружается сборка, реализующая инжектирование.

Алгоритм создания дочернего домена показан на рисунке 10.

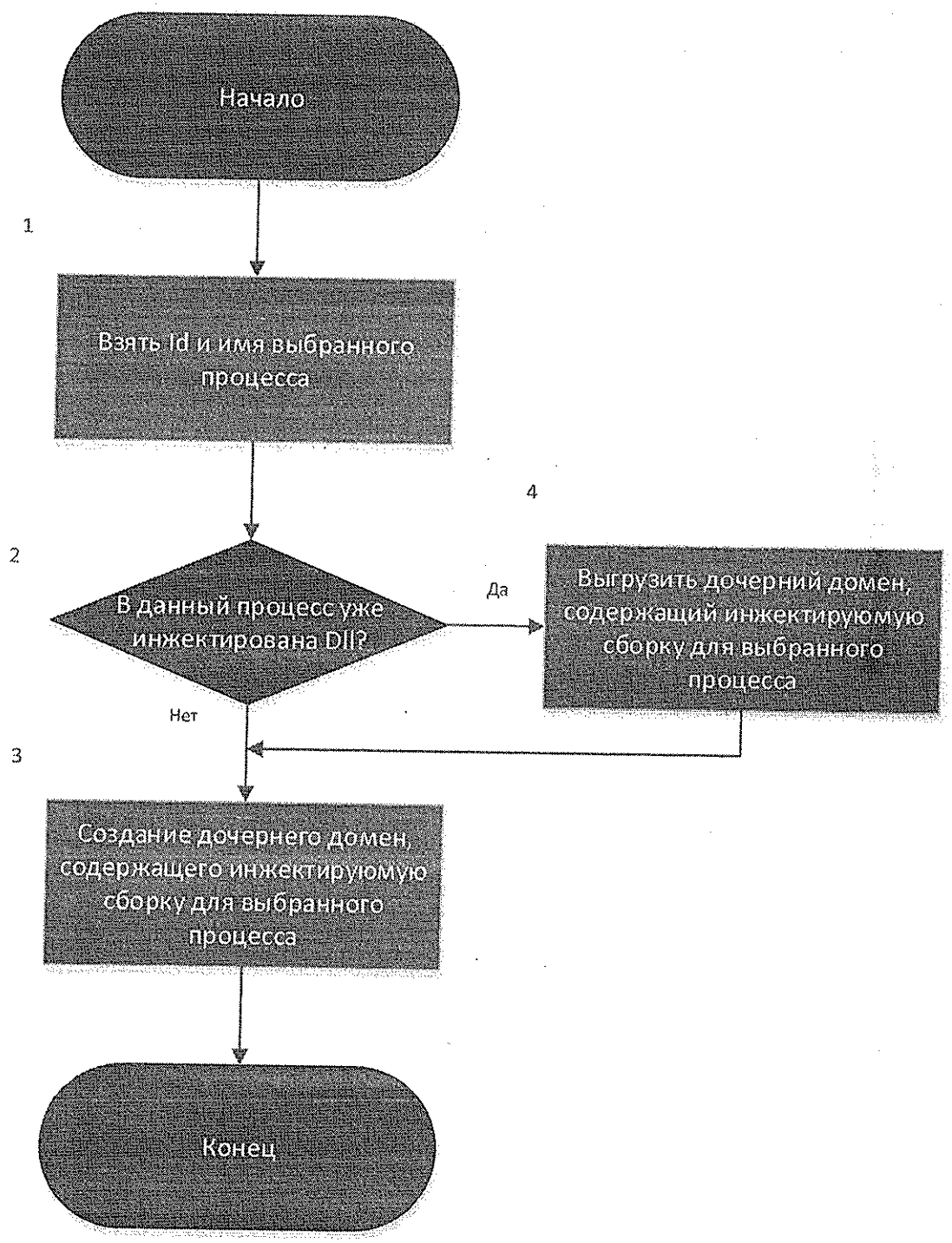

Рис 10. Алгоритм создания дочернего домена

2.4. Интерфейс главного окна и структура файлов логов и белого списка

После запуска модуля активной защиты, появляется главное окно со списком всех запущенных процессов на рабочей машине и двумя пользовательскими кнопками активирующие инжектирование либо поведения сбора сведений, либо защиты.

Интерфейс главной страницы представлен на рисунке 11.

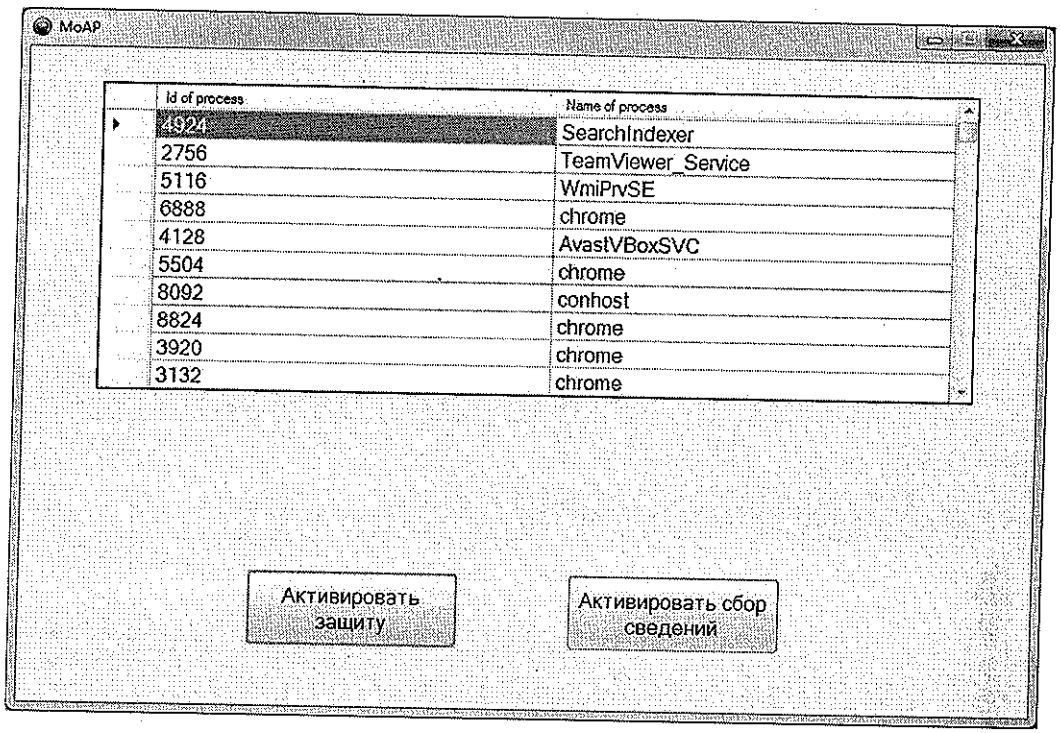

Рис 11. Интерфейс главной страницы модуля активной защиты

При инжектировании любого поведения системы в выбранный процесс на диске на диске С появляется общая директория для всех логов - logs\LoggerModule. Для каждого процесса создается своя папка с названием имени процесса, в которой хранятся файлы логов и файл белого списка(рисунок 12). в проводнике

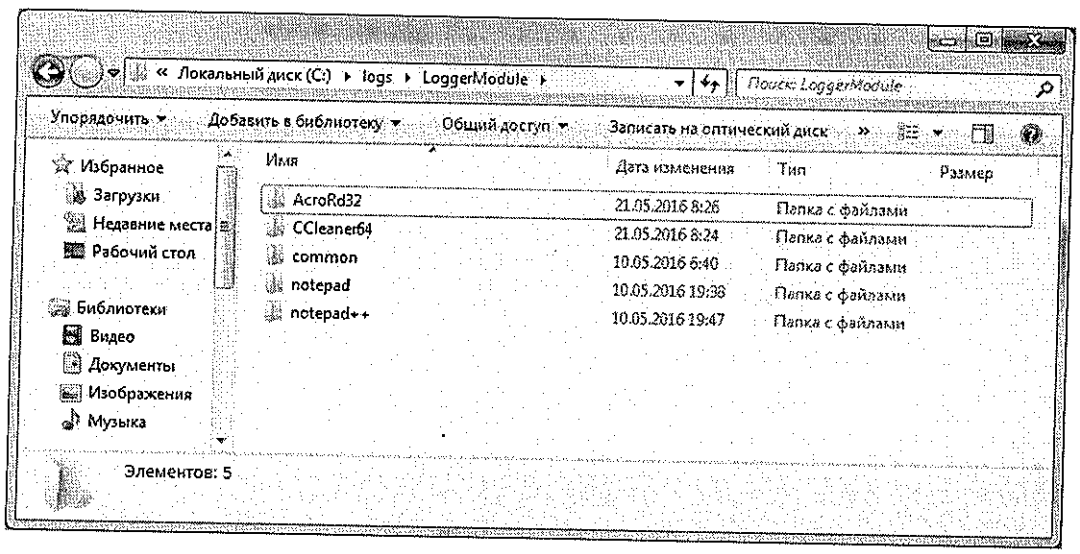

Рис 12. Структура директорий перехватываемых процессов для хранения файлов логов и белых списков

В каждой папке хранится файл логов Info.txt для системных сообщений и перехваченных не валидных директорий. Файл Info.txt представлен на рисунке 13. Сообщений может быть два вида: системное сообщение и лог несанкционированного доступа к ключам белого списка. Форма второго вида сообщений такая: {Имя перехваченной WIN API функции} function call in {ключ белого списка, по которому функция пыталась получить доступ}

Так же для учета системных ошибок создается файл Error.txt. В нем записываются системные ошибки, возникшие в модуле. Файл белого списка является xml файлом с именем test.xml, он представлен на рисунке 14. Для каждой перехватываемой WIN API функции создается свой файл профиля активности так как каждая перехватываемая функция имеет определенный ключ белого списка. Структура файла профиля активности выглядит следующим образом: существует три вида узлов: root, levl{#} и add.Узел root является единственным узлом и представляет корень документа. Дочерний к узлу root узел levl{#} отвечает за хранение ключей равной длинны. Символ # в наименовании узла levl{#} является длиной директории. Узел add дочерний к узлу levl{#}хранит в себе сам ключ длины, указанного в наименовании levl. Такая структура была предложена для более быстрого считывания ключей из файла профиля активности.

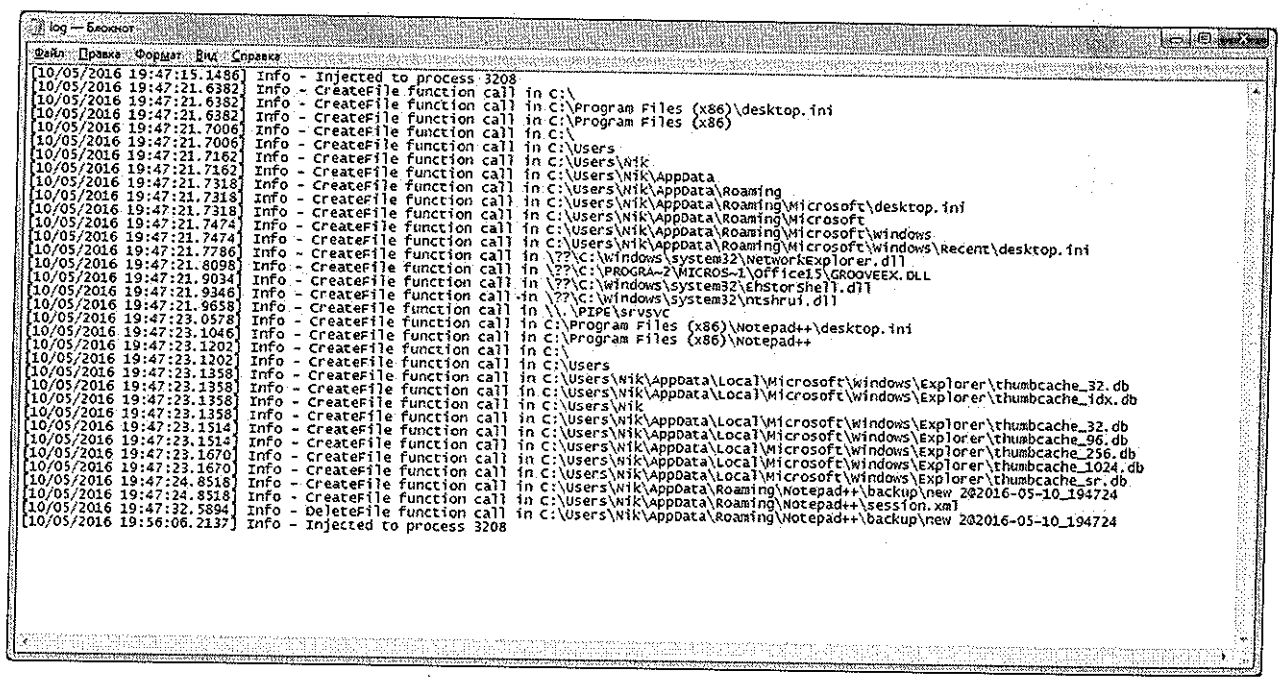

Рис 13. Файл Info.txt

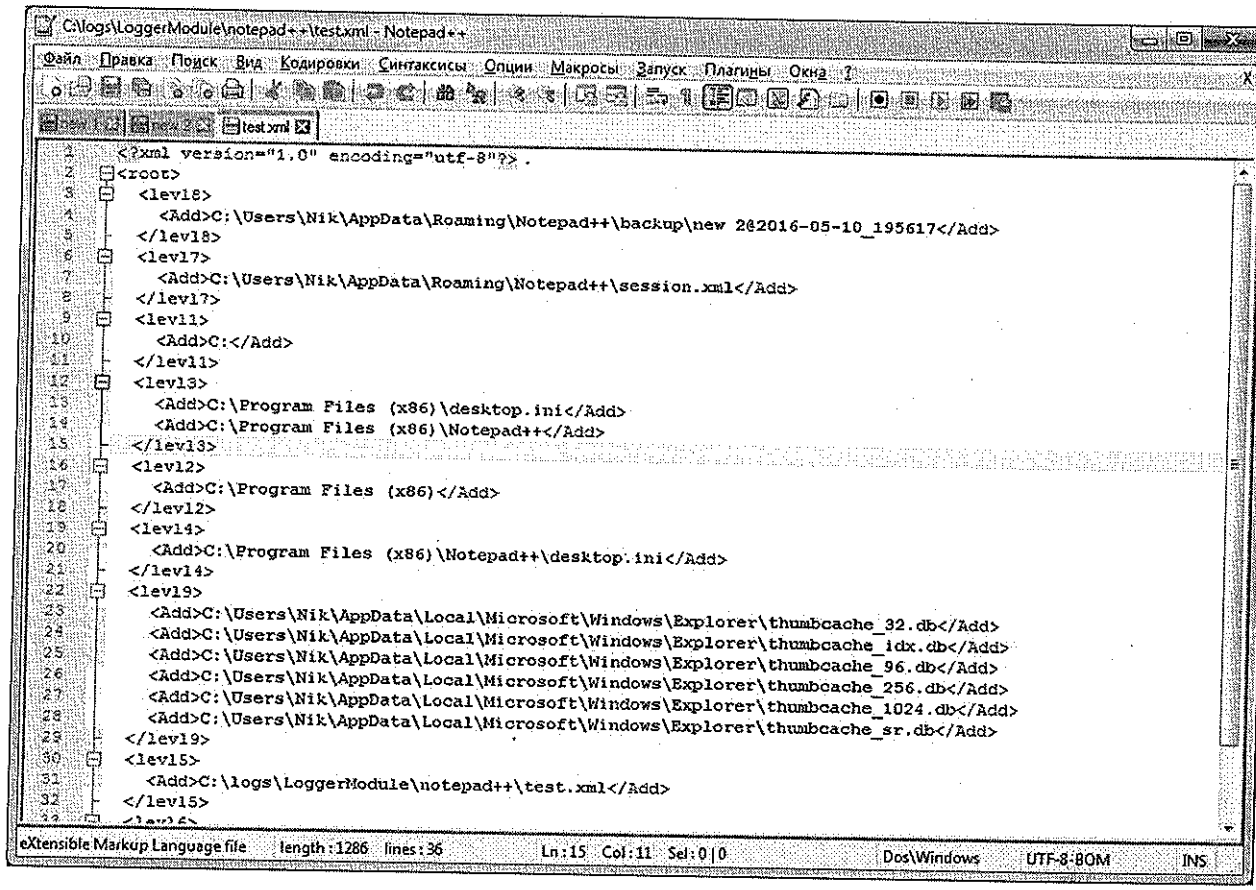

Рис 14. Структура файла белого списка

3. Выводы

В результате проделанной работы было рассмотрено топ 10 самых часто детектируемых вирусов из чего был сделан вывод, что после проникновения в систему программы в основном взаимодействуют с разделами реестра SYSTEM и SOFTWARE, так же они копируют себя в системную директорию или папку AppData, WinDir, Temp. Так же вредоносные программы работают с различными исполняемыми или веб файлами, а также с библиотеками dll. Все это делается вредоносными программами с помощью API функции платформы Windows.

На основе этого вывода для разработки модуля была выбрана технология защиты на основе поведенческого анализа и инжектирования dll и разработан модуль активной защиты. Архитектура модуля предполагает простую поддержку приложения и добавление новых API функции для перехвата и нового поведения для инжектирования.

В разработанном модуле существует два режима работы – режим сбора сведений и режим защиты. В соответствии с этим в выбранные процессы для режима работы сбора сведений, в перехватываемые WIN API функции процесса инжектируется поведение сбора сведений о посещаемых процессом ключей белого списка(директорий, ключей реестра, IP-адресов и т.д.) перехватываемыми WIN API функциями и добавление этих ключей в файл белого списка. Для режима работы защиты модуль инжектирует поведение защиты в перехватываемые WIN API функции выбранного процесса. Поведение защиты предполагает под собой сверку ключей, перехватываемых WIN API функций с файлом белого списка или профиля активности. Если проверяемого ключа нет в файле профиля активности, тогда он считается не валидным и заноситься в логи модуля, а сама WIN API функция прекращает действие по этому ключу.

Так как модуль активной защиты построен на технологии поведенческой защиты, инжектировании dll и созданию белых списков, можно сказать, что процент покрытия защитой модуля от уже известных вирусов тем больше, чем больше WIN API функций перехватываются модулем. Так как в системе Windows имеется большое количество WIN API функций, в разработанном модуле реализован перехват нескольких WIN API функций в демонстративных целях.

Так же существует риск взлома разработанного модуля – злоумышленник может представиться администратором и добавить в файл белого списка ключ, по которому злоумышленнику потребуется приобрести доступ. Так же сама система может быть декомпилирована, изменена и заново собрана.

Заключение

В работе было рассмотрено десять самых часто детектируемых вирусов по данным лаборатории Касперского. Анализ их поведения показал, что после проникновения в систему, они в основном взаимодействуют с разделами реестра, копируют себя в системную директории AppData, WinDir, Temp. Для распространения вирусы используют неизвестные уязвимости программного обеспечения.

Были рассмотрены методы защиты против вирусных угроз. По результатам сравнительного анализа был сделан вывод, что проактивная защита является самой простой в реализации, позволяет обнаруживать неизвестные вирусы, а также защищать программное обеспечение, имеющее уязвимости нулевого дня.

В качестве технологии проактивной защиты было выбрано «инжектирование dll», которая обеспечивает перехват выбранных Win Api функций и контроль доступа к ним.

Разработан прототип приложения, реализующий «инжектирование dll» и обеспечивающее составление профиля активности приложения, а также защиту системы при заданном профиле активности. Система была проанализирована с точки зрения безопасности, протестирована, исходные коды выложены в открытый доступ на GitHub: https://github.com/NikArtes/Modul-of-active-protection

Библиографический список

1. Артес Н.О. Сравнительный Анализ Взаимодействия Разных Видов Вредоносных Программ На Систему «Windows» / Н.О. Артес, С.М. Елсаков// Сборник трудов II всероссийской научно-практической конференции. Под редакцией Ю.М. Ковалева. – Издательский центр ЮУрГУ, 2015. – С. 11 – 18.

2. Журнал «Хакер». Тест бесплатных проактивок: проверяем free-версии AVAST, AVIRA, AVG, COMODO, CLAMAV. – https://xakep.ru/2011/05/14/57049/

3. Хайретдинов, Р. Робо-страж: защита интернет-банка по Кевину Костнеру/ Р. Хайретдинов // BIS Journal - Информационная безопасность банков. – 2016. - № 2(21). – С. 125-126.

4. Лунтовский, Г. Стратегия снижения рисков информационной безопасности в кредитно-финансовой сфере/ Л. Георгий// BIS Journal - Информационная безопасность банков. – 2016. - № 2(21). – С. 120-124.

5. Зайцев, В. Информационная безопасность и IT-ландшафт в банке: не жить друг без друга/ В. Зайцев// BIS Journal - Информационная безопасность банков. – 2016. - № 1(20). – С. 82-89.

6. Джандоменико, Э. Методы проникновения в корпоративные сети/ Э. Джандоменико // Директор Безопасности. – 2016. - №1 (январь). – С. 5-9.

7. Threat description. Technical details and removal instructions for malicious threats identified by Labs – https://www.f-secure.com/en/web/labs_global/threat-descriptions#page-2

8. CTB-Locker Infections on the Rise – https://www.f-secure.com/weblog/archives/00002788.html

9. TROJAN-SPY:W32/ZBOT – https://www.f-secure.com/v-descs/trojan-spy_w32_zbot.shtml

10. Threat description Criptolocker – https://www.f-secure.com/v-descs/trojan_w32_cryptolocker.shtml

11. Описание сетевого червя Net-Worm.Win32.Kido.ih – https://w.securelist.com/ru/descriptions/iframe/Net-Worm.Win32.Kido.ih

12. Описание интернет червя Email-Worm.Win32.Runouce. – https://w.securelist.com/ru/descriptions/iframe/Email-Worm.Win32.Runouce.b

13. Описание червя Email-Worm.Win32.Blare – https://w.securelist.com/ru/descriptions/iframe/Email-Worm.Win32.Blare

14. Описание троянской программы Trojan-Clicker.Win32.Mobs – https://w.securelist.com/ru/descriptions/iframe/Trojan-Clicker.Win32.Mobs

15. Threat description Trojan:W32/Dllpatcher – https://www.f-secure.com/v-descs/trojan_w32_dllpatcher.shtml

16. Описание троянской программы Trojan-Spy.Win32.Stonari – https://w.securelist.com/ru/descriptions/iframe/Trojan-Spy.Win32.Stonari

17. Официальный сайт лаборатории Касперского. Классификация вредоносных программ – http://www.kaspersky.ru/internet-security-center/threats/malware-classifications

18. Официальный сайт лаборатории Касперского. Что такое ботнет? - http://www.kaspersky.ru/internet-security-center/threats/botnet-attacks

19. Официальный сайт лаборатории Касперского. Какие условия нужны для распространения вредоносных программ – http://www.kaspersky.ru/internet-security-center/threats/hacking-system-vulnerabilities

20. Официальный сайт лаборатории Касперского. Выбор антивирусного решения – http://www.kaspersky.ru/internet-security-center/internet-safety/antivirus-choices

21. Официальный сайт лаборатории Касперского. Обнаружение вредоносных программ и эксплойтов – http://www.kaspersky.ru/internet-security-center/internet-safety/antivirus-malware-detection

22. Журнал «Хакер». Новый оборонительный рубеж: обзор популярных систем отражения локальных угроз – https://xakep.ru/2009/12/22/50612/

23. Черкас, Ю. Востребованные решения по защите ИТ-инфраструктуры/ Ю. Черкас // InformationSecurity - http://www.itsec.ru/articles2/Oborandteh/vostrebovannye-resheniya-po-zaschite-it-infrastruktury

24. Уваров, А.С. Тестируем проактивную защиту популярных антивирусов/ А.С. Уваров // Записки IT специалиста - http://interface31.ru/tech_it/2013/01/testiruem-proaktivnuyu-zashhitu-populyarnyh-antivirusov.html

25. Касперски, К. Компьютерные вирусы изнутри и снаружи/ К. Касперски. – СПб.: Питер, 2012. – 527с.

26. Искусство взлома и защиты систем / Дж. Козиол, Д. Личфилд, Д. Эйтэл и др. – СПб.: Питер, 2011. – 416 с.

27. Программно-аппаратные средства защиты компьютерной информации / сост. Е. И. Духан, Н. И. Синадский, Д. А. Хорьков. – Екатеринбург.: УрГУ, 2008. – 240 с.

28. Зайцев, А.П. Технические средства и методы защиты информации / А.П. Зайцев, А.А. Шелупанов, Р.В Мещеряков. - М.: ООО «Издательство Машиностроение», 2009. – 508 с.

29. Шаньгин, В. Ф. Информационная безопасность компьютерных систем и сетей / В.Ф. Шаньгин – М.: ИД «ФОРУМ»: ИНФРА-М, 2011. — 416 с.

ПРИЛОЖЕНИЕ 1

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет математики, механики и компьютерных наук
Кафедра дифференциальных и стохастических уравнений

РАЗРАБОТКА СИСТЕМЫ ЗАЩИТЫ ПРИЛОЖЕНИЙ НА ОСНОВЕ ПРОФИЛЯ АКТИВНОСТИ

ТЕКСТ ПРОГРАММЫ
ЮУрГУ – 09.04.04.2016.129-125.ВКР

Руководитель, канд. физ.-мат.наук,
доцент
_____ /С.М. Елсаков/
«___» _____2016 г.

Автор, студент группы ММиКН-293
_____/Н.О. Артес/
«30» мая 2016 г.

Нормоконтролер, канд. физ.-мат. наук,
доцент
_____/М.А. Сагадеева/
«___» _____2016 г.

Челябинск 2016

УДК - 4.056.5

Артес Н.О.

Исходный код системы защиты приложений на основе профиля активности. /Н.О. Артес. – Челябинск, 2016. - 37 с.

Прототип модуля реализован с помощью MS Visual Studio 2015 Community с использованием .Net Framework 4.6 на языке C#.

ОГЛАВЛЕНИЕ

П1.1 Исходный текст файла «ProcessDto.cs»

```
using System;

namespace Core.Dtos
{
    [Serializable]
    public class ProcessDto
    {
        public int ProcId { get; set; }

        public string ProcName { get; set; }
    }
}
```

П1.2 Исходный текст файла «XDocumentExtentions.cs»

```
using System.IO;
using System.Text;
using System.Xml;
using System.Xml.Linq;

namespace Core.Extentions
{
    public static class XDocumentExtentions
    {
        public static XDocument LoadOrCreate(this XDocument xDocument, string
pathToFile)
        {
            if (!File.Exists(pathToFile))
            {
                using (var xmlTextWriter = new XmlTextWriter(pathToFile,
Encoding.UTF8))
                {
                    xmlTextWriter.WriteStartElement("root");
                    xmlTextWriter.WriteEndElement();
                }
            }

            xDocument = XDocument.Load(pathToFile);

            return xDocument;
        }
    }
}
```

П1.3 Исходный текст файла «AppConfigManager.cs»

```
using System.Configuration;

namespace Core.Managers
{
    public static class AppConfigManager
    {
        private const string _partOfWhiteListFile = "WhiteList_For_";

        public static string GetBasePathToLoggerModule()
        {
            return
ConfigurationManager.AppSettings["basePathToLoggerModule"];
        }

        public static string GetNameOfLogFile()
        {
            return ConfigurationManager.AppSettings["nameOfLogFile"];
        }

        public static string GetNameOfErrorFile()
        {
            return ConfigurationManager.AppSettings["nameOfErrorFile"];
        }
        public static string GetPathToLogForProcess(string processName =
"common")
        {
            return string.Concat(GetBasePathToLoggerModule(),
                                 processName,
                                 "\\",
                                 GetNameOfLogFile());
        }

        public static string GetPathToErrorForProcess(string processName =
"common")
        {
            return string.Concat(GetBasePathToLoggerModule(),
                                 processName,
                                 "\\",
                                 GetNameOfErrorFile());
        }

        public static string GetPathToXmlForProcess(string processName =
"common", string functionName = "")
        {
            return string.Concat(GetBasePathToLoggerModule(),
                                 processName,
                                 "\\",
                                 _partOfWhiteListFile,
                                 functionName,
                                 ".xml");
        }
    }
}
```

П1.4 Исходный текст файла «Logger.cs»

```
using System;
using System.IO;
using NLog;
using NLog.Config;
using NLog.Targets;

namespace Core.Managers
{
    public static class Logger
    {
        private static readonly NLog.Logger _defaultLogger =
LogManager.GetCurrentClassLogger();

        private const string _layout = "[${date:format=dd/MM/yyyy
HH\\:mm\\:ss.ffff}] ${level} -
${message}${onexception:${newline}${exception:format=ToString}${newline}${sta
cktrace:topFrames=10}}";

        public static void Info(string messageValue, string procName)
        {
            ConfigLogger(LogLevel.Info,
AppConfigManager.GetPathToLogForProcess(procName));
            _defaultLogger.Info(messageValue);
        }

        public static void Error(string messageValue, string procName)
        {
            ConfigLogger(LogLevel.Error,
AppConfigManager.GetPathToErrorForProcess(procName));
            _defaultLogger.Error(messageValue);
        }

        private static void ConfigLogger(LogLevel level, string path)
        {
            var config = new LoggingConfiguration();
            var fileTarget = new FileTarget();
            config.AddTarget("file", fileTarget);
            fileTarget.FileName = Path.Combine(path);
            fileTarget.Layout = _layout;
            fileTarget.CreateDirs = true;
            fileTarget.KeepFileOpen = true;
            config.LoggingRules.Add(new LoggingRule("*", level, fileTarget));
            LogManager.Configuration = config;
        }
    }
}
```

П1.5 Исходный текст файла «SystemManager.cs»

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Management;
using System.Reflection;
using System.Text.RegularExpressions;
using Core.Dtos;
using NLog.LayoutRenderers;

namespace Core.Managers
{
    public static class SystemManager
    {
        private static readonly Dictionary<string, AppDomain> ChildDomains;

        private static ManagementEventWatcher _processCloseEventWatcher;

        static SystemManager()
        {
            ChildDomains = new Dictionary<string, AppDomain>();

            _processCloseEventWatcher = new ManagementEventWatcher("SELECT *
FROM Win32_ProcessStopTrace");

            _processCloseEventWatcher.EventArrived += (sender, eventArgs) =>
                                                      {
                                                          var processName
=
eventArgs.NewEvent.Properties["ProcessName"].Value.ToString().Replace(".exe",
string.Empty);

                                                          if
(ChildDomains.ContainsKey(processName))
                                                          {

AppDomain.Unload(ChildDomains[processName]);

ChildDomains.Remove(processName);

Logger.Info($"process {processName} is closing and his domain was unloaded",
"common");

                                                          }
                                                      };

            _processCloseEventWatcher.Start();
        }

        public static void CreateNewInjectProcess(SystemState state,
ProcessDto processDto)
        {
            if (ChildDomains.ContainsKey(processDto.ProcName))
            {
                AppDomain.Unload(ChildDomains[processDto.ProcName]);
                ChildDomains.Remove(processDto.ProcName);
```

```
            }
            ChildDomains.Add(processDto.ProcName,
AppDomain.CreateDomain(processDto.ProcName));


ChildDomains[processDto.ProcName].CreateInstanceFromAndUnwrap(AppDomain.Curre
ntDomain.BaseDirectory + "InterceptedModule.dll",
"InterceptedModule.InterseptDll", false, BindingFlags.CreateInstance, null,
new[] {(object) state, (object) processDto}, CultureInfo.InvariantCulture,
null);
        }
    }
}
```

П1.6 Исходный текст файла «XmlLoggerManager.cs»

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Linq;
using Core.Extentions;

namespace Core.Managers
{
    public static class XmlLoggerManager
    {
        public static void MakeXml(string path, string pathToFile)
        {
            XDocument doc = new XDocument().LoadOrCreate(pathToFile);

            var vlogennost = string.Concat("levl", path.Split('\\').Length);
            if (doc.Root.Element(vlogennost) == null)
            {
                doc.Root.Add(new XElement(vlogennost, new XElement("Add",
path)));
            }
            else
            {
                if (doc.Root.Element(vlogennost).Elements().Any() &&
!doc.Root.Element(vlogennost).Elements().Select(x => x.Value).Contains(path))
                {
                    doc.Root.Element(vlogennost).Add(new XElement("Add",
path));
                }
            }

            doc.Save(pathToFile);
        }

        public static IEnumerable<string> GetXml(string pathToFile, string
level)
        {
            var result = new List<string>();

            var xDocument = new XDocument().LoadOrCreate(pathToFile);
            if (xDocument.Root != null && xDocument.Root.Element(level) !=
null)
            {
                var xElements =
xDocument.Root.Element(level).Elements("Add");
                if (xElements.Any())
                {
                    result = xElements.Select(x =>
x.Value.ToString()).ToList();
                }
            }

            return result;
        }
```

```
        public static bool CheckPathInXml(string pathToFile, string path)
        {
            var enumerable = GetXml(pathToFile, string.Concat("levl",
path.Split('\\').Length));

            return enumerable.Contains(path);
        }
    }
}
```

П1.7 Исходный текст файла «AttachedTypeAttribute.cs»

```
using System;

namespace Core
{
    public class AttachedTypeAttribute : Attribute
    {
        public Type TypeMustCreate { get; }

        public AttachedTypeAttribute(Type typeMustCreate)
        {
            TypeMustCreate = typeMustCreate;
        }
    }
}
```

П1.8 Исходный текст файла «SystemState.cs»

```
namespace Core
{
    public enum SystemState
    {
        Scanning,
        Locking
    }
}
```

П1.9 Исходный текст файла «InterseptDll.cs»

```
using System;
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Runtime.Remoting;
using EasyHook;
using System.Security.Principal;
using LibraryInjected.FunctionBehaviors;
using LibraryInjected.Wrappers;
using Core;
using Core.Dtos;
using Core.Managers;

namespace InterceptedModule
{
    [Serializable]
    public class InterseptDll
    {
        private string ChannelName;

        public InterseptDll(SystemState state, ProcessDto processDto)
        {
            if (processDto.ProcId == -1)
            {
                Logger.Error("No process exists with that name!",
processDto.ProcName);
                return;
            }
            Intersept(processDto, state);
        }

        private void Intersept(ProcessDto processDto, SystemState state)
        {
            try
            {
                switch (state)
                {
                    case SystemState.Scanning:

RemoteHooking.IpcCreateServer<BehaviorsWrapper<FunctionBehaviorForXml>>(ref
ChannelName, WellKnownObjectMode.SingleCall, WellKnownSidType.WorldSid);
                        break;
                    case SystemState.Locking:

RemoteHooking.IpcCreateServer<BehaviorsWrapper<FunctionBehaviorForNLog>>(ref
ChannelName, WellKnownObjectMode.SingleCall, WellKnownSidType.WorldSid);
                        break;
                }

                var str =
Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location),
"LibraryInjected.dll");
```

```
                RemoteHooking.Inject(processDto.ProcId, str, str,
(object)processDto, (object)state, (object)ChannelName);
                Logger.Info($"Injected to process {processDto.ProcId}",
processDto.ProcName);

            }
            catch (Exception ex)
            {
                Logger.Error($"There was an error while connecting to
target:\r\n{(object) ex.ToString()}", processDto.ProcName);
            }
        }

    }
}
```

П1.10 Исходный текст файла «FunctionBehavior.cs»

```
using System;

namespace LibraryInjected.FunctionBehaviors
{
    public abstract class FunctionBehavior : MarshalByRefObject
    {
        protected string _functionName;

        public abstract void Action(string keyForWhiteList, string procName);
    }
}
```

П1.11 Исходный текст файла «FunctionBehaviorForNLog.cs»

```
namespace LibraryInjected.FunctionBehaviors
{
    public abstract class FunctionBehaviorForNLog : FunctionBehavior
    {

    }
}
```

П1.12 Исходный текст файла «FunctionBehaviorForXml.cs»

```
namespace LibraryInjected.FunctionBehaviors
{
    public abstract class FunctionBehaviorForXml : FunctionBehavior
    {

    }
}
```

П1.13 Исходный текст файла «CreateFileFunctionBehaviorForNLog.cs»

```
using Core;
using Core.Managers;
using LibraryInjected.FunctionsInjected.Impl;

namespace LibraryInjected.FunctionBehaviors.Impl
{
    [AttachedType(typeof(CreateFileFunctionInjected))]
    public class CreateFileFunctionBehaviorForNLog : FunctionBehaviorForNLog
    {
        public CreateFileFunctionBehaviorForNLog()
        {
            _functionName = "CreateFile";
        }

        public override void Action(string keyForWhiteList, string procName)
        {
            if
(!XmlLoggerManager.CheckPathInXml(AppConfigManager.GetPathToXmlForProcess(pro
cName, _functionName), keyForWhiteList.Trim('\\')))
            {
                Logger.Info(string.Concat("CreateFile function call in ",
keyForWhiteList), procName);
            }
        }
    }
}
```

П1.14 Исходный текст файла «CreateFileFunctionBehaviorForXml.cs»

```
using Core;
using Core.Managers;
using LibraryInjected.FunctionsInjected.Impl;

namespace LibraryInjected.FunctionBehaviors.Impl
{
    [AttachedType(typeof(CreateFileFunctionInjected))]
    public class CreateFileFunctionBehaviorForXml : FunctionBehaviorForXml
    {
        public CreateFileFunctionBehaviorForXml()
        {
            _functionName = "CreateFile";
        }

        public override void Action(string keyForWhiteList, string procName)
        {
            XmlLoggerManager.MakeXml( keyForWhiteList.Trim('\\'),
AppConfigManager.GetPathToXmlForProcess(procName, _functionName));
        }
    }
}
```

П1.15 Исходный текст файла «DeleteFileFunctionBehaviorForNLog.cs»

```
using Core;
using Core.Managers;
using LibraryInjected.FunctionsInjected.Impl;

namespace LibraryInjected.FunctionBehaviors.Impl
{
    [AttachedType(typeof(DeleteFileFunctionInjected))]
    public class DeleteFileFunctionBehaviorForNLog : FunctionBehaviorForNLog
    {
        public DeleteFileFunctionBehaviorForNLog()
        {
            _functionName = "DeleteFile";
        }

        public override void Action(string keyForWhiteList, string procName)
        {
            if
(!XmlLoggerManager.CheckPathInXml(AppConfigManager.GetPathToXmlForProcess(pro
cName, _functionName), keyForWhiteList.Trim('\\')))
            {
                Logger.Info(string.Concat("DeleteFile function call in ",
keyForWhiteList), procName);
            }
        }
    }
}
```

П1.16 Исходный текст файла «DeleteFileFunctionBehaviorForXml.cs»

```
using Core;
using Core.Managers;
using LibraryInjected.FunctionsInjected.Impl;

namespace LibraryInjected.FunctionBehaviors.Impl
{
    [AttachedType(typeof(DeleteFileFunctionInjected))]
    public class DeleteFileFunctionBehaviorForXml : FunctionBehaviorForXml
    {
        public DeleteFileFunctionBehaviorForXml()
        {
            _functionName = "DeleteFile";
        }

        public override void Action(string keyForWhiteList, string procName)
        {
            XmlLoggerManager.MakeXml(keyForWhiteList.Trim('\\'),
AppConfigManager.GetPathToXmlForProcess(procName, _functionName));
        }
    }
}
```

П1.17 Исходный текст файла «RegOpenKeyFunctionBehaviorForNLog.cs»

```
using Core;
using Core.Managers;
using LibraryInjected.FunctionsInjected.Impl;

namespace LibraryInjected.FunctionBehaviors.Impl
{
    [AttachedType(typeof(RegOpenKeyFunctionInjected))]
    public class RegOpenKeyFunctionBehaviorForNLog : FunctionBehaviorForNLog
    {
        public RegOpenKeyFunctionBehaviorForNLog()
        {
            _functionName = "RegOpenKey";
        }

        public override void Action(string keyForWhiteList, string procName)
        {
            if
(!XmlLoggerManager.CheckPathInXml(AppConfigManager.GetPathToXmlForProcess(pro
cName, _functionName), keyForWhiteList.Trim('\\')))
            {
                Logger.Info(string.Concat("RegOpenKey function call in ",
keyForWhiteList), procName);
            }
        }
    }
}
```

П1.18 Исходный текст файла «RegOpenKeyFunctionBehaviorForXml.cs»

```
using Core;
using Core.Managers;
using LibraryInjected.FunctionsInjected.Impl;

namespace LibraryInjected.FunctionBehaviors.Impl
{
    [AttachedType(typeof(RegOpenKeyFunctionInjected))]
    public class RegOpenKeyFunctionBehaviorForXml : FunctionBehaviorForXml
    {
        public RegOpenKeyFunctionBehaviorForXml()
        {
            _functionName = "RegOpenKey";
        }

        public override void Action(string keyForWhiteList, string procName)
        {
            XmlLoggerManager.MakeXml(keyForWhiteList.Trim('\\'),
AppConfigManager.GetPathToXmlForProcess(procName, _functionName));
        }
    }
}
```

П1.19 Исходный текст файла «FunctionInjected.cs»

```
using EasyHook;

namespace LibraryInjected.FunctionsInjected
{
    public abstract class FunctionInjected
    {
        protected LocalHook _hook;
    }
}
```

П1.20 Исходный текст файла «CreateFileFunctionInjected.cs»

```
using System;
using System.Runtime.InteropServices;
using Core.Dtos;
using Core.Managers;
using EasyHook;
using LibraryInjected.FunctionBehaviors;

namespace LibraryInjected.FunctionsInjected.Impl
{
    public class CreateFileFunctionInjected : FunctionInjected
    {
        private static FunctionBehavior _behavior;

        private static ProcessDto _processDto;

        public CreateFileFunctionInjected(FunctionBehavior behavior,
ProcessDto processDto)
        {
            _behavior = behavior;

            _processDto = processDto;

            _hook = LocalHook.Create(LocalHook.GetProcAddress("kernel32.dll",
"CreateFileW"), new DCreateFile(CreateFile_Hooked), this);

            _hook.ThreadACL.SetExclusiveACL(new int[0]);
        }

        [DllImport("kernel32.dll", CharSet = CharSet.Unicode,
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
        private static extern IntPtr CreateFile(string InFileName, uint
InDesiredAccess, uint InShareMode, IntPtr InSecurityAttributes, uint
InCreationDisposition, uint InFlagsAndAttributes, IntPtr InTemplateFile);

        private static IntPtr CreateFile_Hooked(string InFileName, uint
InDesiredAccess, uint InShareMode, IntPtr InSecurityAttributes, uint
InCreationDisposition, uint InFlagsAndAttributes, IntPtr InTemplateFile)
        {
            try
            {
                _behavior.Action(InFileName, _processDto.ProcName);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, _processDto.ProcName);
            }
            return CreateFile(InFileName, InDesiredAccess, InShareMode,
InSecurityAttributes, InCreationDisposition, InFlagsAndAttributes,
InTemplateFile);
        }

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet =
CharSet.Unicode, SetLastError = true)]
```

```
        private delegate IntPtr DCreateFile(string InFileName, uint
InDesiredAccess, uint InShareMode, IntPtr InSecurityAttributes, uint
InCreationDisposition, uint InFlagsAndAttributes, IntPtr InTemplateFile);
    }
}
```

П1.21 Исходный текст файла «DeleteFileFunctionInjected.cs»

```
using System;
using System.Runtime.InteropServices;
using Core.Dtos;
using Core.Managers;
using EasyHook;
using LibraryInjected.FunctionBehaviors;

namespace LibraryInjected.FunctionsInjected.Impl
{
    public class DeleteFileFunctionInjected : FunctionInjected
    {
        private static FunctionBehavior _behavior;

        private static ProcessDto _processDto;

        public DeleteFileFunctionInjected(FunctionBehavior behavior,
ProcessDto processDto)
        {
            _behavior = behavior;

            _processDto = processDto;

            _hook = LocalHook.Create(LocalHook.GetProcAddress("kernel32.dll",
"DeleteFileW"), new DDeleteFile(DeleteFile_Hooked), this);

            _hook.ThreadACL.SetExclusiveACL(new int[0]);
        }

        [DllImport("kernel32.dll", CharSet = CharSet.Unicode,
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
        private static extern bool DeleteFile(string InFileName);

        private static bool DeleteFile_Hooked(string InFileName)
        {
            try
            {
                _behavior.Action(InFileName, _processDto.ProcName);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, _processDto.ProcName);
            }
            return DeleteFile(InFileName);
        }

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet =
CharSet.Unicode, SetLastError = true)]
        private delegate bool DDeleteFile(string InFileName);
    }
}
```

П1.22 Исходный текст файла «RegOpenKeyFunctionInjected.cs»

```
using System;
using System.Runtime.InteropServices;
using Core.Dtos;
using Core.Managers;
using EasyHook;
using LibraryInjected.FunctionBehaviors;

namespace LibraryInjected.FunctionsInjected.Impl
{
    public class RegOpenKeyFunctionInjected : FunctionInjected
    {
        private static FunctionBehavior _behavior;

        private static ProcessDto _processDto;

        public RegOpenKeyFunctionInjected(FunctionBehavior behavior,
ProcessDto processDto)
        {
            _behavior = behavior;

            _processDto = processDto;

            _hook = LocalHook.Create(LocalHook.GetProcAddress("advapi32.dll",
"RegOpenKeyW"), new
RegOpenKeyFunctionInjected.DRegOpenKey(RegOpenKey_Hooked), this);

            _hook.ThreadACL.SetExclusiveACL(new int[0]);
        }

        [DllImport("advapi32.dll", CharSet = CharSet.Auto)]
        private static extern int RegOpenKey(UIntPtr hKey, string lpSubKey,
out UIntPtr phkResult);

        private static int RegOpenKey_Hooked(UIntPtr hKey, string lpSubKey,
out UIntPtr phkResult)
        {
            try
            {
                _behavior.Action(hKey.ToString(), _processDto.ProcName);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, _processDto.ProcName);
            }
            return RegOpenKey(hKey, lpSubKey, out phkResult);
        }

        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet =
CharSet.Auto)]
        private delegate int DRegOpenKey(UIntPtr hKey, string lpSubKey, out
UIntPtr phkResult);
    }
}
```

П1.23 Исходный текст файла «Wrapper.cs»

```
using System;
using System.Collections.Generic;
using LibraryInjected.FunctionBehaviors;

namespace LibraryInjected.Wrappers
{
    public abstract class Wrapper : MarshalByRefObject
    {
        public List<FunctionBehavior> Functions { get; protected set; }
    }
}
```

П1.24 Исходный текст файла «BehaviorsWrapper.cs»

```
using System;
using System.Linq;
using System.Reflection;
using Core.Managers;
using LibraryInjected.FunctionBehaviors;

namespace LibraryInjected.Wrappers
{
    public class BehaviorsWrapper<T> : Wrapper where T : FunctionBehavior
    {
        public BehaviorsWrapper()
        {
            Functions =
Assembly.GetAssembly(typeof(FunctionBehavior)).GetTypes()
                .Where(x => x.IsSubclassOf(typeof(T)))
                .Select(type =>
(FunctionBehavior)Activator.CreateInstance(type))
                .ToList();
            foreach (var functionBehavior in Functions)
            {
                Logger.Info(functionBehavior.ToString(), "common");
            }
        }
    }
}
```

П1.25 Исходный текст файла «Main.cs»

```
using System;
using System.Collections.Generic;
using EasyHook;
using Core;
using Core.Dtos;
using Core.Managers;
using LibraryInjected.Extentions;
using LibraryInjected.FunctionBehaviors;
using LibraryInjected.FunctionsInjected;
using LibraryInjected.Wrappers;

namespace LibraryInjected
{
    public class Main : IEntryPoint
    {
        private readonly List<FunctionInjected> _functions;

        private readonly Wrapper _behaviorWrapper;

        public Main(RemoteHooking.IContext InContext, ProcessDto procDto,
SystemState state, string InChannelName)
        {
            try
            {
                _functions = new List<FunctionInjected>();

                switch (state)
                {
                    case SystemState.Scanning:
                        _behaviorWrapper =
RemoteHooking.IpcConnectClient<BehaviorsWrapper<FunctionBehaviorForXml>>(InCh
annelName);
                        break;
                    case SystemState.Locking:
                        _behaviorWrapper =
RemoteHooking.IpcConnectClient<BehaviorsWrapper<FunctionBehaviorForNLog>>(InC
hannelName);
                        break;
                }

                foreach (var functionBehavior in _behaviorWrapper.Functions)
                {

_functions.Add(functionBehavior.CreateAttachedTypeOfFunctionInjected(procDto)
);
                }
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, procDto.ProcName);
            }
        }
```

```
        public void Run(RemoteHooking.IContext InContext, ProcessDto procDto,
SystemState state, string InChannelName)
        {
            try
            {
                RemoteHooking.WakeUpProcess();

                while (true)
                {

                }
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, procDto.ProcName);
            }
        }
    }
}
```

П1.26 Исходный текст файла «Program.cs»

```
using System;
using System.Windows.Forms;

namespace WinFormApplication
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form());
        }
    }
}
```

П1.27 Исходный текст файла «App.config»

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="nameOfLogFile" value="log.txt"/>
    <add key="nameOfErrorFile" value="error.txt"/>
    <add key="basePathToLoggerModule" value="C:\logs\LoggerModule\"/>
  </appSettings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2"
/>
    </startup>
</configuration>
```

П1.28 Исходный текст файла «Form»

```
using System;
using System.Diagnostics;
using System.Linq;
using System.Windows.Forms;
using Core;
using Core.Dtos;
using Core.Managers;

namespace WinFormApplication
{
    public partial class Form : System.Windows.Forms.Form
    {
        public Form()
        {
            InitializeComponent();
            Process.GetProcesses().ToList().ForEach(x =>
this.dataGridView.Rows.Add(new string[] { x.Id.ToString(), x.ProcessName }));
        }

        private void notifyIcon_Click(object sender, EventArgs e)
        {
            this.WindowState = this.WindowState == FormWindowState.Normal ?
FormWindowState.Minimized : FormWindowState.Normal;
        }

        private void LockSystemButton_Click(object sender, EventArgs e)
        {
            SystemManager.CreateNewInjectProcess(SystemState.Locking,
GetProcId());
        }

        private void ScanSystemButton_Click(object sender, EventArgs e)
        {
            SystemManager.CreateNewInjectProcess(SystemState.Scanning,
GetProcId());
        }

        private void Form_Resize(object sender, EventArgs e)
        {
            this.ShowInTaskbar = this.WindowState !=
FormWindowState.Minimized;
        }

        private ProcessDto GetProcId()
        {
            if (this.dataGridView.SelectedRows.Count == 1 &&
this.dataGridView.SelectedCells.Count > 1)
            {
                this.label1.Text = string.Empty;
                return new ProcessDto {ProcId =
Convert.ToInt32(this.dataGridView.SelectedCells[0].Value), ProcName =
this.dataGridView.SelectedCells[1].Value.ToString()};
            }
```

```
                this.label1.Text = @"No process exists with that name!";
                return new ProcessDto { ProcId = -1, ProcName = string.Empty };
            }
        }
    }
```