Faculté des Sciences & Techniques — Université de Limoges

Master 2nde année

TMC

Projet

Extraction de clé pour décoder du DTLS via 6LowPan dans RIOT
Étude de contre-mesure logicielle

Description du projet

Le but du projet consiste à :

- **première partie** :
  a. récupérer une clé de chiffrement utilisée dans l'envoi de message DTLS sous RIOT :
     1. dans le firmware statique utilisé pour flasher dans le micro-contrôleur ;
     2. lors de l'exécution de ce firmware sur le contrôleur ;
  b. déchiffrer des messages radio transmis en 6LowPan par Bluetooth :
     1. intercepter des paquets DTLS transmis par radio Bluetooth ;
     2. déchiffrer ces paquets avec la clé récupérée ;
- **seconde partie** :
  a. Proposer des contre mesure logicielles pour éviter la récupération de la clé :
     1. cacher la clé dans le firmware ;
     2. utiliser une fonction dynamique « sans signature » pour transformer la clé à l'exécution.

Le travail donnera lieu à la rédaction d'un compte rendu de la réalisation des deux étapes avec explications, captures d'écrans et source.

Mise en place de la plateforme matérielle et logicielle

Vous installerez les outils de développement adapté à l'architecture du NRF52833 :

Sous Arch :

```
xterm
$ sudo pacman -S arm-none-eabi-gcc
$ sudo pacman -S arm-none-eabi-binutils
```

Sous Ubuntu :

```
xterm
$ sudo apt install arm-none-eabi-gcc
$ sudo apt install
binutils-arm-none-eabi
```

Vous installerez RIOT :

```
xterm
$ git clone https://github.com/RIOT-OS/RIOT.git
```

Création du firmware

Vous utiliserez les examples suivants de RIOT :

- `/home/pef/RIOT/examples/networking/dtls/dtls-echo` : envoi des messages DTLS entre deux nrf52833 suivant deux méthodes :
  - protégé par une PSK :

    ```
    xterm
    $ make BOARD=microbit-v2 DEBUG=1
    ```

  - protégé par une clé :
    Il faut modifier les options du Makefile :

    ```
    ...
    # NOTE: If no cipher suite is selected, CONFIG_DTLS_PSK is used by default.
    # This section should be commented out if using Kconfig
    # This adds support for TLS_PSK_WITH_AES_128_CCM_8
    # CFLAGS += -DCONFIG_DTLS_PSK
    # This adds support for TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
    CFLAGS += -DCONFIG_DTLS_ECC
    ...
    ```

    Active l'utilisation d'une clé

    Vous trouverez la clé dans le fichier `tinydtls_keys.h`
- `/home/pef/RIOT/examples/networking/misc/sniffer` : interception des messages échangés en bluetooth et envoi de leur contenu sur le port série ;

Vous utiliserez wireshark pour réaliser le déchiffrement des paquets à l'aide de la clé.

Recherche de la clé dans le firmware dans le fichier « .elf »

Vous localiserez de manière précise la clé de chiffrement à l'aide de :

xterm
```
$ arm-none-eabi-objdump -h dtls_echo.elf
```

Avec radare2 :

xterm
```
$ radare2 dtls_echo.bin
[0x00000000]> /x 41 c1 cb
0x0001575e hit0_0 41c1cb
0x000157f3 hit0_1 41c1cb
[0x00000000]> px 64 @ 0x0001575e
- offset -  5E5F 6061 6263 6465 6667 6869 6A6B 6C6D  EF0123456789ABCD
0x0001575e  41c1 cb6b 5124 7a14 4321 435b 7a80 e714  A..kQ$z.C!C[z...
0x0001576e  896a 33bb ad72 94ca 4014 55a1 94a9 49fa  .j3..r..@.U...I.
0x0001577e  0000 0000 0000 f357 0100 d357 0100 b357  .......W...W...W
0x0001578e  0100 5374 6172 7420 616e 6420 7374 6f70  ..Start and stop
[0x00000000]>
```

Vous étudierez les outils suivants pour trouver la clé dans le fichier « .elf » :

a. l'outil `binwalk` pour chercher une zone d'entropie maximale normalement associée à la clé ;
b. le site web `https://binvis.io/` pour réaliser la même opération ;

Comparez si vous avez trouver correctement la clé.

Vous essaierez ensuite de trouver la clé lors de l'exécution du firmware :

- en lançant l'outil de contrôle d'exécution `OpenOCD` :

  xterm
  ```
  $ openocd -f interface/cmsis-dap.cfg -f target/nrf52.cfg
  ```

  et ensuite l'outil `gdb` :

  xterm
  ```
  $ gdb -ex "target extended-remote :3333" build/bin/microbit.elf
  ```

  N'oubliez pas de compiler le firmware en mode debug pour qu'il contienne la table des symboles.
  Vous suivrez l'exécution du firmware jusqu'à l'obtention de l'adresse où se trouve la clé et ainsi obtenir sont contenu.

Création d'une fonction dynamique d'obfuscation

Écrivez le code d'une fonction réalisant l'obfuscation de votre clé.

xterm
```
$ arm-none-eabi-gcc -Wall -Os -g -mcpu=cortex-m4 -mthumb -mfloat-abi=hard
-mfpu=fpv4-sp-d16 -nostdlib -c obfuscation.c
```

Récupérez le contenu des instructions :

xterm
```
$ arm-none-eabi-objdump -s -j .text obfuscation.o

obfuscation.o:     file format elf32-littlearm

Contents of section .text:
 0000 01440023 884200d1 70470278 d31a00f8  .D.#.B..pG.x....
 0010 013b1346 f6e7                        .;.F..

$ arm-none-eabi-objcopy -O binary -j .text obfuscation.o code_raw
$ xxd -i code_raw
unsigned char toto[] = {
    0x01, 0x44, 0x00, 0x23, 0x88, 0x42, 0x00, 0xd1, 0x70, 0x47, 0x02, 0x78,
      0xd3, 0x1a, 0x00, 0xf8, 0x01, 0x3b, 0x13, 0x46, 0xf6, 0xe7
};
unsigned int toto_len = 22;
```

Vous pourrez ensuite intégrer ce tableau dans votre firmware, puis traduire l'adresse de ce tableau en pointeur de fonction pour finalement exécuter et transformer la clé avant de l'utiliser dans dtls.

Vous proposerez **différentes méthodes d'obfuscation**.

Vous essaierez de définir votre fonction d'obfuscation en version `__attribute__((naked))` et adapterez votre code afin de la rendre utilisable.

Vous étudierez si la fonction naked ou non naked peut être détectée par un outil comme ghidra.