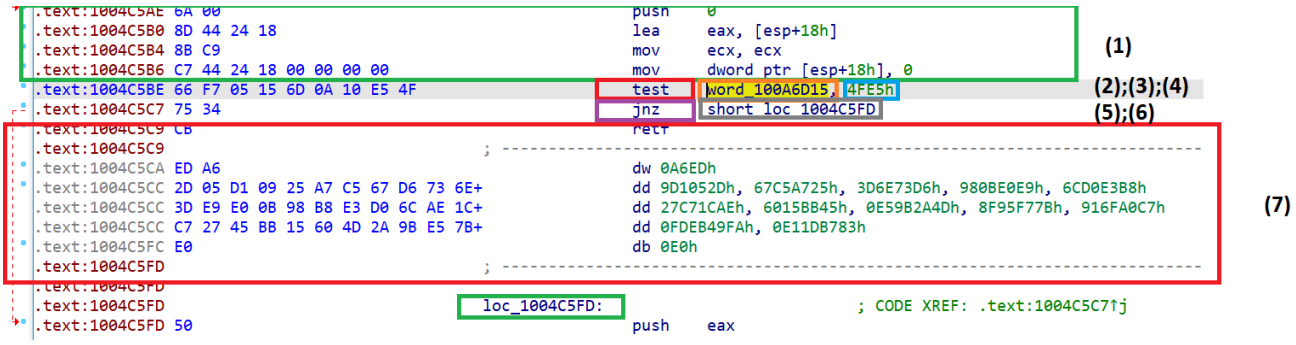
KỸ THUẬT OBFUSCATE SỬ DỤNG KẾT HỢP SO SÁNH VÀ CÁC LỆNH NHẢY CÓ ĐIỀU KIỆN TRONG MALWARE

Phần I: Kỹ thuật Obfuscate sử dụng

1. ***Mô tả***

   Luồng thực thi sẽ được rẽ nhánh bởi các lệnh compare (test/cmp) kết hợp với các lệnh nhảy có điều kiện (short jump/far jump) phía sau là các đoạn code rác được chèn vào để không cho IDA tạo Pseudocode.

2. ***Cấu trúc các phần của kỹ thuật:***



(1) Phần code thực thi bình thường.
(2) Lệnh dùng cho so sánh thiết lập các cờ cho lệnh nhảy phía dưới (gồm test/cmp)
(3) Data dùng cho việc so sánh: là các giá trị qword, dword, word và byte.
(4) Các giá trị constant tương ứng để so sánh
(5) Lệnh nhảy: short jump hoặc far jump
(6) Giá trị địa chỉ nhảy đến
(7) Mã rác
(8) Địa chỉ nhảy đến

3. ***Thuật toán***

➢ Với các lệnh nhảy ngắn: short jump thì điều kiện nhảy luôn đúng, nên câu lệnh nhảy sẽ luôn được thực thi:

```
.text:1004C3E8 F7 05 F3 56 0A 10 6A E1 92 D1                test    dword_100A56F3, 0D192E16Ah
.text:1004C3F2 71 2E                                        jno     short loc_1004C422
.text:1004C3F4 9E                                           sahf
.text:1004C3F5 F2 C9                                        repne leave
.text:1004C3F7 0F FB A5 41 CE 07 C2                         psubq   mm4, qword ptr [ebp-3DF831BFh]
.text:1004C3FE 25 93 62 A0 2D                               and     eax, 2DA06293h
.text:1004C403 9D                                           popf
.text:1004C403                              ; ---------------------------------------------------------------------------
.text:1004C404 DA EB 13 4E F4 C5 35 FA 4A 60 0E+            dd 4E13EBDAh, 0FA35C5F4h, 900E604Ah, 0E838CC91h, 8FF7F32Bh
.text:1004C404 90 91 CC 38 E8 2B F3 F7 8F CE B6+            dd 0EBA1B6CEh, 92FDB798h
.text:1004C420 EB 3F                                        db 0EBh, 3Fh
.text:1004C422                              ; ---------------------------------------------------------------------------
.text:1004C422
.text:1004C422                              loc_1004C422:                           ; CODE XREF: .text:1004C3F2↑j
```

- Với các lệnh nhảy dài (far jump) thì điều kiện nhảy luôn sai, nên câu lệnh nhảy sẽ không được thực thi, vì thế luồng thực thi sẽ không bị rẽ nhánh, đoạn code này sẽ được xem như đoạn code không làm ảnh hưởng đến luồng chương trình, ta có thể nop phần này (thực hiện ở phần II):

```
.text:1004C03A 38 A2 03 C1 8B C9                            cmp     [edx-36743EFDh], ah
.text:1004C040 03 44 24 24                                  add     eax, [esp+24h]
.text:1004C044 8D 34 45 06 00 00 00                         lea     esi, ds:6[eax*2]
.text:1004C04B F6 05 35 56 0B 10 58                         test    byte_100B5635, 58h
.text:1004C052 0F 88 C2 12 FE FF                            js      loc_1002D31A
.text:1004C058 8B CE                                        mov     ecx, esi
.text:1004C05A 89 74 24 10                                  mov     [esp+10h], esi
.text:1004C05E 83 E1 07                                     and     ecx, 7
.text:1004C061 74 0F                                        jz      short loc_1004C072
.text:1004C063 B8 08 00 00 00                               mov     eax, 8
```

- Bằng cách sử dụng thuật toán này kết hợp các đoạn code rác thì ta không thể Generate pseudocode để phục vụ cho phân tích tĩnh mã độc này.

  Để có thể tạo pseudocode, ta sẽ dùng python script patch các byte này bằng các byte NOP.

Phần II: Deobfuscate bằng python script

1. Phân tích và thiết lập thuật toán deobfuscate

a, Phân tích

Các lệnh so sánh sử dụng là test và cmp tương ứng với các opcode sau:

- test (48 F7)/cmp (48 81) + SIB byte (1 byte) + ptr qword data (4byte), qword constant value (4 byte)

- test (F7)/cmp (81) + SIB byte (1 byte) + ptr dword data (4 byte), dword constant value (4 byte)

- test (66 F7)/cmp (66 81) + SIB byte (1 byte) + ptr word data (4 byte), word constant value (2 byte)

- test (F6)/cmp (80) + SIB byte (1 byte) + ptr word data (4 byte), byte constant value (1 byte)

Các lệnh nhảy (short/far) tương ứng:
jump far: 2 byte opcode (0F + <0x80 to 0x8F>) + 4 byte address
jump short: 1 byte opcode (0x70 to 0x7F) + 1 byte value jump

b, Thuật toán

- Tìm kiếm các opcode compare (test/cmp)
- Kiểm tra xem các lệnh jump là short jump hay far jump
    - Short jump: Nop các byte từ opcode compare đến địa chỉ nhảy đến
    - Far jump: Nop các byte từ opcode compare và toàn bộ lệnh nhảy

2. Tiến hành NOP các phần obfuscate

```
# --------------------------------------------------------------------------
def nop_with_value(s, x, type_opcode):
    jump_opcode_arr = [0x70,0x71,0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C, 0x7D, 0x7E, 0x7F]
    b, ret = FindInstructions(s)
    if b:
        # executable segs only?
        if x:
            results = []
            for ea in ret:
                seg = ida_segment.getseg(ea)
                if (not seg) or (seg.perm & ida_segment.SEGPERM_EXEC) == 0:
                    continue
                results.append(SearchResult(ea))
                idc.set_cmt(ea-1,generate_disasm_line(ea-1,0),0)
                idc.set_cmt(ea,generate_disasm_line(ea,0),0)
                value = 0
                byte_pre = get_wide_byte(ea - 1)
                if type_opcode == 1:
                    if byte_pre == 0x66:
                        value = 7
                    else:
                        value = 9
                else:
                    value = 6
                idc.set_cmt(ea+value+1,generate_disasm_line(ea+value+1,0),0)
                byte_type_jump = get_wide_byte(ea + value + 1)
                if byte_type_jump == 0x0F: #jump far
                # nop ea -> ea + value + 2 byte jump + 4 byte addr
                    for addr in range(ea, (ea + value + 2 + 4 + 1)):
                        ida_bytes.patch_byte(addr, 0x90)
                        print("Done nop addr: ", addr)
                else: # jump short
                    for op in jump_opcode_arr:
                        if op == byte_type_jump:
                            value_byte_offset = get_wide_byte(ea + value + 2) # byte sau jump
                            # nop ea -> ea + value + 1 byte jump + 1 byte offset + value_byte_offset(byte)
                            for addr in range(ea, ea + value + 1 + 1 + value_byte_offset + 1):
                                ida_bytes.patch_byte(addr, 0x90)
                                print("Done nop addr: ", addr)
        else:
            results = [SearchResult(ea) for ea in ret]
        title = "Search result for: [%s]" % s
        ida_kernwin.close_chooser(title)
        c = SearchResultChoose(title, results)
        c.Show(True)
    else:
        print(ret)
```

```
def deob():
    nop_with_value("F7",True,1)
    nop_with_value("F6",True,2)
    nop_with_value("81",True,1)
    nop_with_value("80",True,2)

deob()
```

Sau khi patch các byte thành công, có thể vẫn chưa tạo được mã giả vì lỗi tham chiếu chéo từ các phần mã rác đã bị nop, ví dụ:

```
loc_1004C63E:                           ; CODE XREF: .text:1004C619↑j
                                        ; .text:loc_1004C63E↑j
```

Ta có thể fix các phần này bằng hàm del_cref: xóa các tham chiếu lỗi này

```
def del_cref(*args) -> bool
```

Delete a code cross-reference.

del_cref(frm, to, expand) -> bool

frm: linear address of referencing instruction (C++: ea_t)

to: linear address of referenced instruction (C++: ea_t)

expand: policy to delete the referenced instruction 1: plan to delete the referenced instruction if it has no more references. 0: don't delete the referenced instruction even if no more cross-references point to it (C++: bool)

retval: true - if the referenced instruction will be deleted