Bonjour !

Challenge SSTIC 2014

éléments de solution

Julien Perrot

4 juin 2014

Table des matières

Résumé

- **objectif** : trouver une adresse email en `@sstic.org` au sein d'un fichier `challenge` ;
- première étape : extraire un fichier (binaire ELF ARM64) d'une trace USB au format texte ;
- seconde étape :
  - reverse-engineering du binaire ARM64 $\Longrightarrow$ identification d'une machine virtuelle,
  - analyse du programme de la VM $\Longrightarrow$ déchiffrement d'un fichier `payload.bin`,
  - conditions sur le plaintext $\Longrightarrow$ inversion de l'algo de chiffrement et obtention de la clé.
- troisième étape : programme envoyé à un micro-contrôleur distant, exploitation d'une vulnérabilité $\Longrightarrow$ dump d'une zone mémoire secrète et accès à l'adresse email de validation.

Analyse de la trace USB

Contenu de la trace

```
Date: Thu, 17 Apr 2015 00:40:34 +0200
To: <challenge2014@sstic.org>
Subject: Trace USB

Bonjour,

voici une trace USB enregistrée en branchant mon nouveau téléphone Android sur
mon ordinateur personnel air-gapped. Je suspecte un malware de transiter sur
mon téléphone. Pouvez-vous voir de quoi il en retourne ?
--

ffff8804ff109d80 1765779215 C Ii:2:005:1 0:8 8 = 00000000 00000000
ffff8804ff109d80 1765779244 S Ii:2:005:1 -115:8 8 <
ffff88043ac600c0 1765809097 S Bo:2:008:3 -115 24 = 4f50454e fd010000 00000000
  09000000 1f030000 b0afbab1
ffff88043ac600c0 1765809154 C Bo:2:008:3 0 24 >
```

Conclusions

- Format usbmon
- Evènements URB $<$ messages ADB $<$ requêtes `sync`:

Résultats

```
=> LIST "/sdcard/Documents/"
40770 4096 2014-04-17 11:58:18 +0200 .
40771 4096 2014-04-17 11:53:13 +0200 ..
100660 229376 2014-03-12 16:42:15 +0100 CSW-2014-Hacking-9.11_uncensored.pdf
100660 44032 2014-03-12 16:51:01 +0100 NATO_Cosmic_Top_Secret.gpg

=> LIST "/data/local/tmp"
40771 16384 2014-04-17 13:11:23 +0200 .
40751 4096 1970-01-30 00:55:29 +0100 ..

=> STAT "/data/local/tmp/badbios.bin": mode = 0, size = 0
=> SEND /data/local/tmp/badbios.bin,33261
=> DATA 65536
=> DATA 12464
=> DONE, writing 78000 bytes to badbios.bin, mtime = 2014-04-17 13:01:02 +0200
$ shell:chmod 777 /data/local/tmp/badbios.bin

=> LIST "/data/local/tmp"
40771 16384 2014-04-17 13:11:25 +0200 .
40751 4096 1970-01-30 00:55:29 +0100 ..
100777 78000 2014-04-17 13:01:02 +0200 badbios.bin
```

Reverse-engineering de `badbios.bin`

Première exécution

```
$ file badbios.bin
badbios.bin: ELF 64-bit LSB  executable, ARM aarch64, version 1 (SYSV),
  statically linked, stripped
$ chmod +x badbios.bin
$ qemu-aarch64 badbios.bin
:: Please enter the decryption key: AAAAAAAAAAAAAAAA
:: Trying to decrypt payload...
   Invalid padding.
```

Analyse

- deux appels à `mmap` pour allouer deux zones mémoire à des adresses fixes : 0x400000 et 0x500000 ;
- « unpacking » d'une zone de données vers 0x400000 et copie d'une autre zone vers 0x500000 ;
- saute à 0x400514 et continue l'exécution.

Poursuite de l'analyse à 0x400514

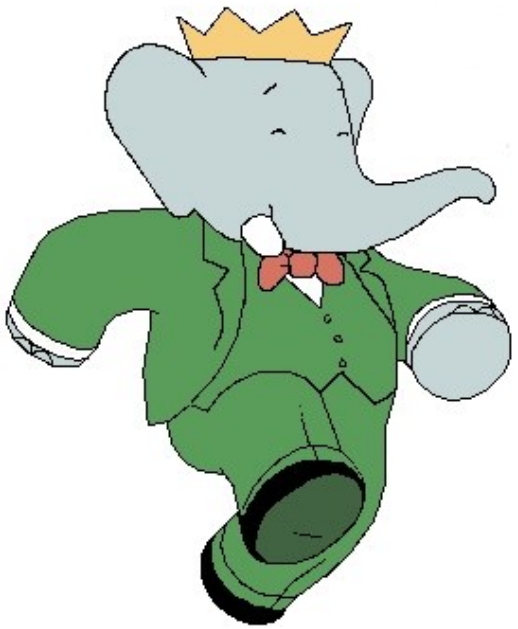
- On reverse . . . ;
- Identification d'une machine virtuelle :
  - 17 instructions,
  - 15 registres,
  - espace mémoire initial chiffré avec l'algorithme chacha et la clé 0BADB1050BADB1050BADB1050BADB105,
  - accès mémoire gérés par une MMU : les blocs accédés (en lecture / écriture) sont déchiffrés à la demande et stockés dans un cache de 32 blocs.
- Développement d'un désassembleur :
  - déchiffrement de l'espace mémoire initial (65536 octets) $\Longrightarrow$ une section de code et une section de données
  - reverse du programme exécuté par la VM $\Longrightarrow$ déchiffrement d'un bloc de 8192 octets à l'aide d'un LFSR 64 bits initialisé avec la clé fournie par l'utilisateur
  - tests sur le padding $\Longrightarrow$ OK / NOK

Linear Feedback Shift Register

```
r10 = k0; r11 = k1;
for (i = 0; i < 8192; i++) {
  r4 = 0;
  for (j = 0; j < 8; j++) {
    r9 = parity( (r10 & 0xb0000000) ^ (r11 & 1) );
    r11 = (r11 >> 1) | ( (r10 & 1) << 31 );
    r10 = (r10 >> 1) | (r9 << 31);
    r4 |= (r11 & 1 ) << (7 - j) ;
  }
  mem[32768 + i] ^= r4;
}
```

Obtention de la clé

- conditions sur le padding : n octets à 0 suivi de 0x80, avec n >= 8 $\implies$ détermination de l'état du LFSR à l'itération 8184 à partir des 8 derniers octets chiffrés : `r10 = 0x40caf153` et `r11 = 0xc32a6d56` ;
- en inversant le LFSR sur 8184 itérations, on retrouve la bonne clé : `0BADB10515DEAD11` $\implies$ déchiffrement, obtention d'un fichier `payload.bin` (archive zip).



Pwnage de micro contrôleur

Fichier fw.hex

Programme du micro contrôleur au format Intel HEX :

```
:100000002100111B2001108CC0D2201010002101F2
:10001000117C2200120FC03C20101000210111B2EF
[...]
```

Fichier upload.py

Envoie le contenu de `fw.hex` pour exécution. Mapping mémoire :

```
#    [0000-07FF] - Firmware                  \
#    [0800-0FFF] - Unmapped                  | User
#    [1000-F7FF] - RAM                       /
#    [F000-FBFF] - Secret memory area        \
#    [FC00-FCFF] - HW Registers              | Privileged
#    [FD00-FFFF] - ROM (kernel)              /
```

Exécution

```
$ python3 upload.py
System reset.
Firmware v1.33.7 starting.
Execution completed in 8339 CPU cycles.
Halting.
```

Découverte du jeu d'instructions

Injection de fautes dans le programme $\Longrightarrow$ déclenchement d'exceptions avec état des registres :

```
-- Exception occurred at 00DB: Unaligned instruction.
   r0:018C     r1:001B    r2:0000    r3:0000
   r4:0000     r5:0000    r6:0000    r7:0000
   r8:0000     r9:0000   r10:0000   r11:0000
  r12:0000    r13:EFFE   r14:0000   r15:000A
   pc:00DB fault_addr:0000 [S:0 Z:0] Mode:user
CLOSING: Unaligned instruction.
```

- En multipliant les tests, on en déduit le jeu d'instructions :
  - développement d'un désassembleur / assembleur,
  - reverse du programme correspondant à `fw.hex`,
  - identification de 3 syscalls :
    - 1 = `exit`
    - 2 = `write_stdout(addr, len)`
    - 3 = `dump_ncycles(addr)`

Dump de la mémoire

- Ecriture d'un programme utilisant le syscall 2 pour :
  - dumper la zone `[F000-FBFF]` $\Longrightarrow$ accès refusé (secret memory area),
  - dumper la zone `[FD00-FFFF]` $\Longrightarrow$ ok, obtention de la ROM (kernel).

Analyse du kernel

- Désassemblage et reverse du code récupéré ;
- Accès à l'implémentation des différents syscalls ;
- Table des syscalls à l'adresse `0xf000` ;
- syscall 3 (`dump_ncycles`) $\Longrightarrow$ primitive d'écriture à une adresse arbitraire = vulnérabilité !

Mise au point d'un exploit

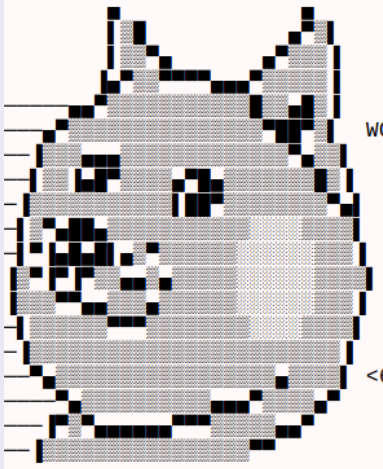
- Utilisation du syscall 3 pour corrompre la table des syscalls $\Longrightarrow$ redirection du flux d'exécution à l'adresse `0x7c0` en restant en mode kernel,
- `0x7c0` appartient à la zone mémoire correspondant au firmware envoyé $\Longrightarrow$ possibilité de mapper du code exécutable à cette adresse,
- écriture d'une boucle pour lire le contenu de la zone secrète et l'écrire octet par octet sur stdout, en s'inspirant de l'implémentation du syscall 2 (`write_stdout`).

Exploit

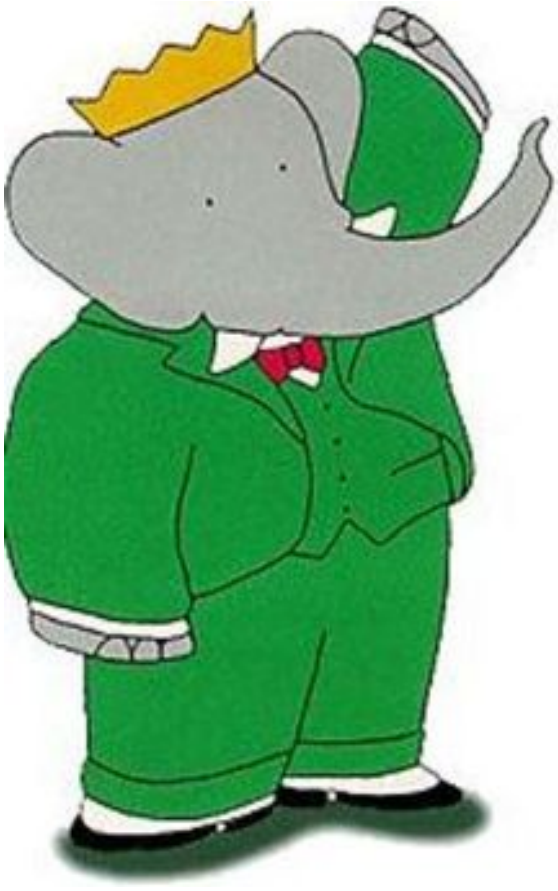
```
mov r0, 0xf004 ; table des
               ; syscalls
syscall 3
syscall 3
xor r0, r0, r0 (1000 fois)
[...]
mov r0, 0xf006 ; début de la
                 zone secrète
mov r1, 0x0000 ; offset
mov r3, 0xfc00 ; hw register
                 write_stdout
mov r4, 0x0bff ; longueur
mov r5, 0x0001 ; incrément
mov r6, 0x0000

loop_start:
mov r2, [r0 + r1]
mov BYTE PTR [r3 + r6], r2
add r1, r1, r5
sub r4, r4, r5
jmp NZ, loop_start
syscall 1
```

Résultat

```
/tmp $ cat secret.bin
```



```
WOW
  SUCH EXPLOIT
    VERY CHALLENGING
    SO OPERATIONAL
  MUCH WIN
<66a65dc050ec0c84cf1dd5b3bbb75c8c@challenge.sstic.org>
```



Merci de votre attention !