
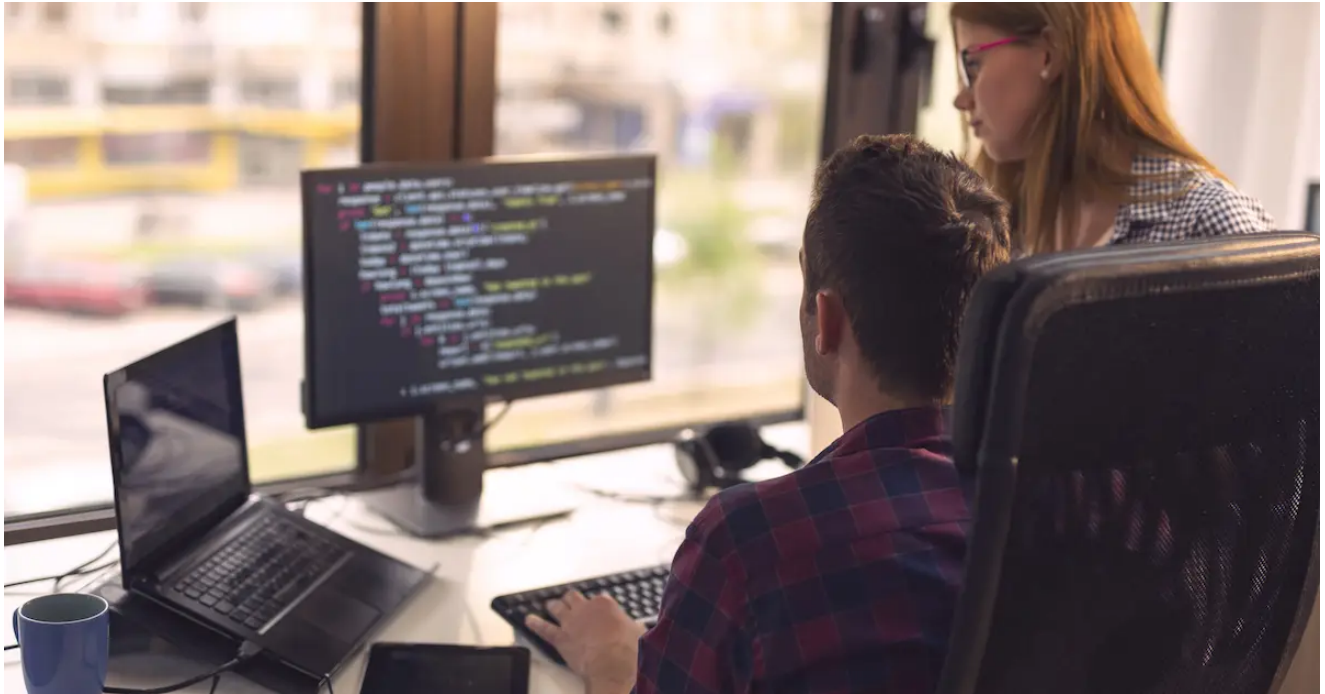


# IcedID Banking Trojan Spruces Up Injection Tactics to Add Stealth

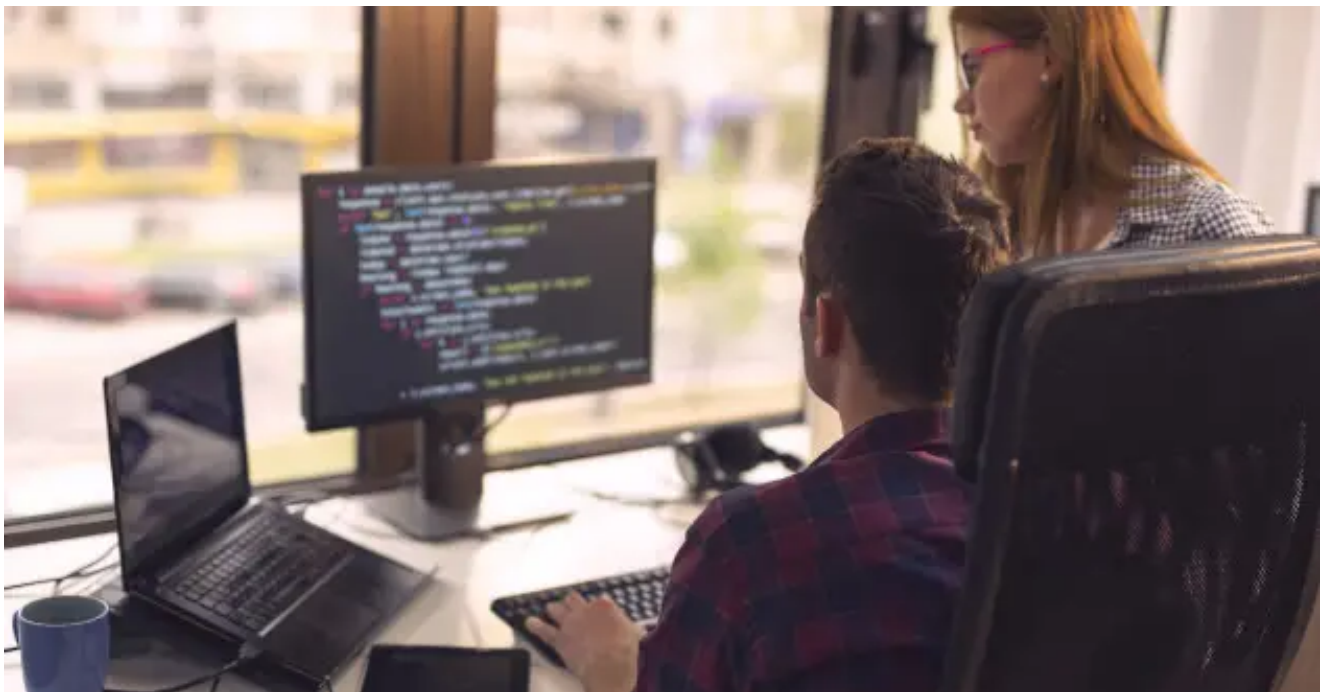
 [securityintelligence.com/icedid-banking-trojan-spruces-up-injection-tactics-to-add-stealth/](https://securityintelligence.com/icedid-banking-trojan-spruces-up-injection-tactics-to-add-stealth/)

April 4, 2019



[Home](#)&nbsp;/ [Malware](#)

IcedID Banking Trojan Spruces Up Injection Tactics to Add Stealth



[Malware](#) April 4, 2019

By [Nir Somech](#) co-authored by [Limor Kessem](#) 10 min read

IBM X-Force Research proactively researches financial cybercrime threats on an ongoing basis. In recent activity, [X-Force](#) studied some changes made to the IcedID banking Trojan that help the malware act more stealthily on infected devices.

Banking Trojans and the organized crime gangs that operate them have risen in the past decade to become one of the most prominent online threats affecting the financial sector and online service providers at large. These highly evolving malicious programs are often modular and sophisticated and, in most cases, supported by an in-house developer that keeps updating code to evade security controls.

It's not always easy to keep up with threat evolution in-house. These details about the updated injection method can help security professionals manage risk to their organization and detect updated versions of IcedID that use this injection tactic.

## No New Threads to See Here

---

Before we delve into the reversing of IcedID's modified injection, let's summarize it in a few words.

### Prior to the Change

---

Prior to the recent modifications, IcedID would write its shellcode to targeted operating system (OS) processes via `ZwWriteVirtualMemory`. IcedID used this tactic in cases where it would inject code into either OS processes or when hooking browser processes.

### After the Change

---

IcedID separated the tactics for OS process injection and browser processes. When the malware injects into an OS process, it creates a process in suspend state. It then hooks the `RtlExitUserProcess`, which is very often called to terminate userland processes. IcedID's developer counts on legitimate processes to call on this now-infected process, which will make it jump to the shellcode and run it via a seemingly legitimate flow of events that would not get flagged by most controls.

When IcedID injects to a browser process, it hooks the `NtWaitForSingleObject` function, again by using `ZwWriteVirtualMemory`.

## A Closer Look Under the Hood

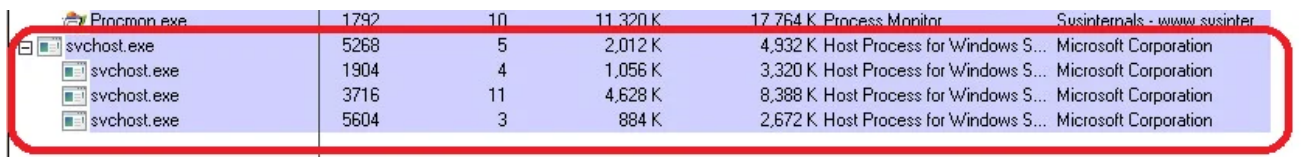
---

To demonstrate how the threat works, let's begin when the IcedID payload launches its execution routine on a target device. The malware starts by creating a `svchost.exe` process instance in suspend state and injects its shellcode into that process. After the injection, `svchost.exe` resumes and the payload terminates itself.

The newly malicious svchost.exe process writes the payload to the %ProgramData% folder with a globally unique identifier (GUID)-like folder naming convention (the GUID is a 128-bit integer number used to identify resources; the term GUID is generally used by developers working with Microsoft technologies). It uses [a-z] [A-Z]{0-9} and the payload, and a name convention of [a-z] {8-9}.exe. The value and file name are generated differently on each operating system running IcedID. However, the name will be the same upon subsequent executions on the same host.

This process also creates a scheduled task so that it will run every time the system reboots to allow IcedID to maintain persistence on the infected device.

The malicious svchost.exe creates three additional svchost.exe subprocesses and injects its shellcode into each one of them. We'll explain the injection process in further detail in the following section.



Procmon.exe	1792	10	11,320 K	17,764 K	Process Monitor	Susinternals - www.susinter
svchost.exe	5268	5	2,012 K	4,932 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	1904	4	1,056 K	3,320 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	3716	11	4,628 K	8,388 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	5604	3	884 K	2,672 K	Host Process for Windows S...	Microsoft Corporation

Figure 1: Three svchost.exe instances injected with IcedID shellcode.

At this point, with four svchost.exe processes in total, these instances will remain active until the device is shut down at some point. After a reboot, these processes will be tasked with monitoring the OS for the launch of a browser application. The malware will then attack the browser's process as well and inject it with shellcode.

## Two Injection Methods

IcedID injects its shellcode into browser processes with the goal of eventually hooking the process and allowing the malware to begin intercepting and manipulating the victim's online activity.

The malware uses API hooking to inject and execute its shellcode via different legitimate threads running on the victim's device. Let's look at that tactic more closely:

- IcedID allocates memory in a target process using the NtAllocateVirtualMemory function.
- The malware writes the shellcode into the target process without suspending that process. To do that, IcedID uses the ZwWriteVirtualMemory or NtWriteVirtualMemory functions.
- Next, it hooks a frequently called API that's commonly used by the target process.

There are two different API hooking scenarios we observed IcedID using:

1. Injecting into a web-browser via svchost.exe; and

## 2. Injecting into svchost.exe via its parent process.

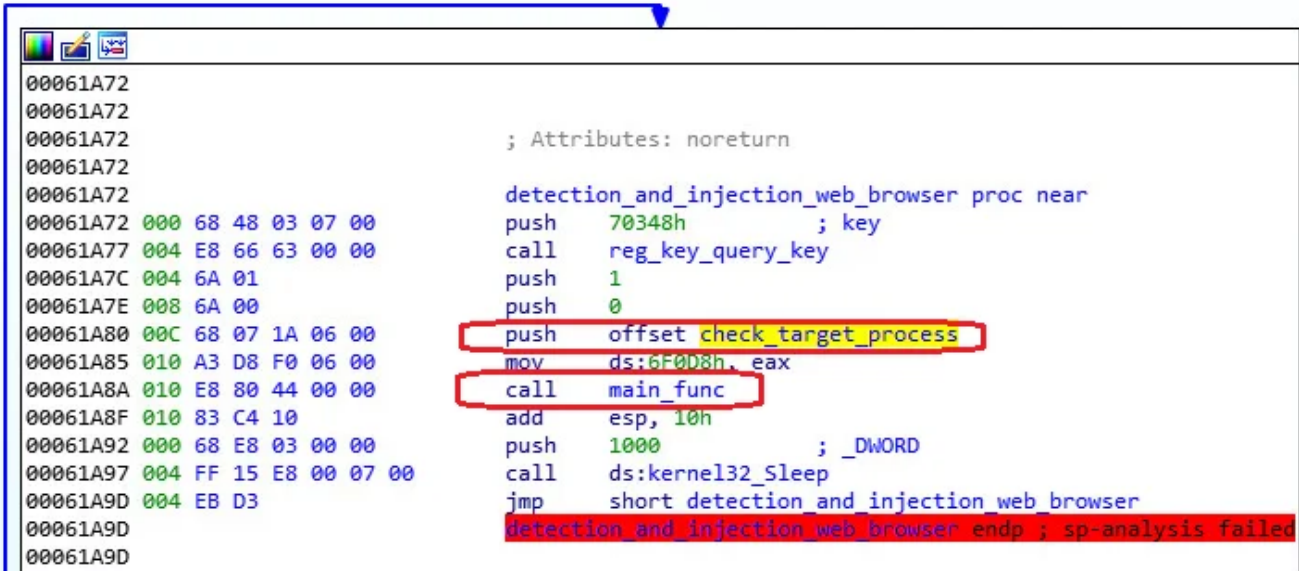
The hooked API is called by the targeted process and the shellcode is thus executed.

To wipe its traces, the shellcode fixes the patched library's function that was used to execute it so that tools that detect hooks and rootkits won't show any trails of that patch. This injection technique can evade security mechanisms that would otherwise detect suspicious API calls, such as the createRemoteThread (CRT), asynchronous procedure calls (APC) and other common malware tactics.

Remotely setting up a hook in a target process without suspending the target process first can cause unexpected behavior. This can easily result in a crash since two threads can execute the code of the API at the same time.

## Deeper Into the Browser Process Injection

The first injection tactic is from a svchost.exe process to a web browser. To begin, the malware needs a detection mechanism to identify that the infected user has launched a browser application. To do that, IcedID takes a snapshot of all the running processes and scans them for browser processes.

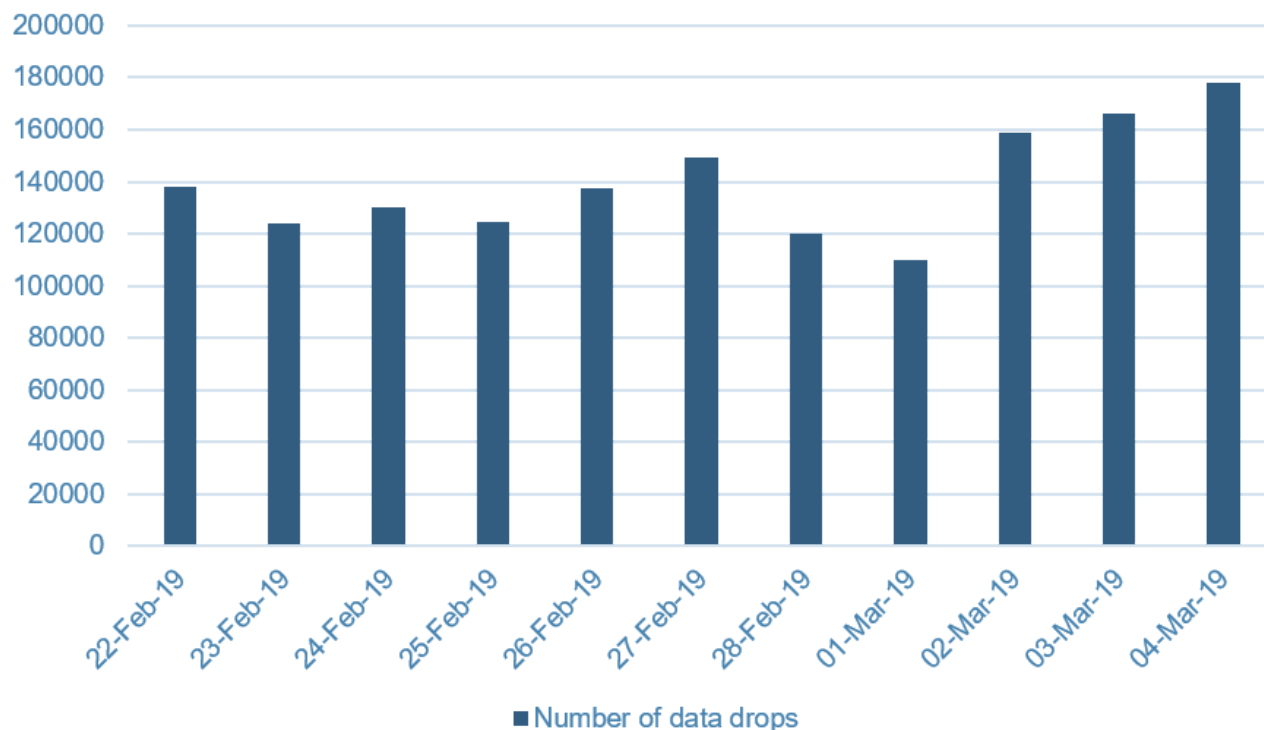


```
00061A72
00061A72
00061A72
00061A72
00061A72
00061A72 000 68 48 03 07 00
00061A77 004 E8 66 63 00 00
00061A7C 004 6A 01
00061A7E 008 6A 00
00061A80 00C 68 07 1A 06 00
00061A85 010 A3 D8 F0 06 00
00061A8A 010 E8 80 44 00 00
00061A8F 010 83 C4 10
00061A92 000 68 E8 03 00 00
00061A97 004 FF 15 E8 00 07 00
00061A9D 004 EB D3
00061A9D

; Attributes: noreturn
detection_and_injection_web_browser proc near
push 70348h ; key
call reg_key_query_key
push 1
push 0
push offset check_target_process
mov ds:6F008h, eax
call main_func
add esp, 10h
push 1000 ; _DWORD
call ds:kernel32_Sleep
jmp short detection_and_injection_web_browser
detection_and_injection_web_browser endp ; sp-analysis failed
```

Figure 2: IcedID scanning for running browser processes.

The scan creates an infinite loop that calls on IcedID's function main\_func. One of that function's parameters is a pointer to another malware function: check\_target\_process. Viewing the details of check\_target\_process reveals that it gets the process ID (PID) of the svchost.exe it runs inside. Next, it calls NtQuerySystemInformation to fetch a list of all processes currently running on the device.



*Figure 3: IcedID fetching process list to scan for open browser applications.*

This scheme keeps running in a loop over all the operating system's processes to allow the malware to detect browser activity. When it encounters a potential browser process, IcedID checks it via the `check_target_process`.

This last function uses two parameters to check whether a process is of interest or not:

1. The target process' ID; and
2. The target process' name.

The function encodes the process' name and compares it with hardcoded strings the developer created, representing the names of browsers IcedID can inject into.

- 25% - China
- 21% - Taiwan
- 14% - Russia
- 8% - India
- 7% - Brazil
- 5% - USA
- 4% - Vietnam
- 3% - Iran
- 3% - Ukraine
- 3% - South Korea
- 2% - Mexico
- 5% - Others

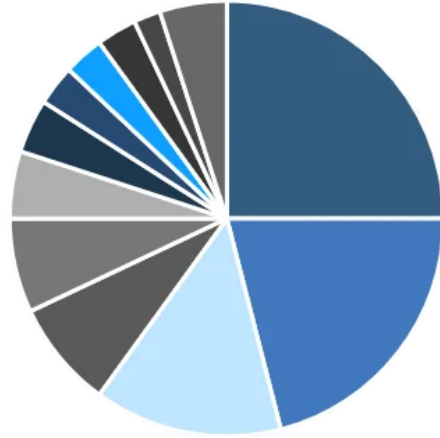


Figure 4: IcedID checking for browser processes to inject its shellcode into.

To compare the process' name to the preconfigured, hardcoded list, the malware calls the resolveProcessName function. That function receives the target process' name as a parameter, decodes it and creates a hash of the result. That hash is what's being compared to a list of precomputed hashes that translate into iexplore.exe, firefox.exe and chrome.exe.

The hash codes for the targeted web browsers are:

- Firefox.exe — 1EACD83D;
- Iexplore.exe — 0D31A30C7; and
- Chrome.exe — 0FA7442ED.

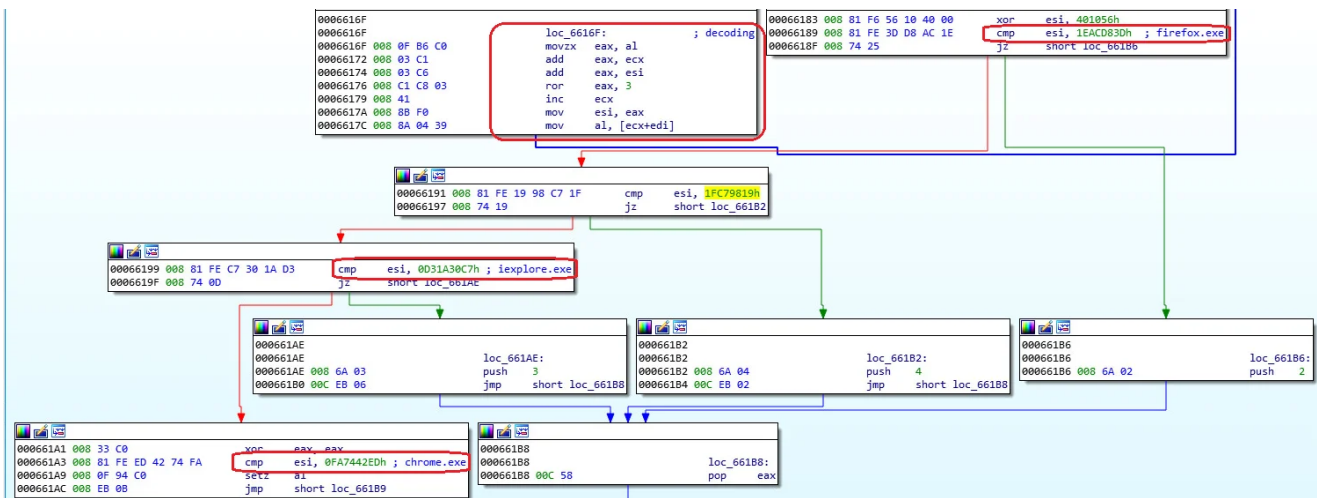


Figure 5: IcedID checking for popular browser processes to inject shellcode into.

If there's no match, the function returns to its scanning routine, looking for other browser processes.

The next step for the malware is to call `decode_target_process_event_name`. This function receives the targeted process' PID as a parameter and generates an event name for the web browser process that's about to be injected with IcedID's shellcode.

This part of the injection mechanism is in place to compute the event name that the shellcode will create after the injection process so that it indicates to the malware that a specific web browser's process has already been injected.

Now, after a match has been found and the validation of the target process is complete, the actual injection process begins by calling the function `inject_target_process`. Here, again, the parameter is the target process' PID.

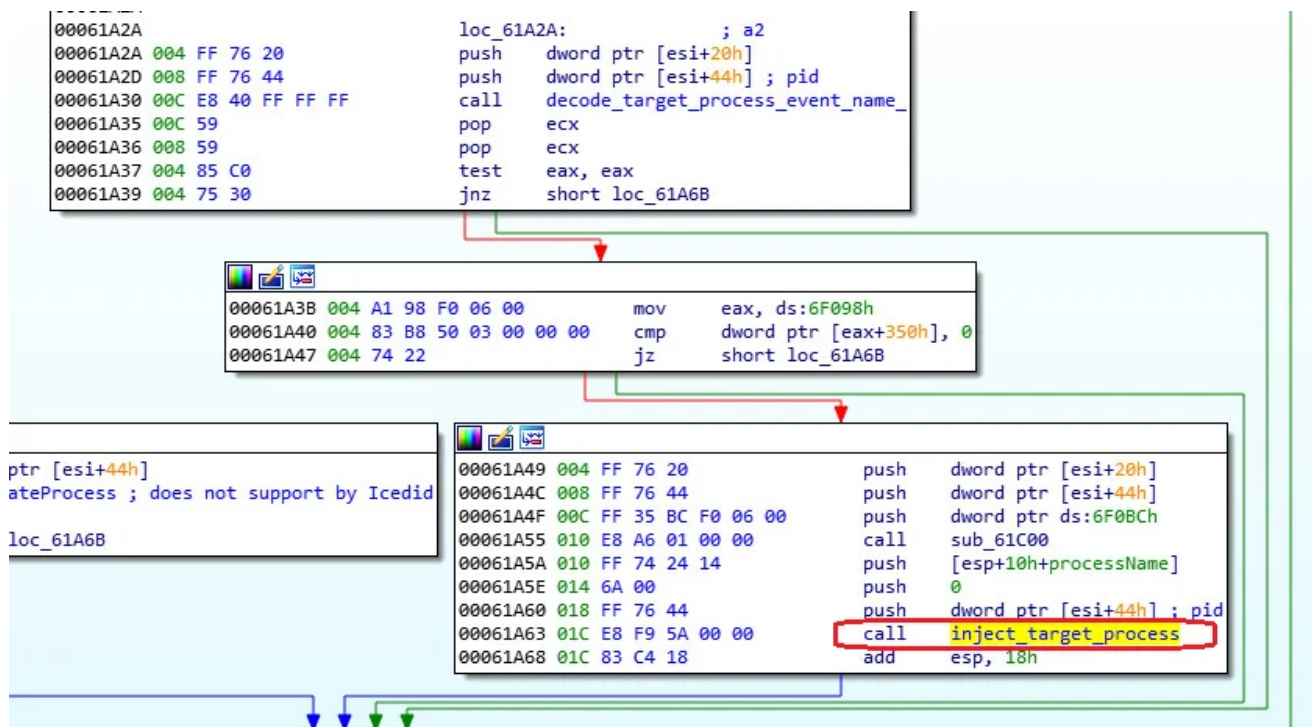


Figure 6: IcedID validates and begins injection of shellcode into browser process.

The `inject_target_process` function receives a handle to the target process using the `OpenProcess` API. It then checks whether the process is either 32-bit or 64-bit to prepare the correct code version for injection.

Next up, the function that's responsible for the injection starts by allocating memory in the target process. We can see the memory allocation call and then writing of the shellcode into the target process using three Windows API functions:

1. `NtAllocateVirtualMemory`;
2. `ZwProtectVirtualMemory`; and
3. `ZwWriteVirtualMemory`.

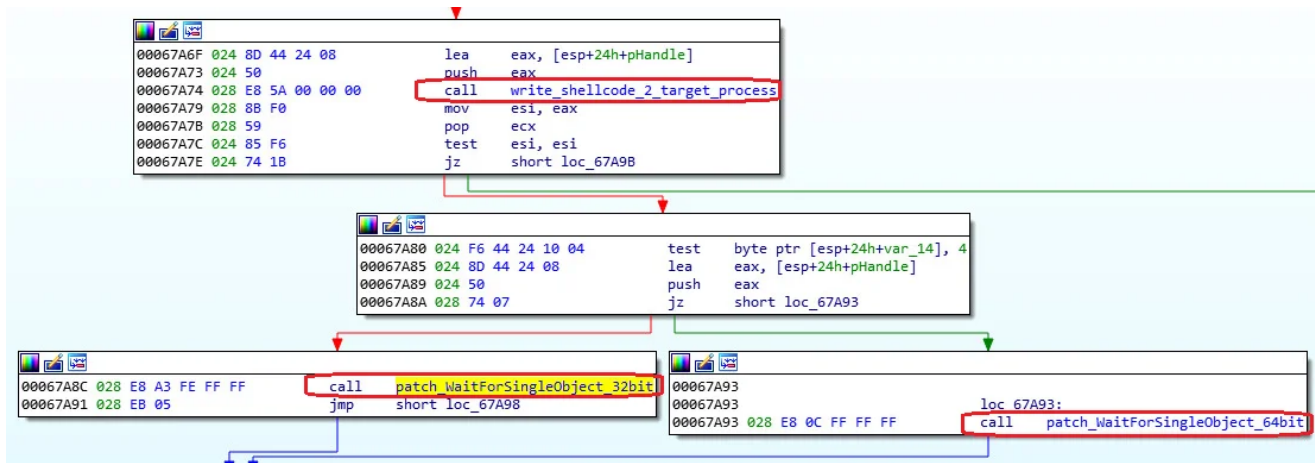


Figure 7: IcedID injects shellcode into targeted browser process.

After it writes the shellcode into the process, IcedID calls to the next function for patching, depending on which architecture applies:

- patch\_NtWaitForSingleObject\_32bit; or
- patch\_NtWaitForSingleObject\_64bit

To patch the NtWaitForSingleObject function and make it jump to the shellcode the next time it is called, IcedID begins by modifying the page protection status of that function using ZwProtectVirtualMemory. Then, the malware applies the hook using ZwWriteVirtualMemory, and ends with switching the protection flag back to its normal permission state. From this point on, subsequent calls to the NtWaitForSingleObject function will jump to the malware's injected shellcode and execute it.

The right-hand part of the image below shows the malware hooking the NtWaitForSingleObject function on the target process, in this case firefox.exe. The left part of the image shows the targeted process with the newly hooked function.



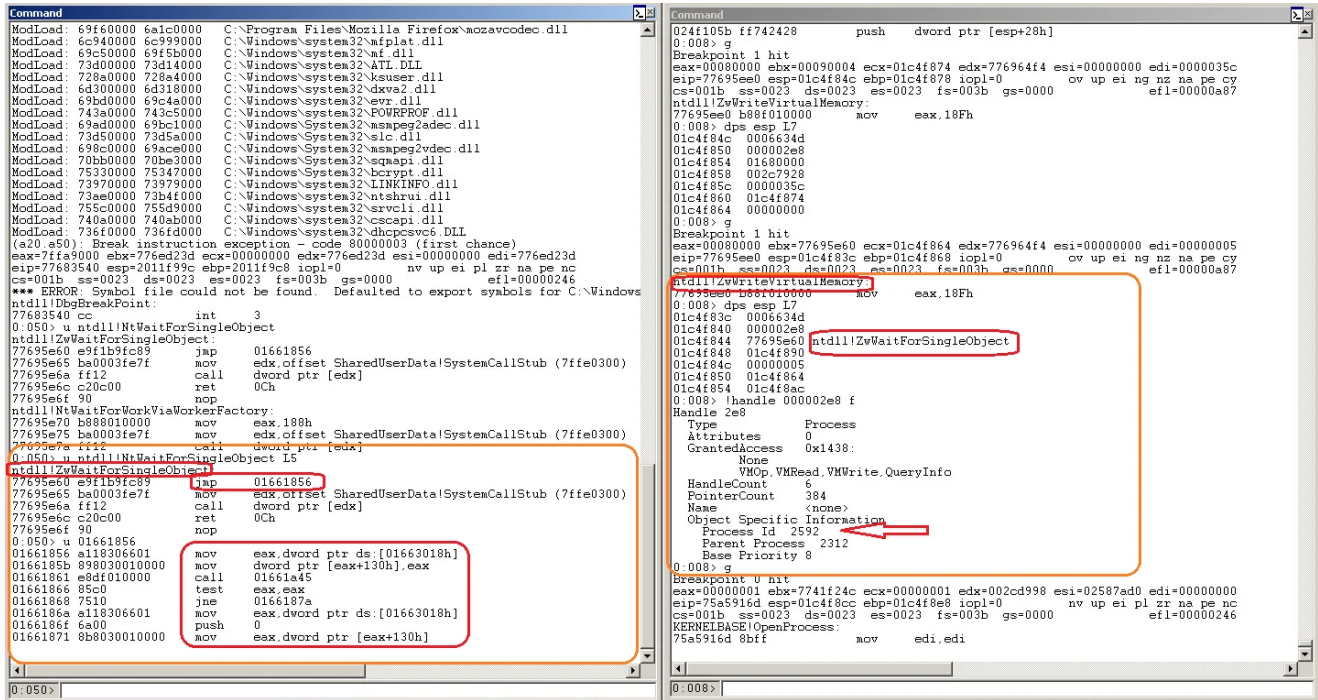


Figure 8: Before and after IcedID's browser process hooking.

The hooking process does not stop here; it is a perpetual loop that continues for as long as the device is infected with IcedID. If the user opts to launch a different browser, the malware will hook it as well and enable itself to intercept and interfere with the victim's online banking activities on any of the three browsers it can hook.

## A Closer Look at IcedID's Shellcode

Let's take a closer look at the shellcode that IcedID injects into browser processes. At the starting point, the newly hooked browser process calls on the `NtWaitForSingleObject` function. The injected shellcode runs and fixes the hook to make sure the shellcode leading to the main function will only be called once and not loop back repeatedly, especially since `NtWaitForSingleObject` is a common API and will be called many times. This also wipes the traces of the hook, which can make it harder for researchers to find.

The image below shows the starting address of the shellcode that is executed when the call for `NtWaitForSingleObject` is launched by a legitimate thread from the hooked process.

```

A1 18 30 66 01          mov     eax, ds:dword_1663018 ; function table
89 80 30 01 00 00      mov     [eax+130h], eax
E8 DF 01 00 00        call    fix_WaitForSingleObject_patch
85 C0                  test    eax, eax
75 10                  jnz     short loc_166187A
A1 18 30 66 01          mov     eax, ds:dword_1663018
6A 00                  push   0
8B 80 30 01 00 00      mov     eax, [eax+130h]
FF 50 38              call    dword ptr [eax+38h] ; RtlExitUserProcess

loc_166187A:          ; CODE XREF: seg000:01661868↑j
E8 17 00 00 00        call    build_IAT
85 C0                  test    eax, eax
74 05                  jz     short loc_1661888
E8 F0 02 00 00        call    main_func

loc_1661888:          ; CODE XREF: seg000:01661881↑j
A1 18 30 66 01          mov     eax, ds:dword_1663018
8B 80 30 01 00 00      mov     eax, [eax+130h]
FF 60 20              jmp     dword ptr [eax+20h] ; ZwWaitForSingleObject

```

Figure 9: Starting address of the shellcode executed when the call for NtWaitForSingleObject is launched.

The function fix\_NtWaitForSingleObject will uninstall the hook from NtWaitForSingleObject in the browser process, reinstating the original code that was there before the malware hooked it. Next, the shellcode will install the hooks inside the web browser's process and then call NtWaitForSingleObject to continue executing the browser's process.

```

Command
ModLoad: 739d0000 73a82000 C:\Windows\system32\DUI70.dll
ModLoad: 72d00000 72d4e000 C:\Windows\system32\actxprxy.dll
ModLoad: 6ec30000 6ec65000 C:\Program Files\Mozilla Firefox\mozavutil.dll
ModLoad: 69f60000 6a1c0000 C:\Program Files\Mozilla Firefox\mozavcodec.dll
ModLoad: 6c940000 6c999000 C:\Windows\system32\mfplat.dll
ModLoad: 69c50000 69f5b000 C:\Windows\system32\mf.dll
ModLoad: 73d00000 73d14000 C:\Windows\system32\ATL.DLL
ModLoad: 728a0000 728a4000 C:\Windows\system32\ksuser.dll
ModLoad: 6d300000 6d318000 C:\Windows\system32\dxva2.dll
ModLoad: 69bd0000 69c4a000 C:\Windows\system32\evr.dll
ModLoad: 743a0000 743c5000 C:\Windows\system32\POWRPROF.dll
ModLoad: 69ad0000 69bc1000 C:\Windows\System32\msmpeg2adec.dll
ModLoad: 73d50000 73d5a000 C:\Windows\System32\slic.dll
ModLoad: 698c0000 69ace000 C:\Windows\System32\msmpeg2vdec.dll
ModLoad: 70bb0000 70be3000 C:\Windows\System32\sqapi.dll
ModLoad: 75330000 75347000 C:\Windows\System32\bcrypt.dll
ModLoad: 73970000 73979000 C:\Windows\system32\LINKINFO.dll
ModLoad: 73ae0000 73b4f000 C:\Windows\system32\ntshrui.dll
ModLoad: 755c0000 755d9000 C:\Windows\system32\srvccli.dll
ModLoad: 740a0000 740ab000 C:\Windows\system32\csccapi.dll
ModLoad: 736f0000 736fd000 C:\Windows\system32\dhcpcsvc6.DLL
(a20.530): Break instruction exception - code 80000003 (first chance)
eax=77f96000 ebx=776ed23d ecx=00000000 edx=776ed23d esi=00000000 edi=776ed23d
eip=77f96354 esp=1ac8fb3c ebp=1ac8fb68 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SYSTEM32\ntdll.dll -
ntdll!DbgBreakPoint:
77683540 cc          int     3
0:051> .ntdll!NtWaitForSingleObject
ntdll!ZwWaitForSingleObject:
77695e60 b887010000    mov     eax,187h
77695e65 ba0003fe7f   mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
77695e6a ff12        call   dword ptr [edx]
77695e6c c20c00      ret     0Ch
77695e6f 90          ncp

ntdll!NtWaitForWorkViaWorkerFactory:
77695e70 b888010000    mov     eax,188h
77695e75 ba0003fe7f   mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
77695e7a ff12        call   dword ptr [edx]

```

Figure 10: lcedID restores the NtWaitForSingleObject function code to its original content.

Examining the shellcode in WinDbg, a commonly used Windows debugger, we can see that NtWaitForSingleObject has indeed been restored to its original code.

One of the browser functions hooked by IcedID is Ws2\_32:connect. This function redirects the victim's browsing traffic to the now-malicious svchost.exe process, allowing the malware to identify data the attacker wishes to receive, such as credentials, payment card details, etc. The data is exfiltrated to the attacker's command-and-control (C&C) sever.

We can see that the malware function hook\_connect gets the same three parameters as the connect function. First, it checks for the address family, browser type and port number and uses the information to determine whether to call the original connect function and exit or continue with the hook.

In the next step, the malware calls the original connect function with a new sockaddr object and new parameters: 127.0.0.1 as the IP address, AF\_INET as the family address and a new calculated port number. The traffic is ultimately redirected to the malicious svchost.exe process.

```
int __stdcall hook_connect(int socket, SOCKADDR_IN *sockaddr, int sockaddr_len)
{
    SOCKADDR_IN *original_sockaddr_1; // esi
    int v4; // edi
    signed int v6; // [esp-1Ch] [ebp-40h]
    SOCKADDR_IN sockaddr_1; // [esp+8h] [ebp-1Ch]
    sendData v8; // [esp+18h] [ebp-Ch]

    original_sockaddr_1 = sockaddr; // BrowserType
    if ( sockaddr->sin_family != AF_INET || !sub_16617FA(*(&ptrTable + 0x18), sockaddr->sin_port) )
        return original_connect(socket, original_sockaddr_1, sockaddr_len); // call to original connect func and return
    sockaddr = 0;
    WS2_32_ioctlsocket(socket, 0x8004667E, &sockaddr);
    *&sockaddr_1.sin_family = 0;
    *sockaddr_1.sin_zero = 0;
    *&sockaddr_1.sin_zero[4] = 0;
    sockaddr_1.sin_family = AF_INET;
    sockaddr_1.sin_addr.S_un.S_addr = 0x100007F; // 127.0.0.1
    sockaddr_1.sin_port = ptrTable->port_2_mask | (*&ptrTable->port_1_mask << 8);
    v4 = original_connect(socket, &sockaddr_1, 16);
    if ( v4 == -1 )
    {
        v6 = 0x2740;
    }
    else
    {
        v8.tableVal = *&ptrTable->field_350;
        v8.originalAddress = original_sockaddr_1->sin_addr.S_un.S_addr;
        v8.originalPort = _ROL2_(original_sockaddr_1->sin_port, 8);
        v8.browserType = *(&ptrTable + 24);
        WS2_32_send(socket, &v8, 12, 0);
        sockaddr = 1;
        WS2_32_ioctlsocket(socket, 0x8004667E, &sockaddr);
        v4 = -1;
    }
}

00001634 hook_connect:9 (1661634)
```

Figure 11: Reversing the hook installed on the connect function in the web browser.

This shellcode is what enables IcedID to gain some control and insight into what the victim is doing online and allows the attacker to interfere with that activity.

## Deeper Into the OS Process Injection

The second injection method IcedID uses is designed for injecting into processes of the operating system, not the browser.

This second tactic relies on the familiar method of creating a new process with the CREATE\_SUSPEND flag and then hooking the RtlExitUserProcess API. Once the new process starts executing, it will fix the function back to its original state.

This injection method is an interaction between the different svchost.exe processes and the malware's payload.

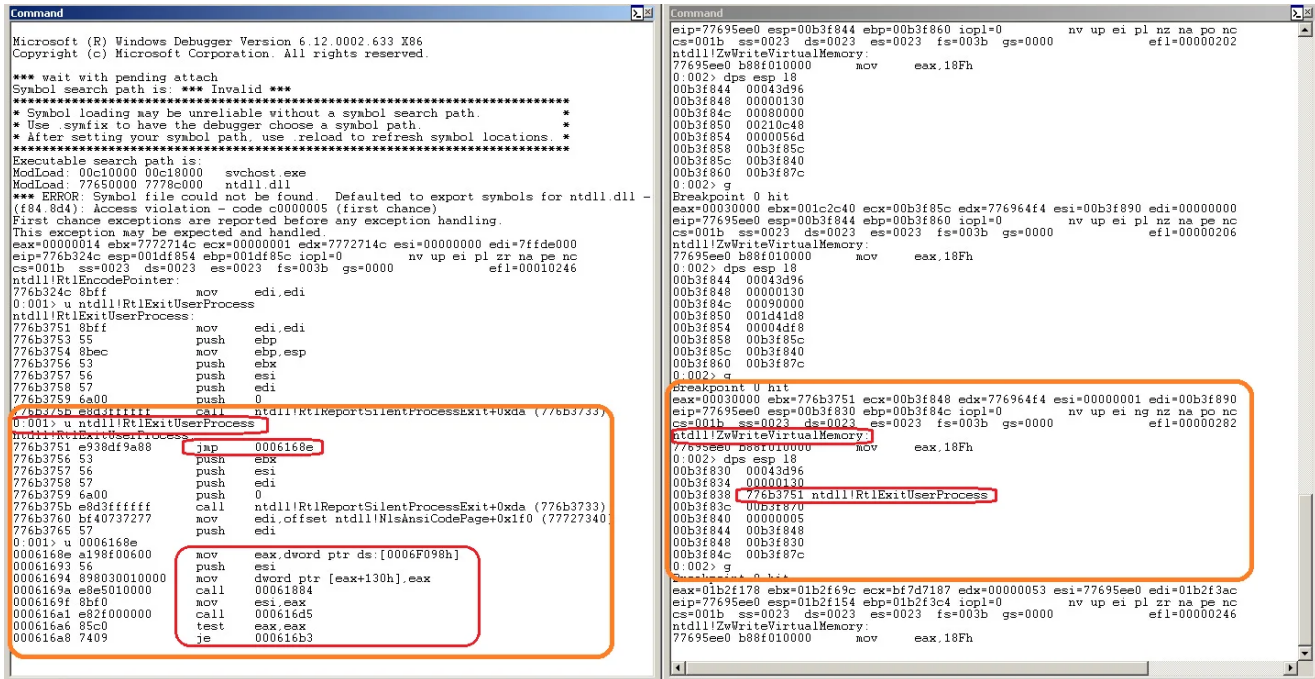


Figure 12: Before and after IcedID's hooks on targeted OS processes.

The left-hand part of the image above shows the svchost.exe process where a hook was installed on the RtlExitUserProcess API, making it jump to the shellcode the next time it is called by a legitimate thread from the svchost.exe process.

We can also see the start of the shellcode in the Interactive Disassembler (IDA), and we can see that the first function being called fixes the hook that was installed on the RtlExitUserProcess API.

```

A1 98 F0 06 00      mov     eax, ds:6F098h
56                  push   esi
89 80 30 01 00 00   mov     [eax+130h], eax
E8 E5 01 00 00     call   fix_patch_RtlExitUserProcess
8B F0              mov     esi, eax
E8 2F 00 00 00     call   Build_IAT
85 C0              test   eax, eax
74 09              jz     short loc_429D3
E8 F0 03 00 00     call   Start
85 C0              test   eax, eax
75 14              jnz   short loc_429E7

loc_429D3:          ; CODE XREF: seg000:000429C8↑j
85 F6              test   esi, esi
74 1A              jz     short loc_429F1
A1 98 F0 06 00     mov     eax, ds:6F098h
6A 00              push   0
8B 80 30 01 00 00   mov     eax, [eax+130h]
FF 50 38           call   dword ptr [eax+38h]

loc_429E7:          ; CODE XREF: seg000:000429D1↑j
; seg000:000429EF↓j
6A FF              push   0FFFFFFFFh
FF 15 E8 00 07 00   call   dword ptr ds:700E8h
EB F6              jmp    short loc_429E7

```

Figure 13: Starting address for the shellcode that's executed upon a call to `RtlExitUserProcess`.

Examining the code of the `RtlExitUserProcess` API after the removal of the malware's hook, we can see that the code was restored to its original state:

```

Command
ModLoad: 76c30000 76cf9000 C:\Windows\system32\USER32.dll
ModLoad: 75560000 755ae000 C:\Windows\system32\GDI32.dll
ModLoad: 77260000 7726a000 C:\Windows\system32\LPK.dll
ModLoad: 76e30000 76ecd000 C:\Windows\system32\NSP10.dll
ModLoad: 75750000 7575f000 C:\Windows\system32\IMM32.DLL
ModLoad: 76d60000 76e2c000 C:\Windows\system32\MSCTF.dll
ModLoad: 76d00000 76d57000 C:\Windows\system32\SHLWAPI.dll
ModLoad: 76ab0000 76b50000 C:\Windows\system32\ADVAPI32.dll
ModLoad: 75c40000 75c75000 C:\Windows\system32\WS2_32.dll
ModLoad: 75ce0000 75ce6000 C:\Windows\system32\NSI.dll
ModLoad: 726d0000 72728000 C:\Windows\system32\WINHTTP.dll
ModLoad: 72680000 726cf000 C:\Windows\system32\webio.dll
ModLoad: 73c60000 73c70000 C:\Windows\system32\NLAapi.dll
ModLoad: 71030000 71040000 C:\Windows\system32\napinsp.dll
ModLoad: 71010000 71022000 C:\Windows\system32\pnprpsp.dll
ModLoad: 74c80000 74cbc000 C:\Windows\System32\msvsocx.dll
ModLoad: 74c10000 74c54000 C:\Windows\system32\DNSAPI.dll
ModLoad: 71000000 71008000 C:\Windows\System32\winmr.dll
ModLoad: 72de0000 72dfc000 C:\Windows\system32\IPHLPAPI.DLL
ModLoad: 72dd0000 72dd7000 C:\Windows\system32\WINNSI.DLL
ModLoad: 72cc0000 72cf8000 C:\Windows\System32\fwpuclnt.dll
ModLoad: 71600000 71606000 C:\Windows\system32\rasadhlp.dll
ModLoad: 74670000 74675000 C:\Windows\System32\wshtcpip.dll
(770.8f4): Break instruction exception - code 80000003 (first chance)
eax=7ffda000 ebx=00000000 ecx=00000000 edx=771af1d3 esi=00000000 edi=00000000
eip=77144108 esp=007afaa4 ebp=007afad0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SYSTEM32\ntdll.dll -
ntdll!DbgBreakPoint:
77144108 77144108 77144108 77144108 77144108 77144108 77144108 77144108
0:004> u ntdll!RtlExitUserProcess
ntdll!RtlExitUserProcess:
7716e12b 8bfc      mov     edi,edi
7716e12d 55       push   ebp
7716e12e 8bec     mov     ebp,esp
7716e130 53       push   ebx
7716e131 56       push   esi
7716e132 57       push   edi
7716e133 6a00     push   0
7716e135 e8820000 call   ntdll!RtlExitUserProcess+0x91 (7716e1bc)

```

Figure 14: `RtlExitUserProcess` code restored to original content after uninstalling the hook.

This sums up IcedID's split injection tactics. It appears that the malware's operators are getting advice from other coders, likely those working on the TrickBot project. These modifications can make the malware's activity stealthier, yet effective.

## IcedID Keeps It Moving

---

IcedID emerged in 2017 as a modular banking Trojan with advanced capabilities to automate fraudulent transactions and control user devices to take over their bank accounts. Since its initial analysis, it has been evolving gradually over time and showing explicit collaboration with the TrickBot Trojan by ways of common distribution and feature similarity.

In August 2018, our researchers noted that IcedID had been upgraded to behave in a similar way to the TrickBot Trojan in terms of its deployment. The binary file had been modified to become smaller and no longer featured embedded modules. The malware's plugins were being fetched and loaded on demand from a remote server after the Trojan was installed on infected devices. These changes made IcedID stealthier, modular and also more similar to TrickBot.

In addition to its increased stealth level, IcedID started encrypting its binary file by obfuscating file names associated with its deployment on infected devices. Another TrickBot-inspired modification saw IcedID add event objects, which are a means to coordinate multiple threads of execution in Windows-based operating systems. IcedID began using named events to synchronize the execution between its core binary and the modular plugins it could fetch from its control server.

Although malware authors do sometimes copy from one another, these modifications were not coincidental. Even if we only looked at the fact that TrickBot and IcedID fetch one another into infected devices, that would be indication enough that these Trojans are operated by teams that work together.

X-Force data from 2018 placed IcedID in the top five most active banking Trojans on a global scale. The malware's operators have links to other key cybergangs in the threat arena, and they have been using IcedID to actively target the customers of major banks, payment card providers, e-commerce companies and cryptocurrency platforms. X-Force researchers expect this malware to continue targeting banks and payment platforms as we move into the second quarter of 2019.

- 25% - TrickBot
- 21% - Ursnif
- 13% - Ramnit
- **13% - IcedID**
- 8% - Zeus Pands
- 4% - GootKit
- 3% - Dridex
- 3% - Zeus Sphinx
- 3% - URLZone
- 3% - QakBot
- 4% - Others



Figure 15: Top most active banking Trojan families in 2019 (source: IBM Trusteer).

Nir Somech

Malware Researcher – Trusteer IBM

Nir Somech is an engineer working as part of IBM X-Force research. He specializes in researching attacks targeting the financial threat landscape. Nir holds ...

**think 2022** **IBM**

IBM Think Broadcast  
Let's think together.

Watch on demand →

