Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе
Курс: Системное программирование
Тема: Создание службы Windows

Выполнил студент группы 13541/3 _____ Д.В. Круминьш
(подпись)

Преподаватель _____ Е.В. Душутина
(подпись)

Санкт-Петербург
2017 г.

Содержание

Постановка задачи

В данной работе необходимо оформить ранее созданное приложение перехвата windows api в виде службы.

Необходимо выполнить:

- Рассмотреть способ добавление собственных служб;
- Написать программу-службу, которая позволяет контролировать инъекцию ранее написанной библиотеки **dll**, с функциями перехватчиками;
- Удостовериться в корректности работы службы.

Способы добавления служб

В операционной системе Windows можно запускать приложения в качестве службы. Такое приложение не имеет рабочего терминала и выполняется в фоновом режиме.

Для добавление новой службы можно использовать утилиту **Sc**.

SC представляет из себя утилиту командной строки, которая реализует вызовы ко всем функциям интерфейса прикладного программирования (API) управления службами Windows. С ее помощью можно производить любые действия со службами — просматривать состояние, управлять (запускать, останавливать и т.п.), изменять параметры, а также создавать новые службы.

Однако в данной работе будет использоваться программный метод, с написанием структуры **SERVICE_TABLE_ENTRY**.

Для начала работы службы необходимо дать знать менеджеру служб о том, что необходимо добавить выбранное приложение в таблицу служб. Для этого необходимо указать точку входа.

SERVICE_TABLE_ENTRY - это структура, которая описывает название службы и её точку входа для менеджера служб. В данном случае вход будет происходить через функцию **ServiceMain**. Функция StartServiceCtrlDispatcher связывает службу с SCM (Service Control Manager, то есть менеджер служб).

Для разработки понадобятся две глобальные переменные:

SERVICE_STATUS и **SERVICE_STATUS_HANDLE**

для хранения статуса службы и её дескриптора. Они будут использоваться в методе **ControlHandler** - обработчике запросов к службе от системы.

ControlHandler вызывается каждый раз, когда SCM шлет запросы на изменения состояния сервиса. В основном она используется для описания корректной работы завершения сервиса.

Обзор составляющих службы

В прошлой работе по написанию перехватчика Windows api, использовался глобальный перехват с помощью реестра и перезагрузкой системы. В данной работе рассматривается вариант с инъекцией и де-инъекцией dll библиотеки в процесс **explorer.exe** (проводник).

Далее приведен обзор логики каждой написанной функции, а также исходный код.

3.1 Функция main

Функция является основной точкой входа, в том числе и для пользователя, реализуя следующие команды прямо в консоли: **install**, **start**, **remove**.

Для менеджера служб описана структура SERVICE_TABLE_ENTRY, в которой указано название службы и указана функция **ServiceMain**. Как итог происходит связывание службы с менеджером.

3.2 Функция ServiceMain

Функция является точкой входа для сервиса менеджера. Выполняются следующие действия:

1. Инъекция dll библиотеки, используя функцию InjectDLL;
2. Регистрация сервиса с помощью системной функции **RegisterServiceCtrlHandler**;
3. Установка статуса - **SERVICE_RUNNING**.

3.3 Функция ControlHandler

Обработчик запросов от системы к службе. Реализована функциональность для:

1. SERVICE_CONTROL_STOP;
    - Вызов метода **EjectDLL**, для выгрузки dll из процесса, к которому он был инжектирован;
    - Обновление статуса сервиса.
2. SERVICE_CONTROL_SHUTDOWN.

3.4 Функция InjectDLL

Алгоритм:

1. Получение дескриптора процесса, для которого необходимо произвести инъекцию;
2. Выделение памяти для имени dll в указанном процессе;
3. Запись имени dll(включая полный путь) в выделенную память;
4. Создание потока в удаленном процессе и загрузка dll;
5. Ожидание успешной инъекции;
6. Освобождение памяти, выделенной под имя dll, закрытие потоков.

Наиболее важной функцией, в том числе и для EjectDLL, является - **CreateRemoteThread**[1].

```
1 HANDLE WINAPI CreateRemoteThread(
2   _In_  HANDLE                 hProcess,
3   _In_  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
4   _In_  SIZE_T                 dwStackSize,
5   _In_  LPTHREAD_START_ROUTINE lpStartAddress,
6   _In_  LPVOID                 lpParameter,
7   _In_  DWORD                  dwCreationFlags,
8   _Out_ LPDWORD                lpThreadId
9 );
```

Листинг 3.1: Прототип CreateRemoteThread

Параметры:

- **hProcess** - дескриптор процесса, в котором будет создан поток;
- **lpThreadAttributes** - параметры безопасности;
- **dwStackSize** - начальный размер стека, если передан 0, то используется стандартный размер(1 MB);
- **lpStartAddress** - адрес функции которую необходимо выполнить, в данном случае это **LoadLibraryA** из **kernel32** для загрузки библиотеки;
- **lpParameter** - параметр передаваемый в функцию(библиотека для загрузки);
- **dwCreationFlags** - дополнительные флаги;
- **lpThreadId** - выставляется если необходимо вернуть идентификатор процесса.

3.5 Функция EjectDLL

Алгоритм:

1. Получение(с помощью GetProcessId) дескриптора процесса, к которому инжектирована библиотека;
2. Снапшот переданного процесса(explorer.exe) с помощью **CreateToolhelp32Snapshot**[2].

3. Поиск адреса инжектированной dll
4. Создание удаленного потока, и освобождение от инжектированной библиотеки

В данном случае при создании удаленного потока с помощью CreateRemoteThread, вызывалась функция **FreeLibrary** и адрес dll для выгрузки.

```
1 HANDLE WINAPI CreateToolhelp32Snapshot(
2   _In_ DWORD dwFlags,
3   _In_ DWORD th32ProcessID
4 );
```

Листинг 3.2: Прототип CreateToolhelp32Snapshot

Параметры:

- **dwFlags** - тип работы функции(снапошоты чего делать(всей системы, конкретного процесса));
- **th32ProcessID** - идентификатор процесса для уточнения работы.

3.6 Функция GetProcessId

Алгоритм:

1. Снапшот всех процессов с помощью CreateToolhelp32Snapshot;
2. Поиск нужного процесса по его имени;
3. Возврат идентификатора найденного процесса.

3.7 Функция StartHookService

Алгоритм:

1. Устанавливаем связь с диспетчером управления служб;
2. Открываем базу данных диспетчера управления службами;
3. Запускаем существующую службу.

3.8 Функция RemoveHookService

Алгоритм:

1. Устанавливаем связь с диспетчером управления служб;
2. Открываем базу данных диспетчера управления службами;
3. Открываем существующую службу;
4. Вызываем системную функцию **DeleteService**[3] для удаления службы.

```
BOOL WINAPI DeleteService(
  _In_ SC_HANDLE hService
);
```

Листинг 3.3: Прототип DeleteService

Параметры:

- **hService** - дескриптор службы для удаления.

3.9 Функция InstallHookService

Алгоритм:

1. Устанавливаем связь с диспетчером управления служб;
2. Открываем базу данных диспетчера управления службами;
3. Создаем объект службы и добавляем его в базу данных диспетчера управления службами;
4. Проверяем на наличие ошибки с помощью **GetLastError**.

3.10 Функция addLogMessage

Функция добавляет сообщение в файл лога службы.

3.11 Исходный код

Исходный код также имеет комментарии.

```
#include <iostream>
#include <fstream>
#include <atlstr.h>
#include <tlhelp32.h>
using namespace std;


#define SERVICE_NAME "myHookService"
#define SERVICE_PATH "C:\\myService.exe"
#define HOOK_DLL_PATH "C:\\AppInitHook.dll"

SERVICE_STATUS ServiceStatus;    // Статус службы
SERVICE_STATUS_HANDLE hStatus;   // Дескриптор статуса службы


void addLogMessage(char * msg) {
    char filename[] = "E:\\myServiceLog.txt";
    wofstream logfile;
    logfile.open(filename, ios::app);
    logfile << msg << endl;
    logfile.close();
}

```

```
24 int InstallHookService() {
25     //устанавливаем связь с диспетчером управления службами на ПК
26     // и открываем указанную базу данных диспетчера управления службами.
27     SC_HANDLE hSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_CREATE_SERVICE)
       ↪ ;
28     if (!hSCManager) {
29         addLogMessage("Error: Can't open Service Control Manager");
30         return -1;
31     }
32     // создаем объект службы
33     // и добавляем его в указанную базу данных диспетчера управления службами.
34     //ДУС-- – диспетчер управления службами
35     SC_HANDLE hService = CreateService(
36         hSCManager, // Дескриптор базы данных диспетчера управления службой
37         SERVICE_NAME,// устанавливаемое имя службы.
38         SERVICE_NAME,// отображаемое имя
39         SERVICE_ALL_ACCESS,// Доступ к службе.
40         SERVICE_WIN32_OWN_PROCESS, //Служба запускается в своем собственном
       ↪ процессе.
41         SERVICE_DEMAND_START,//Служба, запускается ДУС, когда процесс вызывает
       ↪ функцию StartService.
42         SERVICE_ERROR_NORMAL,//Прогр. запуска регистрирует ошибку, но продолжает
       ↪ операцию запуска.
43         SERVICE_PATH,//полный путь доступа к двоичному файлу службы
44         NULL, NULL, NULL, NULL, NULL
45     );
46
47     if (!hService) {
48         int err = GetLastError();
49         switch (err) {
50         case ERROR_ACCESS_DENIED:
51             addLogMessage("Error: ERROR_ACCESS_DENIED");
52             break;
53         case ERROR_CIRCULAR_DEPENDENCY:
54             addLogMessage("Error: ERROR_CIRCULAR_DEPENDENCY");
55             break;
56         case ERROR_DUPLICATE_SERVICE_NAME:
57             addLogMessage("Error: ERROR_DUPLICATE_SERVICE_NAME");
58             break;
59         case ERROR_INVALID_HANDLE:
60             addLogMessage("Error: ERROR_INVALID_HANDLE");
61             break;
62         case ERROR_INVALID_NAME:
63             addLogMessage("Error: ERROR_INVALID_NAME");
64             break;
65         case ERROR_INVALID_PARAMETER:
66             addLogMessage("Error: ERROR_INVALID_PARAMETER");
67             break;
68         case ERROR_INVALID_SERVICE_ACCOUNT:
69             addLogMessage("Error: ERROR_INVALID_SERVICE_ACCOUNT");
70             break;
71         case ERROR_SERVICE_EXISTS:
72             addLogMessage("Error: ERROR_SERVICE_EXISTS");
73             break;
74         default:
75             addLogMessage("Error: Undefined");
76         }
77         CloseServiceHandle(hSCManager);
78         return -1;
79     }
```

```
80      CloseServiceHandle(hService);
81
82      CloseServiceHandle(hSCManager);
83      addLogMessage("Success install service!");
84      return 0;
85  }
86
87  int RemoveHookService() {
88      //устанавливаем связь с диспетчером управления службами на ПК
89      // и открываем указанную базу данных диспетчера управления службами.
90      SC_HANDLE hSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
91      if (!hSCManager) {
92          addLogMessage("Error: Can't open Service Control Manager");
93          return -1;
94      }
95      // открываем существующую службу.
96      SC_HANDLE hService = OpenService(hSCManager, SERVICE_NAME, SERVICE_STOP |
        ↪ DELETE);
97      if (!hService) {
98          addLogMessage("Error: Can't remove service");
99          CloseServiceHandle(hSCManager);
100         return -1;
101     }
102
103     DeleteService(hService);
104     CloseServiceHandle(hService);
105     CloseServiceHandle(hSCManager);
106     addLogMessage("Success remove service!");
107     return 0;
108 }
109
110 int StartHookService() {
111     //устанавливаем связь с диспетчером управления службами на ПК
112     // и открываем указанную базу данных диспетчера управления службами.
113     SC_HANDLE hSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_CREATE_SERVICE)
        ↪ ;
114     // запускаем существующую службу.
115     SC_HANDLE hService = OpenService(hSCManager, SERVICE_NAME, SERVICE_START);
116     if (!StartService(hService, 0, NULL)) {
117         CloseServiceHandle(hSCManager);
118         addLogMessage("Error: Can't start service");
119         return -1;
120     }
121
122     CloseServiceHandle(hService);
123     CloseServiceHandle(hSCManager);
124     addLogMessage("Success start service!");
125     return 0;
126 }
127
128 DWORD GetProcessId(LPCTSTR ProcessName)
129 {
130     PROCESSENTRY32 pt;
131     // получение снапшота всех процессов
132     HANDLE hsnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
133     pt.dwSize = sizeof(PROCESSENTRY32);
134     if (Process32First(hsnap, &pt)) { // поиск необходимого процесса
135         do {
136             if (!lstrcmpi(pt.szExeFile, ProcessName)) {
137                 CloseHandle(hsnap);
```

```
138                 return pt.th32ProcessID; // возврат найденного идентификатора
139             }
140         } while (Process32Next(hsnap, &pt));
141     }
142     CloseHandle(hsnap);
143     return 0;
144 }
145
146 BOOL InjectDLL(DWORD pId, char *dllName)
147 {
148     // дескриптор процесса, для которого необходимо произвести инъекцию
149     HANDLE h = OpenProcess(PROCESS_ALL_ACCESS, false, pId);
150     if (h)
151     {
152         // для загрузки библиотеки
153         LPVOID LoadLibAddr = (LPVOID)GetProcAddress(GetModuleHandleA("kernel32.
    ↪ dll"), "LoadLibraryA");
154         // выделение памяти для имени dll в указанном процессе
155         LPVOID dereercomp = VirtualAllocEx(h, NULL, strlen(dllName), MEM_COMMIT
    ↪  | MEM_RESERVE, PAGE_READWRITE);
156         // запись имени dllвключая( полный путь) в выделенную память
157         WriteProcessMemory(h, dereercomp, dllName, strlen(dllName), NULL);
158         // создаем поток в удаленном процессе и загружаем dll
159         HANDLE asdc = CreateRemoteThread(h, NULL, NULL, (LPTHREAD_START_ROUTINE
    ↪ )LoadLibAddr, dereercomp, 0, NULL);
160         // ожидание успешной инъекции
161         WaitForSingleObject(asdc, INFINITE);
162         // освобождаем память выделенную под имя dll, закрываем потоки
163         VirtualFreeEx(h, dereercomp, strlen(dllName), MEM_RELEASE);
164         CloseHandle(asdc);
165         CloseHandle(h);
166         return true;
167     }
168     return false;
169 }
170
171 BOOL EjectDLL(DWORD pId, CONST CHAR * ChaDLL)
172 {
173     // получение дескриптора процесса, к которому инжектирована библиотека
174     HANDLE HanProcess = OpenProcess(PROCESS_ALL_ACCESS, false, pId);
175
176     // копирование пути к библотеке и инициализация переменных для работы
177     CHAR ChaDLLFilePath[(MAX_PATH + 16)] = { 0 };
178     strcpy(ChaDLLFilePath, ChaDLL);
179     HMODULE ModDLLHandle = NULL;
180     BYTE * BytDLLBaseAdress = 0;
181     MODULEENTRY32 MOEModuleInformation = { 0 };
182     MOEModuleInformation.dwSize = sizeof(MODULEENTRY32);
183     // снапшот переданного процесса
184     HANDLE HanModuleSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE, pId)
    ↪ ;
185     // для перечисления модулей в процессе
186     Module32First(HanModuleSnapshot, &MOEModuleInformation);
187     do
188     {
189         // если в процессе имеется инъекция нужной dll библиотеки
190         if (!strcmp(MOEModuleInformation.szExePath, ChaDLLFilePath))
191         {
192             // получение дескриптора процесса
193             ModDLLHandle = MOEModuleInformation.hModule;
```

```
194             BytDLLBaseAdress = MOEModuleInformation.modBaseAddr;
195         }
196     } while (Module32Next(HanModuleSnapshot, &MOEModuleInformation));
197     CloseHandle(HanModuleSnapshot);
198     // дескриптор для kernel32
199     HMODULE ModKernel32 = GetModuleHandle("Kernel32.dll");
200     if (ModKernel32 != NULL)
201     {
202         if (ModDLLHandle != NULL &&
203             BytDLLBaseAdress != 0)
204         {
205             // создание удаленного потока, и освобождение от инжектированной
    ↪ библиотеки
206             HANDLE HanDLLThread = CreateRemoteThread(HanProcess, NULL, 0,
    ↪ LPTHREAD_START_ROUTINE(GetProcAddress(ModKernel32, "FreeLibrary")), (VOID
    ↪  *)BytDLLBaseAdress, 0, NULL);
207             if (HanDLLThread != NULL)
208             {
209                 // ожидание выполнения
210                 if (WaitForSingleObject(HanDLLThread, INFINITE) != WAIT_FAILED)
211                 {
212                     CloseHandle(HanDLLThread);
213                     CloseHandle(HanProcess);
214                     return TRUE;
215                 }
216                 CloseHandle(HanDLLThread);
217             }
218         }
219     }
220     CloseHandle(HanProcess);
221     return FALSE;
222 }
223
224 //Обработка запросов к службе от системы
225 void ControlHandler(DWORD request) {
226     switch (request)
227     {
228     case SERVICE_CONTROL_STOP:
229         EjectDLL(GetProcessId(TEXT("explorer.exe")), HOOK_DLL_PATH);
230         ServiceStatus.dwWin32ExitCode = 0;
231         ServiceStatus.dwCurrentState = SERVICE_STOPPED;
232         SetServiceStatus(hStatus, &ServiceStatus);
233         return;
234
235     case SERVICE_CONTROL_SHUTDOWN:
236         ServiceStatus.dwWin32ExitCode = 0;
237         ServiceStatus.dwCurrentState = SERVICE_STOPPED;
238         SetServiceStatus(hStatus, &ServiceStatus);
239         return;
240
241     default:
242         break;
243     }
244
245     SetServiceStatus(hStatus, &ServiceStatus);
246
247     return;
248 }
249
250 void ServiceMain(int argc, char *argv[])
```

```
{
    InjectDLL(GetProcessId(TEXT("explorer.exe")), HOOK_DLL_PATH);

    int err;  // Возвращаемое значение
              //  char buffer[128]; // Буфер для сообщений
    WORD wVersionRequested;  // Запрашиваемая версия
    WSADATA wsaData;  // Структура инфции– о сокетах
    HANDLE hCp;  // Описатель порта завершения
                 //  LPOVERLAPPED overlapped; // Структура асинхронного I/O
    HANDLE hThread;  // Хендл потока
    DWORD ThreadId;  // Идентификатор потока
    DWORD flags;  // Флаги фции– WSARecv

    //инициализация статуса
    ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
    ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP |
    ↪ SERVICE_ACCEPT_SHUTDOWN;
    ServiceStatus.dwWin32ExitCode = 0;
    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwCheckPoint = 0;
    ServiceStatus.dwWaitHint = 0;

    hStatus = RegisterServiceCtrlHandler(SERVICE_NAME, (LPHANDLER_FUNCTION)
    ↪ ControlHandler);
    if (hStatus == (SERVICE_STATUS_HANDLE)0) {
        return;
    }

    ServiceStatus.dwCurrentState = SERVICE_RUNNING;
    SetServiceStatus(hStatus, &ServiceStatus);

    return;
}

//Запуск службы
int main(int argc, char *argv[]) {
    if (argc - 1 == 0) {     //Описание точки входа для менеджера служб
        SERVICE_TABLE_ENTRY ServiceTable[1];
        ServiceTable[0].lpServiceName = SERVICE_NAME;
        ServiceTable[0].lpServiceProc = (LPSERVICE_MAIN_FUNCTION)ServiceMain;

        //Связывание службы с менеджером.
        if (StartServiceCtrlDispatcher(ServiceTable) == false) {
            return GetLastError();
        }
    }
    else if (strcmp(argv[argc - 1], "install") == 0) {
        InstallHookService();
    }
    else if (strcmp(argv[argc - 1], "remove") == 0) {
        RemoveHookService();
    }
    else if (strcmp(argv[argc - 1], "start") == 0) {
        StartHookService();
    }
    return 0;
}
```

Листинг 3.4: main.cpp

Проверка корректности работы

В корне диска **C** находятся:

1. AppInitHook.dll - библиотека для инъекции, с функциями перехватчиками;
2. myService.exe - программа-служба.

Была открыта командная строка от имени администратора, это необходимо для работы с реестром и инъекцией библиотеки.

4.1 Изменение статусов службы

Для начала, служба была установлена.

```
1 Microsoft Windows [Version 10.0.14393]
2 (c) Корпорация Майкрософт (Microsoft Corporation), 2016. Все права защищены.
3
4 C:\Windows\system32>cd..
5
6 C:\Windows>cd..
7
8 C:\>myService.exe install
```

Листинг 4.1: Лог консоли

Как и предполагалось, в службах появилась новая запись.

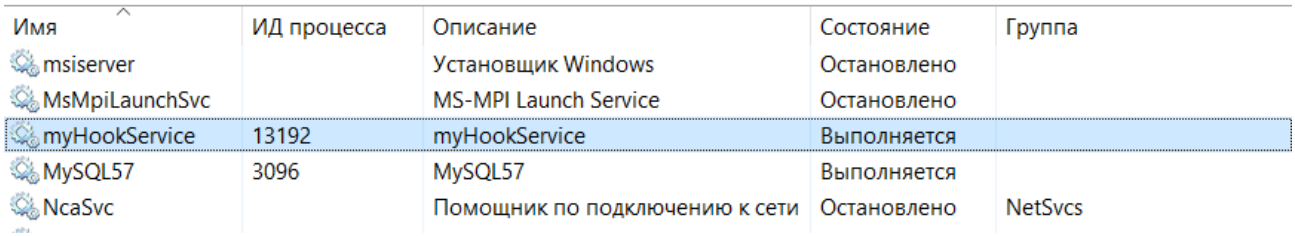
Имя	ИД процесса	Описание	Состояние	Группа
msiserver		Установщик Windows	Остановлено	
MsMpiLaunchSvc		MS-MPI Launch Service	Остановлено	
myHookService		myHookService	Остановлено	
MySQL57	3096	MySQL57	Выполняется	
NcaSvc		Помощник по подключению к сети	Остановлено	NetSvcs

Рис. 4.1: Новая служба

Запустим службу.

```
1 C:\>myService.exe start
```

Листинг 4.2: Лог консоли



Имя	ИД процесса	Описание	Состояние	Группа
msiserver		Установщик Windows	Остановлено	
MsMpiLaunchSvc		MS-MPI Launch Service	Остановлено	
myHookService	13192	myHookService	Выполняется	
MySQL57	3096	MySQL57	Выполняется	
NcaSvc		Помощник по подключению к сети	Остановлено	NetSvcs

Рис. 4.2: Запущенная служба

Состояние службы изменилось, а также процессу был дан идентификатор. В данном этапе была выполнена проверка на работу с перехватчиком.

После выполнения действий по перехвату, служба была остановлена, прямо из менеджера служб.

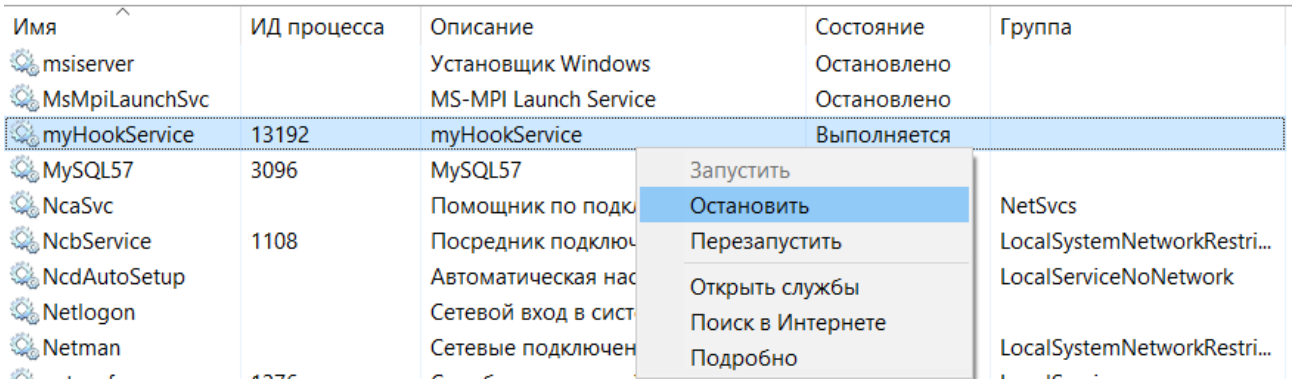
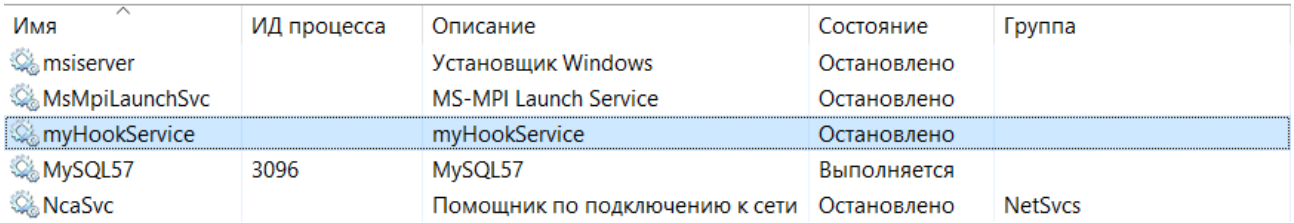


Рис. 4.3: Остановка службы

После остановки, служба становиться неактивной, и никакие действия более не логируются.



Имя	ИД процесса	Описание	Состояние	Группа
msiserver		Установщик Windows	Остановлено	
MsMpiLaunchSvc		MS-MPI Launch Service	Остановлено	
myHookService		myHookService	Остановлено	
MySQL57	3096	MySQL57	Выполняется	
NcaSvc		Помощник по подключению к сети	Остановлено	NetSvcs

Рис. 4.4: Неактивная служба

4.2 Работа перехватчика

Проведем туже последовательность действий, что и в работе с глобальным перехватчиком.

Для двух вариантов работы корзины, были выполнены следующие действия:

1. Открытие диска **E** в проводнике;
2. Переход в папку **testFolder**;
3. Создание новой папки, с именем **qwerty**;
4. Переход в созданную папку;

5. Создание нового текстового файла **123.txt**;
6. Открытие созданного файла в блокноте;
7. Запись произвольных символов;
8. Сохранение и закрытие файла;
9. Снова открытие файла **123.txt**;
10. Закрытие файла;
11. Удаление файла **123.txt**;
12. Удаление папки **testFolder**;

```
1  Time: 25-11-2017 13:11:25 | CreateDirectoryW | User: Tom | Path: E:\testFolder\ Новая
       ↪ папка
2  Time: 25-11-2017 13:11:39 | ReadFile | User: Tom | Path: E:\testFolder\qwerty\123.txt
3  Time: 25-11-2017 13:11:45 | ReadFile | User: Tom | Path: E:\testFolder\qwerty\123.txt
4  Time: 25-11-2017 13:12:11 | DeleteFileW | User: Tom | File: E:\$RECYCLE.BIN\S
       ↪ -1-5-21-2936131933-2071197896-2896244546-1000\$I4T64F2.txt
5  Time: 25-11-2017 13:12:11 | RemoveDirectoryW | User: Tom | Path: E:\$RECYCLE.BIN\S
       ↪ -1-5-21-2936131933-2071197896-2896244546-1000\$RPXIH2Q
6  Time: 25-11-2017 13:12:54 | CreateDirectoryW | User: Tom | Path: E:\testFolder\ Новая
       ↪ папка
7  Time: 25-11-2017 13:13:06 | ReadFile | User: Tom | Path: E:\testFolder\qwerty\123.txt
8  Time: 25-11-2017 13:13:08 | ReadFile | User: Tom | Path: E:\testFolder\qwerty\123.txt
9  Time: 25-11-2017 13:13:12 | RemoveDirectoryW | User: Tom | Path: E:\testFolder\qwerty
10 Trying to detach
11 Detached
```

Листинг 4.3: Содержимое файла лога

```
1  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder
2  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder
3  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\ Новая
       ↪ папка
4  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\ Новая
       ↪ папка
5  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\ Новая
       ↪ папка
6  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder
7  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\ Новая
       ↪ папка
8  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\
       ↪ logFolder
9  Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\
       ↪ logFolder
10 Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\
       ↪ logFolder
11 Time: 25-11-2017 13:11:25 | GetFileAttributesW | User: Tom | File: E:\testFolder\ Новая
       ↪ папка
12 ...
```

Листинг 4.4: Лог GetFileAttributesW

Содержимое лога также идентично логу из работы с глобальным перехватчиком. Строки

Trying to detach и Detached

были добавлены в код библиотеки, при вызове функции **DLL_PROCESS_DETACH**, то есть при выгрузке библиотеки.

Вывод

Данная работа расширила мои знания в инъекции сторонних библиотек. Так с помощью функции создания удаленного потока, имеется возможность загружать сторонние библиотеки-перехватчики, в режиме реального времени, уже без перезагрузки системы.

Преобразование программы в службу windows, представило возможности по контролю её состояния (выполняется, остановлено). Сам механизм служб схож с концепцией демонов в UNIX.

Первоначально работа проводилась на **Windows 10 сборка 14393**, но с течением времени система обновилась на версию **Windows 10 сборка 16299**, в которой перестал работать перехватчик на функцию **ReadFile**.

Литература

[1] CreateRemoteThread function [Электронный ресурс]. — URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682437(v=vs.85).aspx (дата обращения: 2017-11-25).

[2] CreateToolhelp32Snapshot function [Электронный ресурс]. — URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682489(v=vs.85).aspx (дата обращения: 2017-11-25).

[3] DeleteService function [Электронный ресурс]. — URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682562(v=vs.85).aspx (дата обращения: 2017-11-25).

[4] SERVICE_TABLE_ENTRY structure [Электронный ресурс]. — URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms686001(v=vs.85).aspx).aspx (дата обращения: 2017-11-25).

[5] Создание своего Windows Service [Электронный ресурс]. — URL: https://habrahabr.ru/post/71533/ (дата обращения: 2017-11-25).

Дополнения

Работа производилась на реальной системы, с параметрами представленными ниже.

Элемент	Значение
Имя ОС	Майкрософт Windows 10 Pro (Registered Trademark)
Версия	10.0.14393 Сборка 14393
Дополнительное описание ОС	Недоступно
Изготовитель ОС	Microsoft Corporation
Имя системы	USER-PC
Изготовитель	HP
Модель	OMEN by HP Laptop 15-ce0xx
Тип	Компьютер на базе x64
SKU системы	1ZB00EA#ACB
Процессор	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2496 МГц, ядер: 4, логических процессоров: 4
Версия BIOS	American Megatrends Inc. F.04, 10.05.2017
Версия SMBIOS	3.0
Версия встроенного контроллера	40.20
Режим BIOS	Устаревший
Изготовитель основной платы	HP
Модель основной платы	Недоступно
Имя основной платы	Основная плата
Роль платформы	Мобильный
Состояние безопасной загрузки	Не поддерживается
Конфигурация PCR7	Привязка невозможна
Папка Windows	C:\Windows
Системная папка	C:\Windows\system32
Устройство загрузки	\Device\HarddiskVolume1
Язык системы	Россия
Аппаратно-зависимый уровень (HAL)	Версия = ”10.0.14393.1378”
Имя пользователя	USER-PC\Tom

Часовой пояс	RTZ 2 (зима)
Установленная оперативная память (RAM)	8,00 ГБ
Полный объем физической памяти	7,87 ГБ
Доступно физической памяти	3,54 ГБ
Всего виртуальной памяти	12,6 ГБ
Доступно виртуальной памяти	6,82 ГБ
Размер файла подкачки	4,75 ГБ
Файл подкачки	C:\pagefile.sys

Таблица 6.1: Информация об используемой системе

Для разработки использовалась Microsoft Visual Studio Enterprise 2017 (Версия 15.3.0).