# Project 2 Report

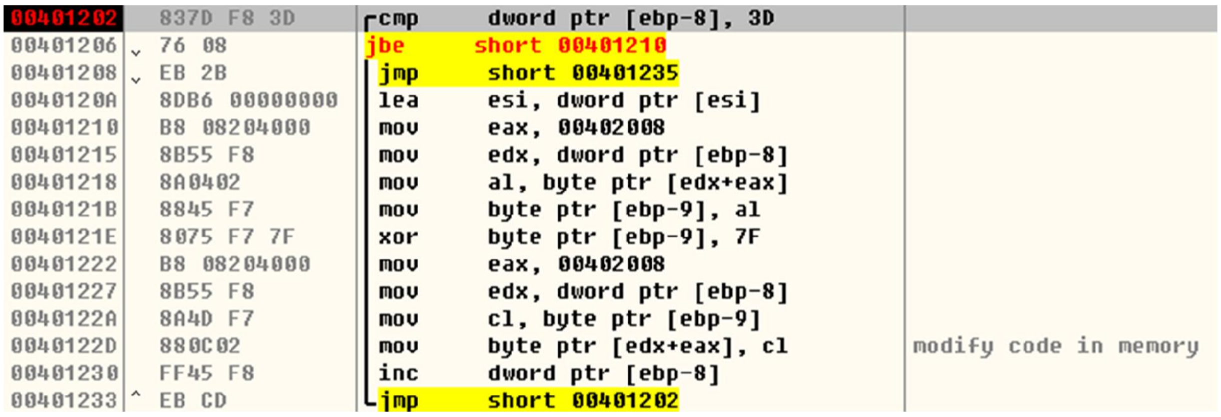
Reporter: Yucheng Liu Date: 10/10/2013

==============================

## 1. Describe in detail how the sample's runtime-generated code is created. What is the mechanism used to change data into code? What is the virtual address range of the code that performs this transformation?

This sample is self-modifying code which means this code changes itself when it runs. The mechanism here it uses is XORing data it intends to modify with 7F and changes them into runnable instructions. After this modification, the sample will also modify return address stored in stack (Question 2) such that it can call these runtime-generated codes which are actually stored in data segment.

Range of the code that performs this transformation: is **00401202 - 00401233**

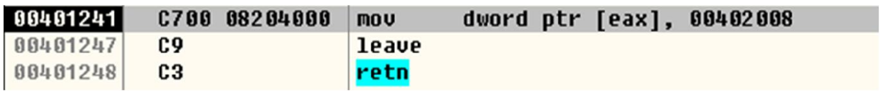


```
00401202    837D F8 3D       cmp     dword ptr [ebp-8], 3D
00401206  v 76 08            jbe     short 00401210
00401208  v EB 2B            jmp     short 00401235
0040120A    8DB6 00000000    lea     esi, dword ptr [esi]
00401210    B8 08204000      mov     eax, 00402008
00401215    8B55 F8          mov     edx, dword ptr [ebp-8]
00401218    8A0402           mov     al, byte ptr [edx+eax]
0040121B    8845 F7          mov     byte ptr [ebp-9], al
0040121E    8075 F7 7F       xor     byte ptr [ebp-9], 7F
00401222    B8 08204000      mov     eax, 00402008
00401227    8B55 F8          mov     edx, dword ptr [ebp-8]
0040122A    8A4D F7          mov     cl, byte ptr [ebp-9]
0040122D    880C02           mov     byte ptr [edx+eax], cl      modify code in memory
00401230    FF45 F8          inc     dword ptr [ebp-8]
00401233  ^ EB CD            jmp     short 00401202
```

==============================

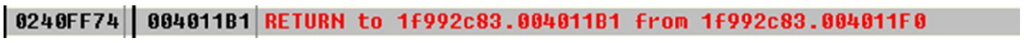
## 2. List the virtual address and type of instruction that transfers control to the dynamically generated code. Is there anything notable or unexpected about the mechanism used to transfer control to the dynamically generated code? Explain.

The following three instructions are used to transfer control,



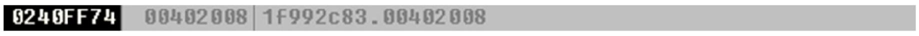
```
00401241    C700 08204000    mov     dword ptr [eax], 00402008
00401247    C9               leave
00401248    C3               retn
```

The thing that is notable is the return address is changed by these instructions. The original address that the return instruction will call is stored in this stack as follows,



```
0240FF74  004011B1  RETURN to 1f992c83.004011B1 from 1f992c83.004011F0
```

This is the return address that has been modified,



```
0240FF74   00402008|1F992c83.00402008
```

=============================

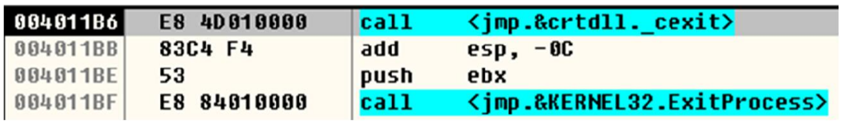
## 3. Excluding any initial jmp instructions, list the reachably executable virtual address range of the dynamically generated code. What does the code do?

Reachably executable virtual address: (Initial jmp excluded)

- **0040200A - 00402011** and
- **00402034**

We can see that two calls show up: **call 0040200A** (A) and **call 004011B6** (B)

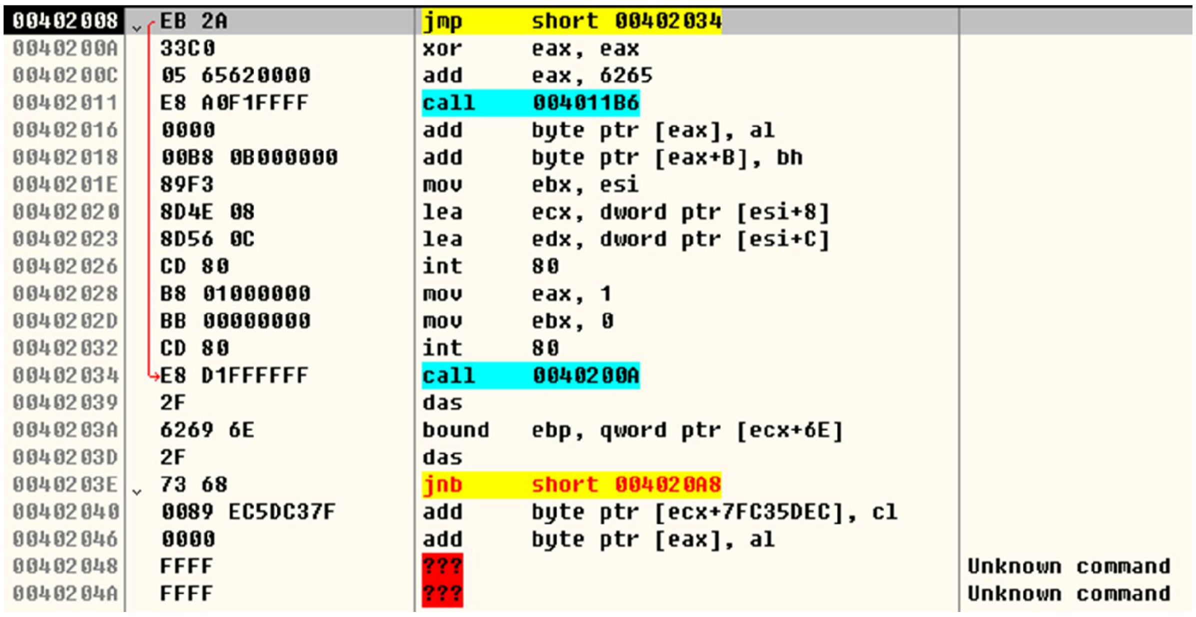
- Instruction A calls 00resets register eax, add value 6265 and makes call to Instruction B.
- Instruction B calls 004011B6 as follows, which are simply some system APIs and terminate this program.



```
004011B6   E8 4D010000    call    <jmp.&crtdll._cexit>
004011BB   83C4 F4        add     esp, -0C
004011BE   53             push    ebx
004011BF   E8 84010000    call    <jmp.&KERNEL32.ExitProcess>
```

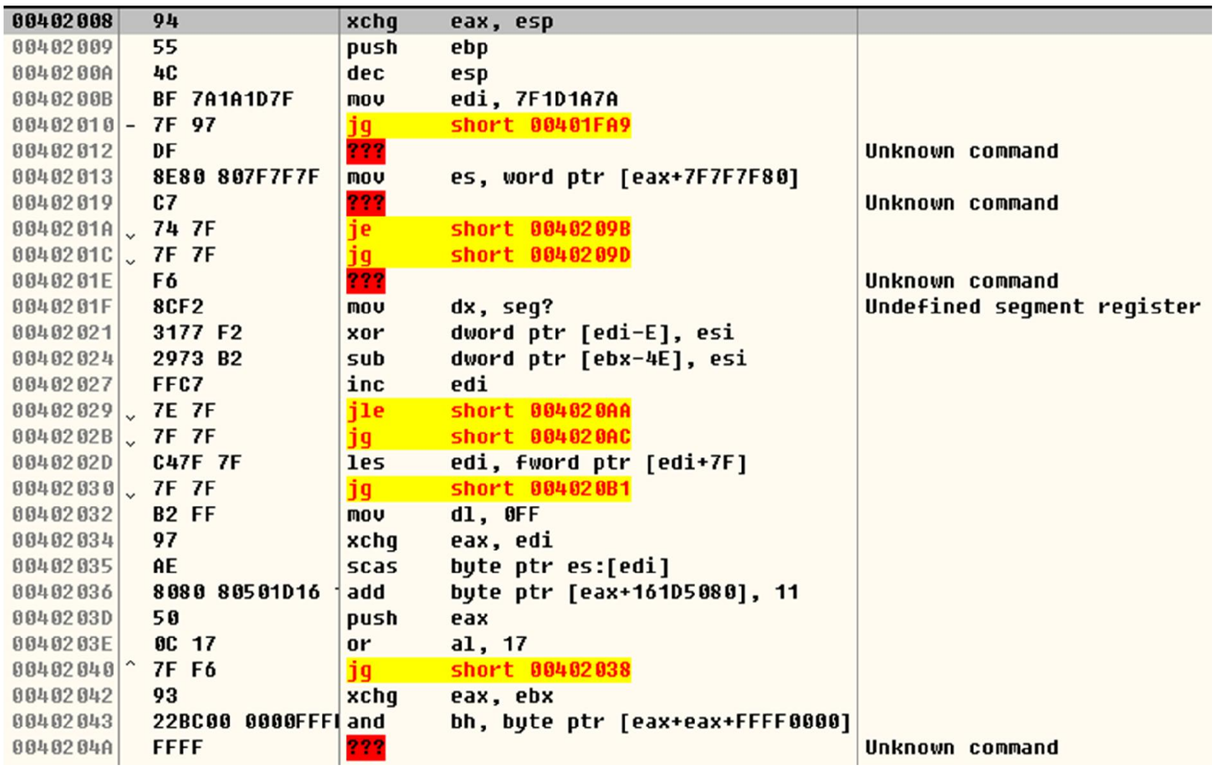
---Dynamically generated code---

*Code after Self-modification*



```
00402008    EB 2A            jmp     short 00402034
0040200A    33C0             xor     eax, eax
0040200C    05 65620000      add     eax, 6265
00402011    E8 A0F1FFFF      call    004011B6
00402016    0000             add     byte ptr [eax], al
00402018    00B8 0B000000    add     byte ptr [eax+B], bh
0040201E    89F3             mov     ebx, esi
00402020    8D4E 08          lea     ecx, dword ptr [esi+8]
00402023    8D56 0C          lea     edx, dword ptr [esi+C]
00402026    CD 80            int     80
00402028    B8 01000000      mov     eax, 1
0040202D    BB 00000000      mov     ebx, 0
00402032    CD 80            int     80
00402034    E8 D1FFFFFF      call    0040200A
00402039    2F               das
0040203A    6269 6E          bound   ebp, qword ptr [ecx+6E]
0040203D    2F               das
0040203E    73 68            jnb     short 004020A8
00402040    0089 EC5DC37F    add     byte ptr [ecx+7FC35DEC], cl
00402046    0000             add     byte ptr [eax], al
00402048    FFFF             ???                                        Unknown command
0040204A    FFFF             ???                                        Unknown command
```

**Original code of this piece of memory**



```
00402008     94               xchg    eax, esp
00402009     55               push    ebp
0040200A     4C               dec     esp
0040200B     BF 7A1A1D7F      mov     edi, 7F1D1A7A
00402010  -  7F 97            jg      short 00401FA9
00402012     DF               ???                                  Unknown command
00402013     8E80 807F7F7F    mov     es, word ptr [eax+7F7F7F80]
00402019     C7               ???                                  Unknown command
0040201A  ⌄  74 7F            je      short 0040209B
0040201C  ⌄  7F 7F            jg      short 0040209D
0040201E     F6               ???                                  Unknown command
0040201F     8CF2             mov     dx, seg?                     Undefined segment register
00402021     3177 F2          xor     dword ptr [edi-E], esi
00402024     2973 B2          sub     dword ptr [ebx-4E], esi
00402027     FFC7             inc     edi
00402029  ⌄  7E 7F            jle     short 004020AA
0040202B  ⌄  7F 7F            jg      short 004020AC
0040202D     C47F 7F          les     edi, fword ptr [edi+7F]
00402030  ⌄  7F 7F            jg      short 004020B1
00402032     B2 FF            mov     dl, 0FF
00402034     97               xchg    eax, edi
00402035     AE               scas    byte ptr es:[edi]
00402036     8080 80501D16    add     byte ptr [eax+161D5080], 11
0040203D     50               push    eax
0040203E     0C 17            or      al, 17
00402040  ^  7F F6            jg      short 00402038
00402042     93               xchg    eax, ebx
00402043     22BC00 0000FFFI  and     bh, byte ptr [eax+eax+FFFF0000]
0040204A     FFFF             ???                                  Unknown command
```

=============================

## 4. Describe what you believe are the intentions of the sample. Is it malicious?

The purpose of this sample is to hide some calls that it does not want to be discovered. After it self-modifies its code, some new instructions come out and calls 4011B6.

In this case, instruction at 4011B6 makes some system calls to terminate itself and is not malicious. However, if the instruction "call 004011B6" that this sample hides is not a system call but some piece of malicious code instead, it will cause some potential risks.