How TrickBot Malware Hooking Engine Targets Windows 10 Browsers - SentinelLabs

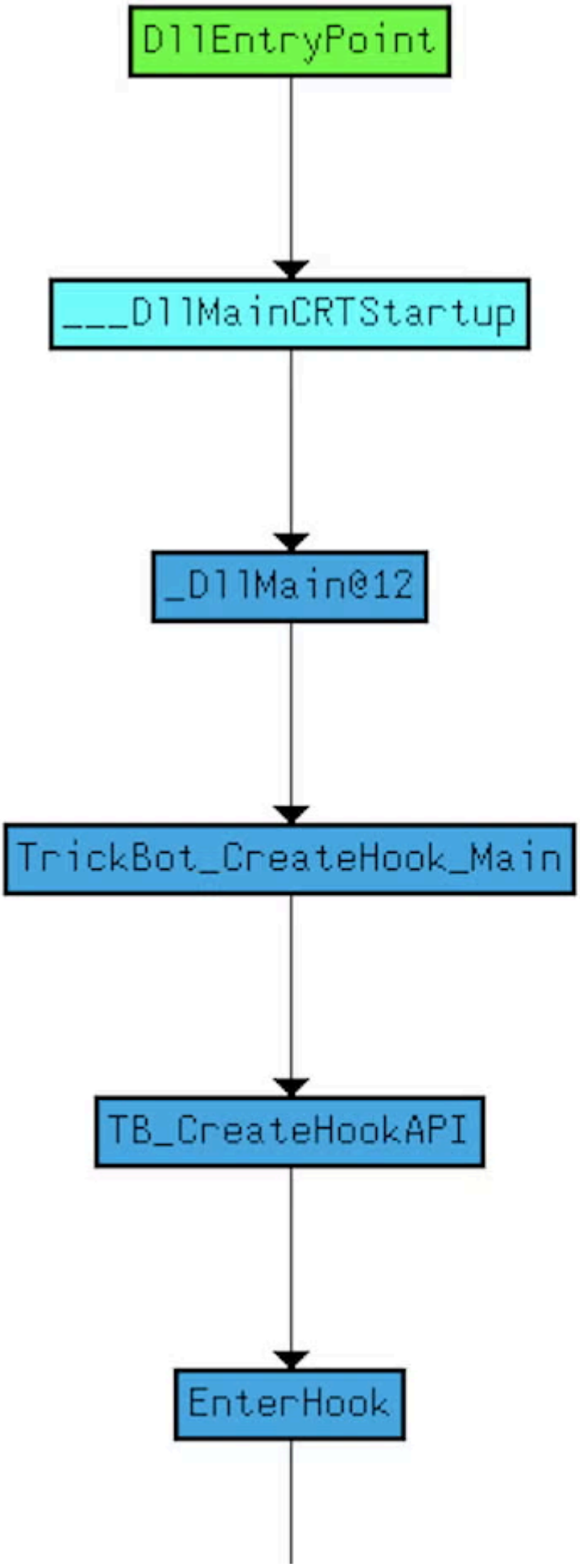
By Vitali Kremez

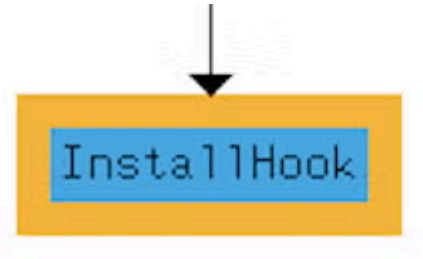
Published: 2019-10-24 · Archived: 2026-04-05 21:10:12 UTC

Vitali Kremez revealing how TrickBot's hooking engine targets Chrome, Firefox, Explorer and Edge in Windows 10.

What is TrickBot Malware? Background & Summary

TrickBot banking malware remains one of the more interesting and continually developing malware on the financial crimeware landscape. It employs multiple means and methods to exploit compromised machines of interest. The focus of this post is to cover in-depth some of its Windows 10 Microsoft Edge and other browser hooking engine functionality. We will focus on the internals, and how TrickBot leverages these browsers to set up hooks for API calls of interest. The ultimate goal of the malware browser hooking is predominantly to intercept online banking credentials before they become SSL encrypted. The stolen credentials can subsequently be used for account takeover (ATO) fraud.


DllEntryPoint
___DllMainCRTStartup
_DllMain@12
TrickBot_CreateHook_Main
TB_CreateHookAPI
EnterHook

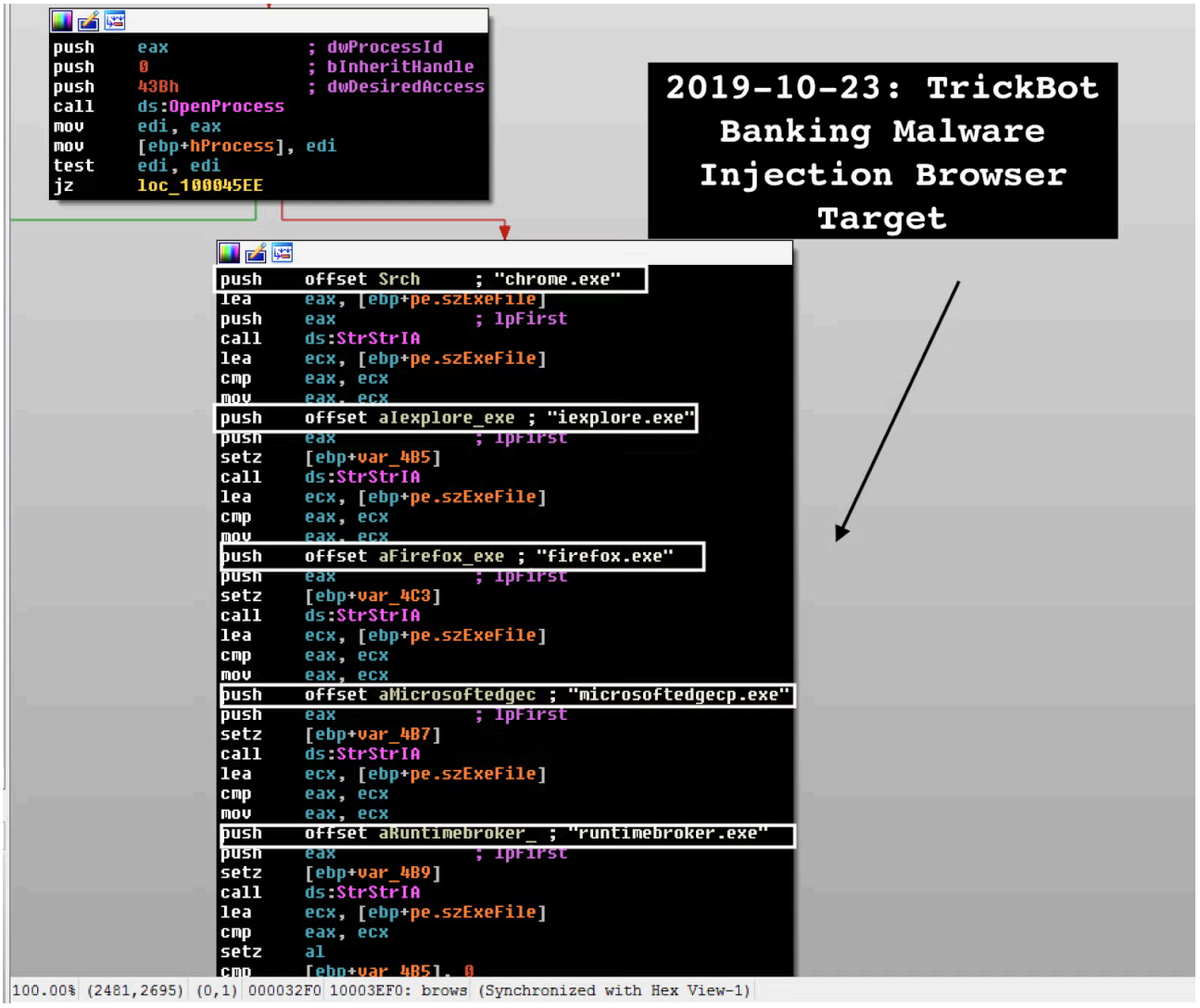


Since Windows 10 came with a new browser, “Microsoft Edge”, TrickBot operators needed their banking malware to operate on that software. To implement form-grabbing and web injections in the Windows 10 Edge browser, TrickBot’s rogue `rtlbroker` hooks the `microsoftedgecp.exe` process. Normally, `runtimebroker.exe` is the parent process of the Microsoft Edge browser on Windows 10 machines.

TrickBot Browser Process Injection Technique “Reflective Loader”

In order to hook browser functions, TrickBot malware injects the payload into the browser of choice via the so-called “ReflectiveLoader” methodology.

The TrickBot process injection function targets four browsers from Microsoft Edge to Google Chrome and one Microsoft Edge related process.



TrickBot injects the malware targeting the following processes:

- chrome.exe
- iexplore.exe
- firefox.exe
- microsoftedgecp.exe
- runtimebroker.exe

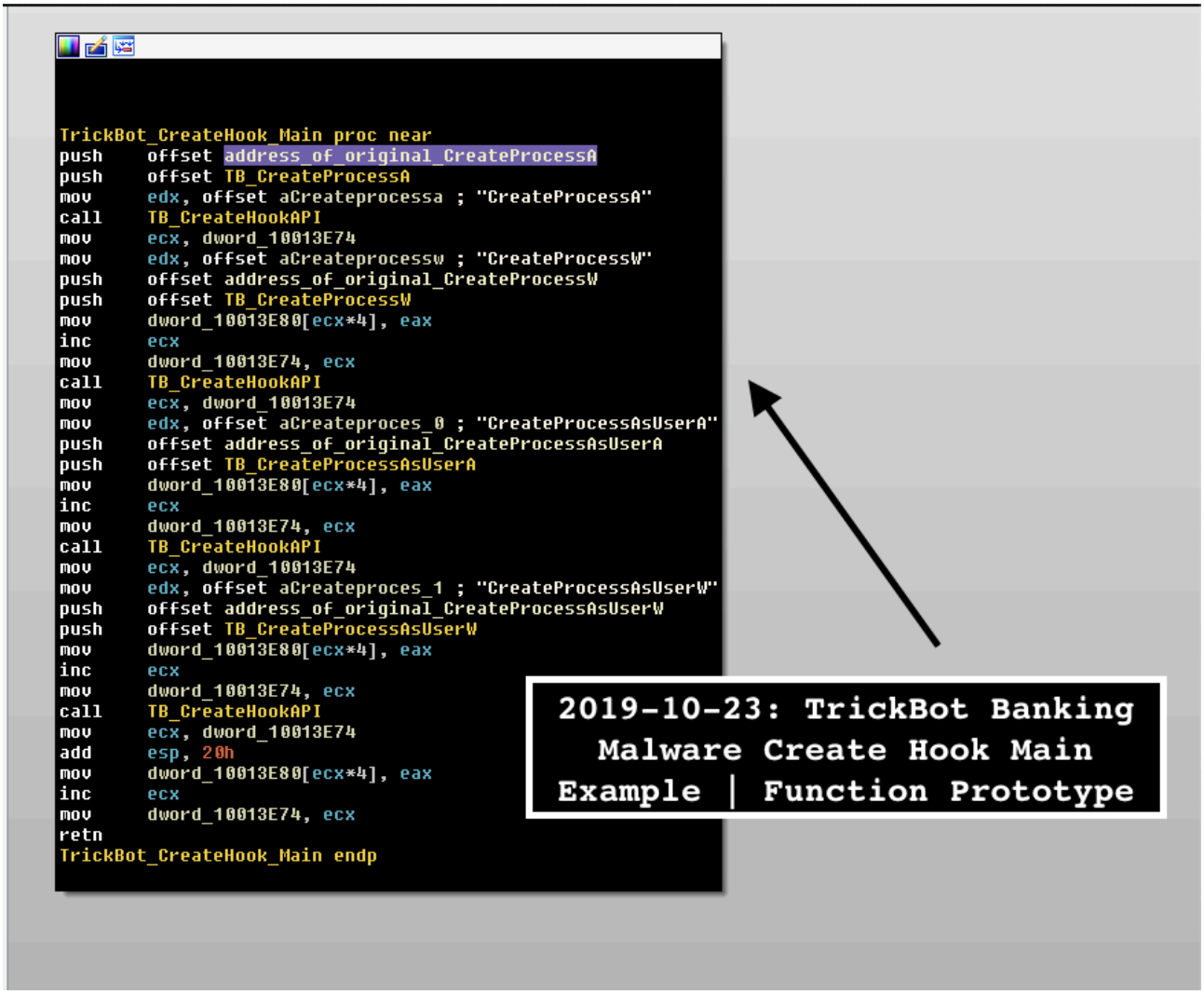
The malware also “relaxes” browser security and write changes files locally before injection occurs.

TrickBot's reflective injection works as follows:

- Open target process and allocate memory address in remote process via `VirtualAllocEx`
- Copy function `WriteProcessMemory` into the allocated memory space
- Copy shellcode `WriteProcessMemory` into the allocated memory space
- Call `FlushInstructionCache` API to make sure our changes are written right away
- Call inject `RemoteThread` function call
- Call `ResumeThread`
- Else, call undocumented API function `RtlCreateUserThread` to start execution in the remote process, using the offset address of the reflective loader function as the entry point.

TrickBot Malware Hooking Engine

When the TrickBot banker hooks the API function, it enters the new hooked one and checks to make sure the process is `microsoftedgecp.exe` while passing control to the original one when the hooked function concludes.



The basic TrickBot banking API hooking template is as follows:

```
"CreateHook_API" Function Template ->

{ int CreateHook_API(LPCSTR DLL_name, int original_function_name,

        int myHook_function,    int address_of_original_function) }
```

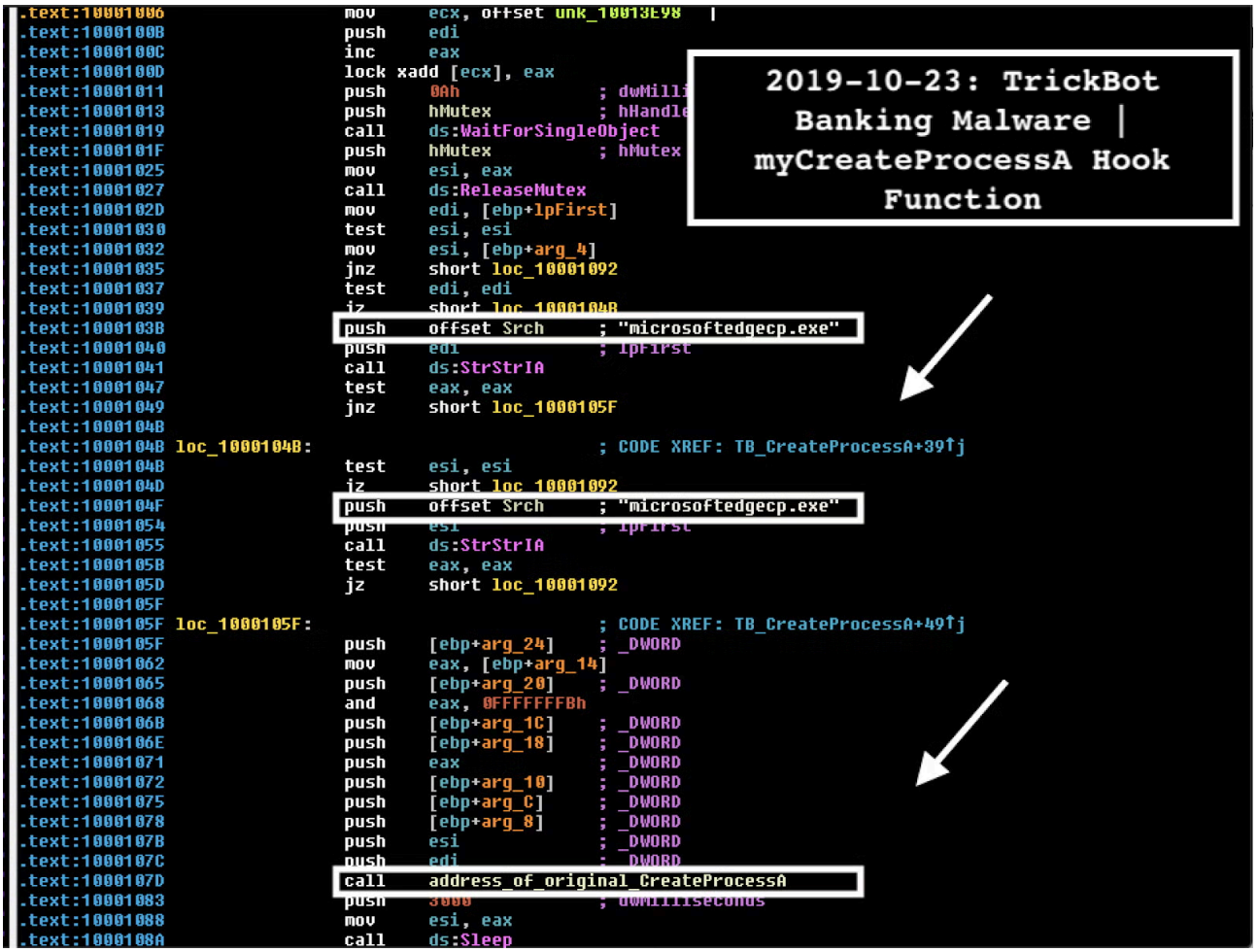
By and large, TrickBot hooking engine works via overwriting the basic API with the redirect functions with the 0xe9 opcode, which is the call for a jump with 32-bit relative offset. TrickBot uses a trampoline function and the write hook call with the `VirtualProtectEx` API to make sure that the function has the **0x40** (PAGE_EXECUTE_READWRITE) property. Additionally, it attempts to conceal detection of this hooking technique via prepending NOP and/or RETN.

The exact TrickBot hook pseudo-code is as follows:

```
signed int __cdecl TrickBot_Hook_Install(int myHook_function, int *function_address)
{
        char *original_function;
        char *current_func_id_thread;
        int v5;
        char jump_len;
        signed int result;
        SIZE_T v8;
        void *trampoline_lpvoid;
        int v10;
        int v11;
        unsigned __int8 jmp_32_bit_relative_offset_opcode;
        int relative_offset;
        DWORD flOldProtect;
        original_function = func_name;
        current_func_id_thread = func_name + 0x24;
        iter_func(func_name + 0x24, 0x90, 0x23);
        if ( function_address )
                jump_len = walker_byte_0(*(_BYTE **)(original_function + 1), (int)current_func_id_th
        else
                jump_len = 5;

        original_function[5] = jump_len;

        if ( !jump_len )
                goto LABEL_12;
                write_hook_iter((int)(original_function + 6), *(_BYTE **)(original_function + 1), (u

        if ( function_address )
                *function_address = (int)current_func_id_thread;

        relative_offset = myHook_function - *(_DWORD *)(original_function + 1) - 5;
        v8 = (unsigned __int8)original_function[5];
        trampoline_lpvoid = *(void **)(original_function + 1);
        jmp_32_bit_relative_offset_opcode = 0xE9u;

        if ( VirtualProtectEx((HANDLE)0xFFFFFFFF, trampoline_lpvoid, v8, 0x40u, &flOldProtect) )
        {
                v10 = *(_DWORD *)(original_function + 1);
                v11 = (unsigned __int8)original_function[5] - (_DWORD)original_function - 0x47;
                original_function[66] = 0xE9u;
                *(_DWORD *)(original_function + 0x43) = v10 + v11;
                write_hook_iter(v10, &jmp_32_bit_relative_offset_opcode, 5);
```

```
                VirtualProtectEx(
                        (HANDLE)0xFFFFFFFF,
                        *(LPVOID *)(original_function + 1),
                        (unsigned __int8)original_function[5],
                        flOldProtect,
                        &flOldProtect);
        result = 1;
```

For instance, TrickBot malware sets up its own custom `myCreateProcessA` function prototype after the hook on `CreateProcessA`. The idea is to catch any instance of `microsoftedgecp.exe` execution to intercept it for subsequent injection. This function ultimately returns the flow back to `CreateProcessA` after intercepting and collecting necessary process execution information.



The following four API calls being hooked are in the child Microsoft Edge via rogue `rtlbroker.dll`, allowing TrickBot operators to intercept and manipulate Microsoft Edge calls:

- CreateProcess
- CreateProcessW
- CreateProcessAsUserA
- CreateProcessAsUserW

TrickBot hooks Internet Explorer and Microsoft Edge in `wininet.dll` library API calls:

- HttpSendRequestA
- HttpSendRequestW
- HttpSendRequestExA
- HttpSendRequestExW
- InternetCloseHandle
- InternetReadFile
- InternetReadFileExA
- InternetQueryDataAvailable
- HttpQueryInfoA
- InternetWriteFile
- HttpEndRequestA
- HttpEndRequestW
- InternetQueryOptionA
- InternetQueryOptionW
- InternetSetOptionA
- InternetSetOptionW
- HttpOpenRequestA
- HttpOpenRequestW
- InternetConnectA
- InternetConnectW

The malware hooks Mozilla Firefox Browser in `nspr4.dll` library API calls:

- PR_OpenTCPSocket
- PR_Connect
- PR_Close
- PR_Write
- PR_Read

It hooks Chrome in `chrome.dll` library API calls:

- ssl_read
- ssl_write

Reference

injectDll32.dll	C546D40D411D0F0BB7A1C9986878F231342CDF8B
rtlbrokerDll.dll	0785D0C5600D9C096B75CC4465BE79D456F60594
testnewinj32Dll.dll	D5F98BFF5E33A86B213E05344BD402350FC5F7CD

Source: https://labs.sentinelone.com/how-trickbot-hooking-engine-targets-windows-10-browsers/