Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Институт компьютерных наук и кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА № 2

«Мониторинг событий в процессе»

1. по дисциплине «Технологии разработки современного программного обеспечения»
2. Выполнил
3. студент гр. 5151004/10101 Веремейчук Я.Ю.

<подпись>

1. Проверил
2. преподаватель Завадский Е.В.

<подпись>

1. Санкт-Петербург
2. 2024
3. Цель работы

Написать программу, которая осуществляет мониторинг вызова функций в процессе.

1. Постановка задачи

Программа должна использовать технику Hook&Inject, которая заключается в том, чтобы заставить целевой процесс осуществить загрузку указанной DLL, которая, в свою очередь выполняет изменение таблиц с функциями DLL, осуществляя их перехват. При использовании такой техники есть возможность осуществлять мониторинг вызова заданных функций, а также существует возможность подменять значения результата работы функции, меняя поведение целевого процесса.

В лабораторной требуется написать программу и dll библиотеку, которые позволяют:

1. Отслеживать вызов указанных заранее функций. Каждый вызов функции в целевом процессе должен сопровождаться печатью сообщения на консоль разрабатываемой программы. Распечатанное сообщение должно содержать имя функции и метку времени.
2. Изменять поведение функции FindFirstFile, FindNextFile, CreateFile таким образом, что для процесса пропадает указанный заранее файл. Для изменения поведения достаточно изменять значения, возвращаемые этими функциями в соответствии с определенными входными параметрами. Должно работать для cmd.exe в 32х битной версии.

При написании программы должны быть выполнены несколько условий:

1. Программа должна компилироваться Visual Studio.
2. Программа требует привилегий администратора для своей работы.
3. Программа должна обладать интерфейсом командной строки.
4. Программа должна предлагать следующие действия:
   1. –pid < Target\_Process\_PID> – целевой процесс выбирается по значению PID.
   2. –name <Target\_Process\_Name> – целевой процесс выбирается по имени.
   3. –func <Function\_Name> – имя функции, вызов которой требуется отследить.
   4. –hide <File\_Name> – имя файла, который надо скрыть от процесса.
5. Считается, что доступны следующие сценарии запуска программы:
   1. Monitor.exe –pid 123 –func CreateFile
   2. Monitor.exe –name explorer.exe –func CreateFile

Monitor.exe –name explorer.exe –hide “C:\hello.txt”

1. Теоретические исследования

Перехват  (hooking) – это технология вычленения процесса из всех процессов системы для отслеживания в нем вызовов определенных функций, либо изменения поведения этих функций путем их подмены с помощью загрузки указанной DLL.

Внедрение (injection) – это технология, позволяющая выполнять другой код(полезную нагрузку) в адресном пространстве уже запущенного процесса. DLL-инъекции используются внешними программами, чтобы повлиять на поведение программы-жертвы таким образом, чтобы она начала работать не так, как задумывал её автор.

Программа, используемая для внедрения измененного кода в процесс называется DLL-инжектором.

Таким образом, техника Hook&Inject заключается в том, чтобы заставить целевой процесс осуществить загрузку указанной DLL, которая, в свою очередь, выполняет изменение таблиц с функциями DLL, изменяя тем самым поведение целевого процесса.

Есть несколько методов внедрения библиотеки:

1. **Реестр**. Чтобы внедрить библиотеку DLL в процессы, которые связаны с USER32.DLL, можно добавить имя библиотеки DLL к значению следующего раздела реестра: HKEY\_LOCAL\_MACHINE\Software\ Microsoft\ Windows NT \ CurrentVersion \ Windows\AppInit\_DLLs. Его значение содержит одно имя библиотеки DLL или группу библиотек DLL, разделенных запятой или пробелами. Все библиотеки DLL, указанные значением этого ключа, загружаются каждым приложением под управлением Windows, запущенным в текущем сеансе входа в систему.
2. **Общесистемные перехваты Windows.** Основная концепция перехватов такого рода заключается в том, что процедура обратного вызова перехвата выполняется в адресных пространствах каждого подключенного процесса в системе. Чтобы установить перехват, вы вызываете SetWindowsHookEx() с соответствующими параметрами. Как только приложение устанавливает общесистемный перехват, операционная система сопоставляет DLL с адресным пространством в каждом из своих клиентских процессов. Следовательно, глобальные переменные внутри библиотеки DLL будут "для каждого процесса" и не могут использоваться совместно процессами, которые загрузили перехват DLL.
3. **Внедрение библиотеки DLL с помощью CreateRemoteThread() функции API.** Метод, который в итоге был использован в лабораторной работе, заключается в использовании функции CreateRemoteThread() для создания удаленного потока в целевом процессе. Затем в этом удаленном потоке вызывается функция LoadLibrary(), которая загружает указанную DLL в адресное пространство целевого процесса. Таким образом, DLL внедряется в процесс, и его функциональность становится доступной в контексте этого процесса.
4. **Внедрение через надстройки BHO.** Иногда вам потребуется внедрить пользовательский код только в Internet Explorer. К счастью, Microsoft предоставляет простой и хорошо документированный способ для этой цели - вспомогательные объекты браузера. BHO-это реализован в виде com-DLL и когда он будет должным образом зарегистрировано, каждый раз, когда IE запускается, он загружает все com-компоненты, которые уже реализовали IObjectWithSite интерфейс.
5. **Надстройки MS Office.** Аналогично BHOs, если вам нужно внедрить в приложения MS Office собственный код, вы можете воспользоваться преимуществами предоставленного стандартного механизма, внедрив надстройки MS Office. Существует множество доступных примеров, показывающих, как реализовать такого рода надстройки.
6. Ход работы

Локальные перехваты (влияющие на определенные приложения), реализованные с помощью подхода модификации среды выполнения, должны выполняться в адресном пространстве целевой программы. Программа, которая манипулирует целевым процессом и заставляет его загружать перехват, называется инжектором. В лабораторной работе код настройки подключения содержится во внешнем ресурсе DLL, который является объектом внедрения.

Общий процесс подготовки подключения к загрузке и выполнению требует, чтобы инжектор выполнил следующие шаги:

1. Получение PID целевого процесса.
2. Выделение памяти внутри целевого процесса и запись в нее внешнего пути к DLL (запись пути к динамической библиотеке, который содержит подключение).
3. Создание потока внутри целевого процесса, который загрузит библиотеку и настроит подключение.

В лабораторной работе код настройки подключения находится в функции внешней библиотеки DLL, поэтому он будет автоматически выполнен при успешной загрузке библиотеки.

Дескриптор целевого процесса можно получить с помощью функции CreateToolhelp32Snapshot(), которая делает «снимок» указанных процессов. Флаг TH32CS\_SNAPPROCESS включает все процессы в системе в моментальный снимок.



Рисунок 1 – "Снимок" процессов

Полученные данные необходимо распарсить, для этого применяется функция Process32First(), которая извлекает сведения о первом процессе, обнаруженном в системном snapshot. Process32Next() используется для получения информации о следующем процессе после текущего. При каждом вызове Process32Next() структура entry обновляется новыми данными о следующем процессе в списке.

MultiByteToWideChar() преобразует строку из многобайтового формата в широкий формат. При первом вызове используются следующие входные значения:

* CP\_ACP: Кодовая страница текущей системы.
* 0: Флаги. В данном случае они не используются.
* proc\_name: Указатель на многобайтовую строку, которую необходимо преобразовать.
* -1: Функция обрабатывает всю входную строку, включая завершающий нулевой символ. Следовательно, результирующая строка Unicode имеет завершающий нулевой символ, и длина, возвращаемая функцией, включает этот символ.
* NULL: Указатель на выходной буфер. Первый вызов функции используется для определения размера необходимого буфера, поэтому передается NULL.
* 0: Размер выходного буфера. Первый вызов функции используется для определения размера необходимого буфера, поэтому передается

При втором вызове используются следующие входные значения:

* CP\_ACP: Кодовая страница текущей системы.
* 0: Флаги. В данном случае они не используются.
* proc\_name: Указатель на многобайтовую строку, которую необходимо преобразовать.
* -1: Длина входной строки. Значение -1 указывает, что функция должна определить длину строки автоматически, пока не встретится нулевой символ завершения строки.
* (LPWSTR)proc\_name\_w: Указатель на выходной буфер, в который будет записан результат преобразования.
* nChars: Размер выходного буфера.

Если имя процесса найдено в снимке, функция возвращает его идентификатор (th32ProcessID). В противном случае возвращается 0.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2 – Получение PID процесса

После успешного получения PID процесса можно переходить к внедрению библиотеки.

Для начала с помощью функции OpenProcess() открывается целевой процесс:

* PROCESS\_ALL\_ACCESS: Это флаг, указывающий, что мы запрашиваем максимальные права доступа к процессу. Он позволяет выполнять различные операции с процессом, включая чтение, запись и выполнение кода в его адресном пространстве.
* FALSE: Это флаг, указывающий, что дескриптор процесса не должен наследоваться дочерними процессами.
* PID: Это идентификатор процесса (PID) целевого процесса, в который мы хотим внедрить DLL.

Затем вычисляется количество байт, необходимых для хранения пути к DLL, и выделяется память в удаленном процессе с помощью функции VirtualAllocEx(). Эта память будет использоваться для хранения пути к DLL в адресном пространстве целевого процесса.

* process\_handle: Дескриптор открытого процесса, в который мы хотим выделить память.
* NULL: Это указатель на базовый адрес, с которого начнется выделение памяти в адресном пространстве процесса. В данном случае мы хотим, чтобы система сама выбрала подходящий адрес.
* bytes\_to\_alloc: Это количество байт, которые мы хотим выделить в удаленном процессе для хранения пути к DLL.
* MEM\_COMMIT: Это флаг, указывающий, что выделенная память должна быть выделена непосредственно (а не только зарегистрирована).
* PAGE\_READWRITE: Это флаг, указывающий, что выделенная память должна иметь разрешения на чтение и запись.

Путь к DLL копируется в выделенную память в удаленном процессе с помощью функции WriteProcessMemory(). Теперь путь к DLL доступен в адресном пространстве целевого процесса.

* process\_handle: Дескриптор открытого процесса, в который мы хотим записать данные (путь к DLL).
* remoteBufferForLibraryPath: Это указатель на выделенную память в удаленном процессе, в которую мы хотим записать данные.
* dll\_path: Это указатель на строку, содержащую путь к DLL, который мы хотим записать в удаленный процесс.
* bytes\_to\_alloc: Это количество байт, которое мы хотим записать в удаленный процесс.

С помощью функции GetProcAddress() ищется адрес функции LoadLibraryW() из библиотеки Kernel32.dll.

* GetModuleHandleW(L"Kernel32"): Эта функция возвращает дескриптор модуля Kernel32.dll, который представляет стандартную библиотеку Windows Kernel32.
* "LoadLibraryW": Это имя экспортируемой функции, которую мы ищем в указанной библиотеке (Kernel32.dll). В данном случае мы ищем функцию LoadLibraryW, которая загружает указанную библиотеку в адресное пространство процесса.

И наконец, создается удаленный поток в целевом процессе с помощью функции CreateRemoteThread(). Этот поток запускает функцию LoadLibraryW, передавая ей путь к DLL в качестве аргумента.

* process\_handle: Дескриптор открытого процесса, в который мы хотим создать удаленный поток.
* NULL: Это указатель на атрибуты потока, которые мы не используем в данном случае.
* 0: Это размер стека потока, который мы также не используем.
* loadLibraryFunction: Это указатель на функцию, которую мы хотим выполнить в удаленном процессе. В данном случае это адрес функции LoadLibraryW.
* remoteBufferForLibraryPath: Это указатель на аргумент, который мы хотим передать функции LoadLibraryW в удаленном процессе (путь к DLL).
* 0: Это флаги создания потока, которые мы не используем.
* NULL: Это идентификатор потока, который мы не сохраняем.

Нельзя просто передать LoadLibraryA или LoadLibraryW в четвертом параметре функции CreateRemoteThread. При сборке программы в конечный двоичный файл помещается раздел импорта. Этот раздел состоит из серии шлюзов к импортируемым функциям. Поэтому, когда код вызывает функцию, например, LoadLibraryA, то в разделе импорта модуля генерируется вызов соответствующего шлюза. А уже от шлюза происходит переход к реальной функции.

Следовательно, прямая ссылка на LoadLibraryA в вызове CreateRemoteThread преобразуется в обращение к шлюзу LoadLibraryA в разделе импорта модуля. Передача адреса шлюза в качестве стартового адреса удаленного потока заставит этот поток неизвестно что выполнить. И скорее всего это окончится нарушением доступа. Чтобы напрямую вызывать LoadLibraryA, минуя шлюз, необходимо выяснить ее точный адрес в памяти с помощью GetProcAddress. Вызов CreateRemoteThread предполагает, что Kerne32.dll спроецирована в локальном процессе на ту же область памяти, что и в удаленном. Kernel32.dll используется всеми приложениями, и, как показывает опыт, система проецирует эту DLL в каждом процессе по одному и тому же адресу. Поэтому CreateRemoteThread необходимо вызывать следующим образом:

HANDLE remoteThreadHandle = CreateRemoteThread(process\_handle, NULL, 0, loadLibraryFunction, remoteBufferForLibraryPath, 0, NULL);

Строка, хранящая путь до библиотеки, находится в адресном пространстве вызывающего процесса. Ее адрес передается только что созданному потоку, который в свою очередь передает его в LoadLibraryA. Но, когда LoadLibraryA будет проводить разыменование (dereferencing) этого адреса, она не найдет по нему строку с полным именем файла DLL и скорее всего вызовет нарушение доступа в потоке удаленного процесса; пользователь увидит сообщение о необрабатываемом исключении, и удаленный процесс будет закрыт.

Эта проблема решается размещением строки с полным именем файла DLL в адресном пространстве удаленного процесса. Впоследствии, вызывая CreateRemoteThread, мы передадим ее адрес (в удаленном процессе). На этот случай в Windows предусмотрена функция VirtualAllocEx, которая позволяет процессу выделять память в чужом адресном пространстве.

Выделив память, необходимо скопировать строку из локального адресного пространства в удаленное с помощью функции WriteProcessMemory.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Внедрение библиотеки

После этого необходимо установить связь с удаленным сервером, отправить ему переданные параметры, ожидать ответа от сервера и очищать все использованные ресурсы после завершения обмена данными.

На этом этапе DLL внедрена в удаленный процесс, а ее функция DllMain получила уведомление DLL\_PROCESS\_ATTACH и может приступить к выполнению нужного кода.

Для реализации вредоносной библиотеки, необходимо воспользоваться библиотекой Detourse. Точка входа подключенной библиотеки – это DLLMain().

Функция DetourFunction предназначена для перехвата и изменения поведения определенных системных функций операционной системы Windows. Она принимает имя функции в качестве аргумента и осуществляет перехват для функций, начинающихся с определенных строк, таких как "FindFirstFile", "FindNextFile", "CreateFile", "CloseHandle".

Когда функция вызывается с определенным именем функции, она начинает транзакцию перехвата (DetourTransactionBegin), обновляет поток для перехвата (DetourUpdateThread(GetCurrentThread())) и прикрепляет перехват к соответствующей функции с помощью библиотеки Detours (DetourAttach). Если перехват успешно прикреплен, это отображается сообщением об успешном перехвате, иначе - сообщением о неудаче. После каждого перехвата транзакция завершается (DetourTransactionCommit).

Для примера была рассмотрена простейшая программа, вызывающая функцию CreateFileW. Для нее была выполнена команда:

Monitoring.exe --name VICTIM.exe --func CreateFileW.

Исполняемый файл был запущен в IDA Pro. В режиме отладки был просмотрен код функции CreateFileW.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4 – Фрагмент тестовой программы

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 5 – Переход к функции, которая будет отслеживаться

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 6 – Первоначальный вид функции CreateFileW

Далее, не прекращая отладку, был запущен инжектор.



Рисунок 7 – Запуск инжектора

И вновь был проанализирован код функции CreateFileW. Он изменился. Первые две инструкции затерлись на инструкцию jmp. Код, который должен исполняться далее, будет выполнен позже. Как и инструкции, которые были перезатерты.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 8 – Измененный код функции CreateFileW

Нажав D (data – преобразовать код в символы), C (code - в код), можно увидеть инструкцию jmp, которая перенаправляет на код вредоносной Dll.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 9 – Новая инструкция jmp

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 10 – Запись адреса возврата на стек

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 11 – Перенаправление на вредоносную Dll

После выполнения этих инструкций можно увидеть, что функция CreateFileW была перехвачена.

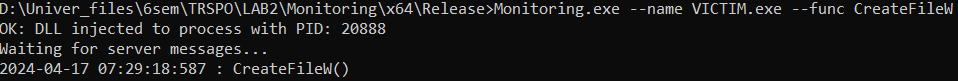


Рисунок 12 – Успешное выполнение мониторинга

После выполнения вредоносного кода происходит прыжок на код, который был ранее перезатерт.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 13 – Возврат на затертые инструкции

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 14 – Выполнение затертых инструкций

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 15 – Передача управления и выполнение изначальной функции

Еще один пример выполнения внедрения.

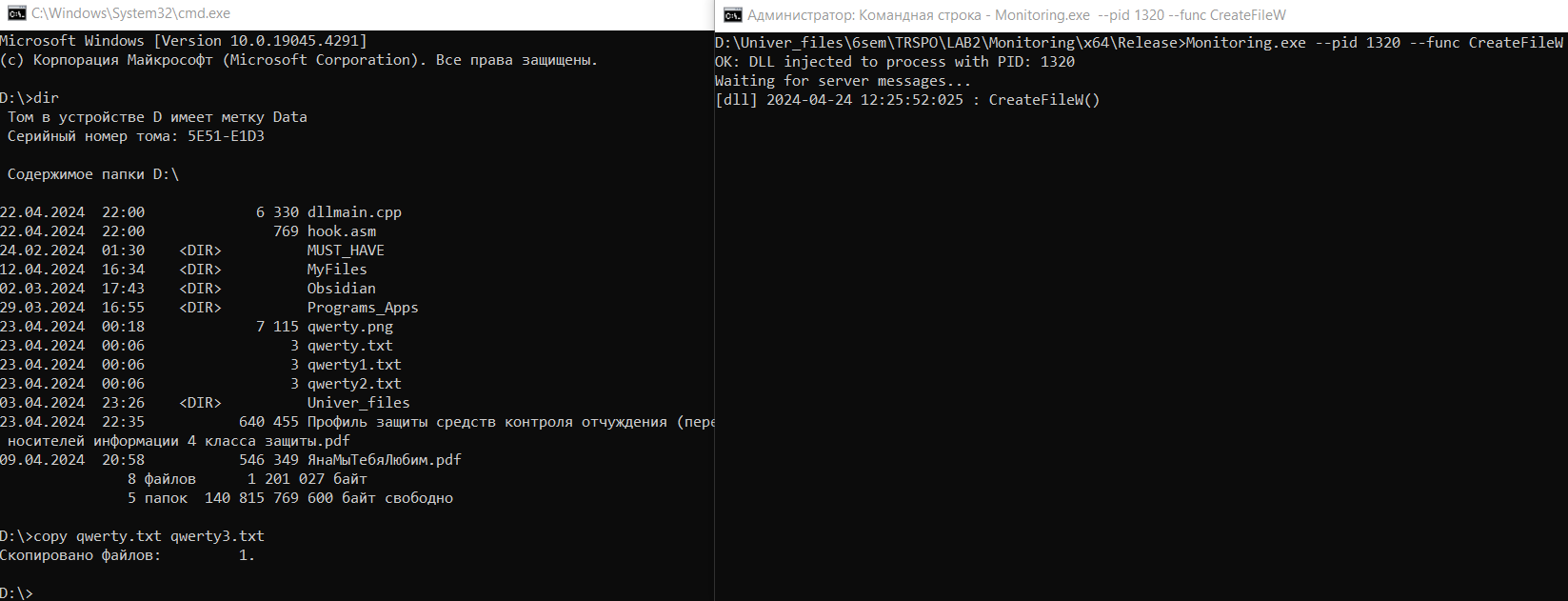


Рисунок 16 – Отслеживание функции CreateFileW

1. Вывод

В ходе лабораторной работы была написана программа, которая осуществляет мониторинг вызова функций в процессе.