Méthode avancée de détection des shellcodes polymorphiques par analyse statique

21 mars 2016

Plan

Vocabulaire

Shellcode :

- Binaire exécutable utilisé en post exploitation d'une vulnérabilité logicielle

Shellcode polymorphique :

- Shellcode chiffré auto-modifiant (procédure de décodage à l'exécution)

Analyse dynamique :

- Émulation + analyse des résultats

Analyse statique :

- Analyse directe du code binaire

Étude de l'existant

Détection des shellcodes polymorphiques :

- Une guerre sans fin entre attaquants et défenseurs
- Une solution : détection par analyse dynamique
  - → Unicorn : Next Generation CPU Emulator Framework
  - → Problème : temps d'exécution de l'émulation

```
1 ms 777 us 461 ns
```

- Idée : réduire son utilisation via une méthode d'analyse statique

La méthode

Objectif :

- Une analyse rapide du binaire

Condition :

- La connaissance des encodages les plus utilisés

Procédé :

- Étude des encodages
- Création d'une procédure d'analyse statique

En pratique

- Étape 1 : Détection par signature (Suricata) :
  → Décodeur fixe : pattern
  → Décodeur métamorphique : PCRE


```
,"packet":"ABRe+jdsAJD18f/gCABFAAAoEzxAAIAG+ev
ev":0,"signature":"Encoder : Alpha Mixed Case",
```

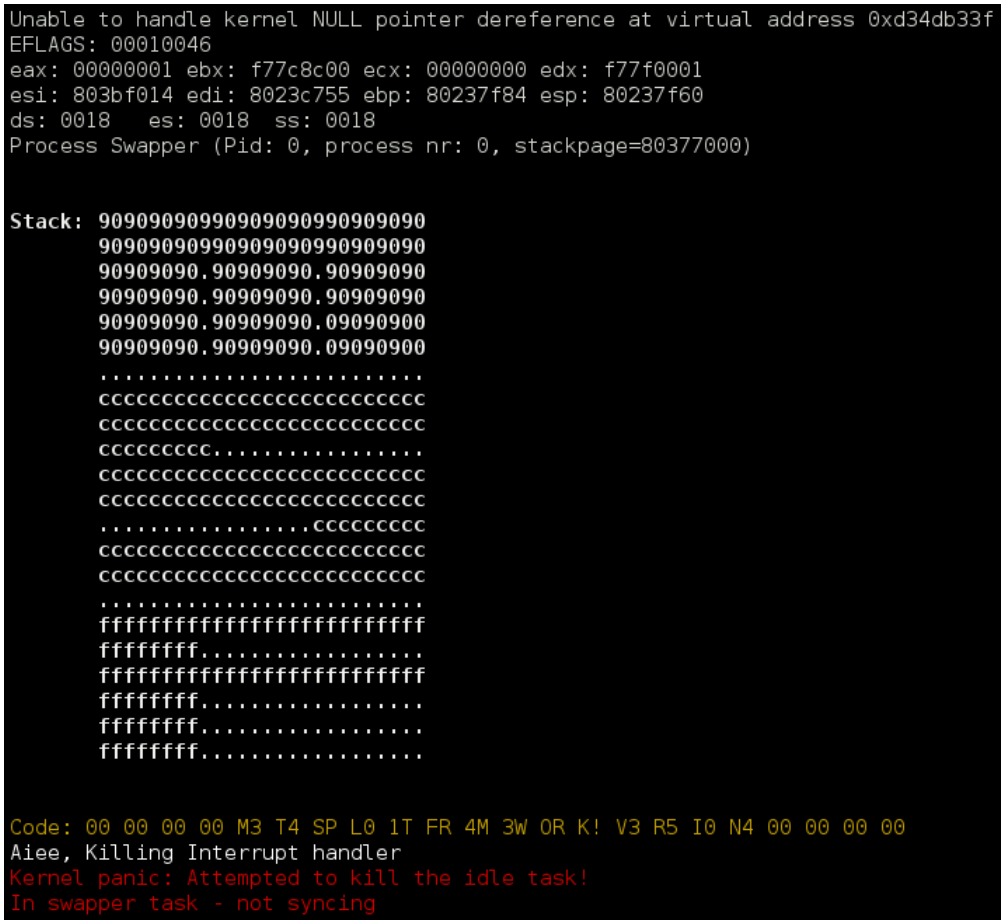

- Étape 2 : Confirmer la présence de l'encodage


```
"iteration": 7, "encoders": ["Countdown (5)", "Alpha (2)"]
```


- Étape 3 : Décoder le shellcode


```
Decoded binary :
31 C0 31 DB 6A 0F 58 68 6A 73 77 64 5B C1 EB 08 53 68 2F 70 61 73 68 2F 65
77 5B C1 EB 08 53 68 2F 73 68 61 68 2F 65 74 63 89 E3 68 41 41 FF 01 59 C1
```

BloXor


Unable to handle kernel NULL pointer dereference at virtual address 0xd34db33f
EFLAGS: 00010046
eax: 00000001 ebx: f77c8c00 ecx: 00000000 edx: f77f0001
esi: 803bf014 edi: 8023c755 ebp: 80237f84 esp: 80237f60
ds: 0018 es: 0018 ss: 0018
Process Swapper (Pid: 0, process nr: 0, stackpage=80377000)
Stack: 90909090990909090990909090
90909090990909090990909090
90909090.90909090.90909090
90909090.90909090.90909090
90909090.90909090.09090900
90909090.90909090.09090900
.....
ccccccccccccccccccccccccc
ccccccccccccccccccccccccc
cccccccc.....
ccccccccccccccccccccccccc
ccccccccccccccccccccccccc
.....cccccccc
ccccccccccccccccccccccccc
ccccccccccccccccccccccccc
.....
fffffffffffffffffffffffff
fffffff.....
fffffffffffffffffffffffff
fffffff.....
fffffff.....
fffffff.....
Code: 00 00 00 00 M3 T4 SP L0 1T FR 4M 3W OR K! V3 R5 I0 N4 00 00 00 00
Aiee, Killing Interrupt handler
Kernel panic: Attempted to kill the idle task!
In swapper task - not syncing

BloXor : Décodeur (Init)

```
0x00: e8ffffffff    call 0x04                 push eip (+0x05)
0x04: ffc0          inc eax                   instruction inutile
0x06: 5d            pop ebp                   ebp = +0x05
0x07: 6a05          push 0x5
0x09: 5b            pop ebx                   ebx = 5
0x0a: 29dd          sub ebp, ebx              ebp = +0x00 (+0x05 - 5)

0x0c: 81edb2ffffff  sub ebp, 0xffffffb2       ebp = +0x4e

0x12: 89ea          mov edx, ebp              edx = ebp
0x14: 81eafeffffff  sub edx, 0xfffffffe       edx = +0x50 (+0x4e + 2)

0x1a: bfc69b1a03    mov edi, 0x031a9bc6
0x1f: 81efc19b1a03  sub edi, 0x031a9bc1       edi = 5
```

BloXor : Décodeur (Loop)

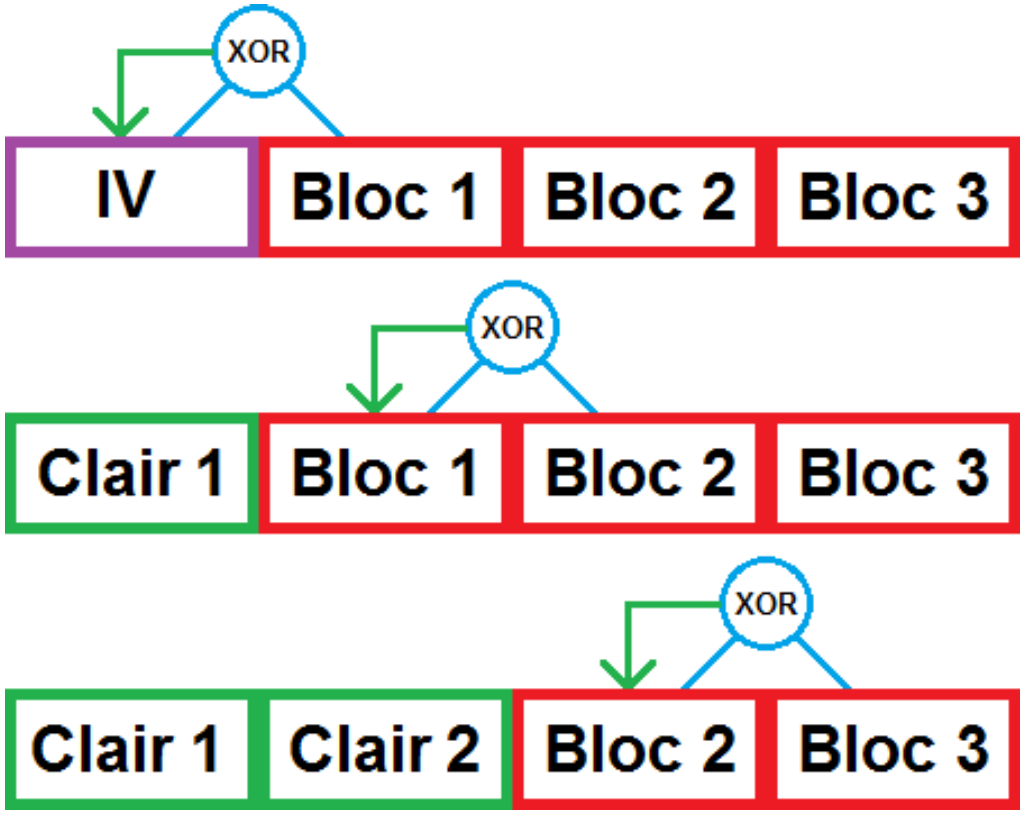
```
0x25: 8b02          mov eax, DWORD PTR [edx]
0x27: c1e010        shl eax, 0x10
0x2a: c1e810        shr eax, 0x10               eax = 0x602d
0x2d: 81eafeffffff  sub edx, 0xfffffffe         edx = +0x52 (+0x50 + 2)
0x33: 0fb77500      movzx esi, WORD PTR [ebp]   esi = 0xbe80

0x37: 89f3          mov ebx, esi                xor esi, eax
0x39: 09c3          or ebx, eax
0x3b: 21c6          and esi, eax
0x3d: f7d6          not esi
0x3f: 21de          and esi, ebx                esi = 0xdead

0x41: 6656          push esi
0x43: 668f4500      pop WORD PTR [ebp]          [+0x4e] = 0xdead
0x47: 83c502        add ebp, 0x2                ebp = +0x50 (+0x4e + 2)

0x4a: 4f            dec edi                     edi = 4
0x4b: 85ff          test edi, edi
0x48: 0f85d2ffffff  jne +0x25                   Recommencer si edi != 0
0x4e: be80                                      IV
0x50: 602d                                      ???
```

BloXor : Résumé



BloXor : Cas général

- Fonctionnement :
  - → Xor par bloc de 2 octets
    - Bloc 0 = IV (aléatoire)
    - Bloc N ^ Bloc N+1
    - Bloc N ← Résultat
- Forme :
  - → Métamorphique
  - → 12 étapes (ordre fixe)
  - → 6 registres utilisés
  - → 37 à 95 octets de long

```
(1)  GetPC + Pop reg1
(2)  Add reg1, SIZE
(3)  Set reg2, reg1
(4)  Add reg2, 0x2
(5)  Set reg3, NB_LOOP
(6)  Load reg4, [reg2]
(7)  Add reg2, 0x2
(8)  Load reg5, [reg1]
(9)  Xor reg5, reg4
(10) Store [reg1], reg5
(11) Add reg1, 0x2
(12) Loop (reg3, (6))
```

BloXor : Détection

- Travail effectué :
  - → Étude du décodeur
  - → Recherche de toutes les formes des 12 étapes
  - → Développement du processus d'analyse
- Le processus :
  - → Présence des 12 étapes
  - → Vérification de la position
  - → Recherche du nombre d'itérations
  - → Décodage du binaire

BloXor : Résultats*

Unicorn (émulation seule)

```
1 ms 777 us 461 ns
```

Analyseur statique

```
Init : 0 ms 76 us 743 ns
Bloxor : 0 ms 7 us 89 ns
Process : 0 ms 47 us 694 n
{"file": "encoded", "resul
tails": ["Bloxor(1)"], "de
InhsGZQUVOzA4nhzYBSaC8vc2h
Output : 0 ms 27 us 217 ns
Total : 0 ms 151 us 654 ns
```

```
Init : 0 ms 86 us 445 ns
Bloxor : 0 ms 6 us 81 ns
Process : 0 ms 36 us 245 ns
Output : 0 ms 219 us 188 ns
Total : 0 ms 341 us 878 ns
```

(Output dans un fichier)

* Tests effectués sur une machine peu performante

Conclusion

(+) Beaucoup plus rapide qu'une analyse dynamique

(+) On extrait toutes les informations

(+) 0% de faux-positifs / faux-négatifs

(−) L'algorithme d'encodage doit être connu et étudié