# Sécurité des ordivisions : exploitation de CVE-2012-5958

Frédéric Basse
frederic.basse@thalesgroup.com
contact@fredericb.info

Thales

**Résumé** Ordivision est le terme pressenti par la Commission générale de terminologie et de néologie pour dénommer les SmartTV. Outre son souci de préserver la langue française avec ce nouveau mot, la commission fait le parallèle avec les ordinateurs pour mettre en évidence les nouveaux risques de sécurité. En effet, ces ordinobjets présentent des vulnérabilités qui peuvent être exploitées. Leur couche logicielle opaque et entièrement contrôlée par le constructeur ne suffit pas à les protéger. Bien au contraire, ce verrouillage de l'intérieur rend difficile le maintien en conditions de sécurité et l'audit par des tiers.

## 1 Introduction

Les télévisions connectées, ou SmartTV, ont pris place dans les foyers et les entreprises depuis plusieurs années. Elles peuvent être équipées de connectivité Wifi/Ethernet et de capteurs audio/vidéo pour offrir des services en ligne comme le streaming multimédia, la messagerie instantanée, ou même une plateforme de téléchargement d'applications. Et inévitablement, ces innovations constituent de nouveaux vecteurs d'attaques qui élèvent les risques de sécurité pour ces objets connectés et leur entourage.

## 2 Contexte

Ces recherches ont été réalisées sur un modèle de SmartTV de 2011 avec un firmware de juillet 2013 (up-to-date), et aucune attaque matérielle n'a été réalisée. Le but est d'accéder au firmware pour en évaluer le niveau de sécurité.

Actuellement, il n'existe pas de méthode publique pour exécuter du code sur la cible. Les fichiers de mises à jour sont disponibles en téléchargement, mais ils sont chiffrés. Certaines licences des logiciels libres embarqués dans ce système, comme par exemple le noyau Linux, imposent au fabriquant de publier les modifications appliquées à leur code source. Ces sources nous

indiquent que le firmware contenu sur la mémoire flash est protégé par une couche de chiffrement.

Certaines informations sur Internet [3] indiquent qu'un port de service de type jack 3.5 est présent sur le panneau arrière. Il s'agit d'un port console qui est actif dès le démarrage du système.

```
Linux version 2.6.28.9-oslinuxR7.5 (root@lxdevenv) (gcc version
    4.2.4) #1 Thu Jun 16 23:27:36 CEST 2011
console [early0] enabled
CPU revision is: 00019651 (MIPS 24Kc)
FPU revision is: 01739300
282 MB SDRAM allocated to Linux on MIPS
512 MB total SDRAM size
Endianess     : LITTLE
```

**Listing 1.** Extrait de la sortie console

Cette console fournit des informations techniques importantes sur le système (listing 1), telles que la version du noyau Linux (2.6.28.9), l'architecture (MIPS 32bit little-endian) ou la version du compilateur GCC utilisée (4.2.4).

Un scan des ports réseau met en évidence l'existence d'un service UPnP.

## 3 Exploitation du service UPnP

Le protocole UPnP est à la base de la technologie DLNA, très présente dans les appareils audio/vidéo connectés. En janvier 2013, la société Rapid7 a publié les résultats [4] de ses recherches sur la sécurité de plusieurs bibliothèques UPnP sous licenses libres. Ces travaux ont permis de révéler plusieurs vulnérabilités critiques, notamment dans la bibliothèque libupnp [2]. Ce chapitre détaille la démarche pour exploiter une de ces vulnérabilités sur la cible.

### 3.1 Identification de la bibliothèque UPnP

L'analyse des trames réseau du protocole UPnP révèle que la bibliothèque libupnp est utilisée, dans une version antérieure à 2013. Un simple paquet UPnP, forgé pour déclencher un des débordements de pile découverts par Rapid7 (CVE-2012-5958), permet de vérifier cette supposition :

```
#!/usr/bin/python
import socket

pkt = "NOTIFY * HTTP/1.1\r\n" +\
```

```
      "HOST: 239.255.255.250:1900\r\n" +\
      "USN:uuid:schemas:device:" +\
      "A" * 512 + ":end\r\n\r\n"
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(pkt, ('239.255.255.250', 1900))
```

**Listing 2.** PoC de crash libupnp

Cela provoque un redémarrage du système, et une exception apparaît dans la console :

```
03 <2> 001990235 Exception in process 443: SIGSEGV: address not
    mapped to object
03 <2> 001990235      EPC    = 0x41414141
03 <2> 001990235      RA     = 0x41414141
```

**Listing 3.** Sortie console de la SmartTV

Le fil d'exécution a été redirigé vers l'adresse 0x41414141 qui n'est pas mappée en mémoire.

Ce firmware datant de juillet 2013 est donc toujours vulnérable face à un bug corrigé en décembre 2012. De plus, nous pouvons déduire de ce crash que le code de libupnp n'a pas de protection de pile.

### 3.2 Démarche

Pour exploiter cette vulnérabilité, nous ne disposons ni de la projection mémoire du processus, ni du code exécutable du programme ou des bibliothèques partagées. Nous ne pourrons donc pas prédire à quelle adresse mémoire sera le code arbitraire. Aussi, les techniques de ROP sont difficilement praticables dans ces conditions [1]. Mais comme nous disposons du code source de libupnp, nous avons des informations sur la structure de la pile au moment du débordement de mémoire.

Dans un premier temps nous allons extraire une projection mémoire approximative du processus vulnérable. Nous injecterons ensuite du code arbitraire en mémoire et déterminerons son adresse. Enfin, nous redirigerons le fil d'exécution vers ce code arbitraire.

### 3.3 Découverte de la projection mémoire

La vulnérabilité CVE-2012-5958 est un débordement de mémoire dans la pile de la fonction unique_server_name. Nous compilons une version vulnérable de la bibliothèque à partir des informations dont nous disposons sur la cible (version de libupnp vulnérable, du compilateur, et de l'architecture processeur).

Après désassemblage de la fonction vulnérable, nous constatons que le prologue sauvegarde les registres $s0-$s3 et $ra puis que l'épilogue les restaure.

```
.text:00004D4C                  lw      $ra, 0x158($sp)
.text:00004D50                  lw      $s3, 0x154($sp)
.text:00004D54                  lw      $s2, 0x150($sp)
.text:00004D58                  lw      $s1, 0x14C($sp)
.text:00004D5C                  lw      $s0, 0x148($sp)
.text:00004D60                  jr      $ra
.text:00004D64                  addiu   $sp, 0x160
.text:00004D64  # End of function unique_service_name
```

**Listing 4.** Epilogue de la fonction unique_server_name

Le débordement de mémoire dans la pile nous permet donc de modifier la valeur de ces registres au retour de cette fonction.

Parmi les fonctions qui appellent unique_server_name, la fonction ssdp_request_type utilise directement les registres $s0 et $s1 au retour de l'appel.

```
.text:00004DB4                  jalr    $t9 ; unique_service_name
.text:00004DB8                  move    $a1, $s0          # Evt
.text:00004DBC                  lw      $gp, 0x28+saved_gp($sp)
.text:00004DC0                  move    $a0, $s1
.text:00004DC4                  la      $t9, 0x49C0
.text:00004DC8                  or      $at, $zero
.text:00004DCC                  jalr    $t9 ; ssdp_request_type1
.text:00004DD0                  sw      $zero, 8($s0); écrit 0 @ $s0
```

**Listing 5.** Extrait de la fonction ssdp_request_type

Le registre $s1 est passé en tant que premier paramètre de la fonction ssdp_request_type1, qui le considère comme un pointeur de chaîne de caractères en lecture. Le registre $s0 est déréférencé pour écrire la valeur nulle. Après, ces registres ne sont plus utilisés jusqu'au retour de la fonction où ils seront restaurés.

La réécriture d'une adresse mémoire arbitraire dans un des registres $s0, $s1, et $ra peut entrainer un crash du processus si l'adresse n'est pas mappée ou non-accessible respectivement en écriture, lecture, et exécution.

Les crashs du processus peuvent être détectés de plusieurs façons telles que le déni du service réseau (le processus ne répond plus aux sollicitations), l'activité réseau causée par le redémarrage du processus (trames réseau émises au démarrage du service réseau) ou encore l'écriture de logs sur le port série.

Ainsi, nous pouvons déduire si une adresse donnée est mappée en mémoire et le cas échéant quelles sont ses permissions. Nous répétons cette

démarche de façon automatique afin de couvrir une partie de la mémoire du processus :

```
0x00402020-0x00532120  r-x
0x00542020-0x0091af20  rw-
0x0091b020-0x00927efc  ---
0x00928020-0x00930920  rw-
0x00942920-0x00980920+ rwx
```

**Listing 6.** Projection mémoire déduite

La stabilité des résultats obtenus induit que ces zones mémoire ne sont pas randomisées.

Nous constatons que la dernière zone mémoire est exécutable, et de taille variable selon le contexte. Nous faisons l'hypothèse qu'il s'agit du tas. Ses propriétés sont favorables pour y placer du code arbitraire.

A titre de comparaison, voici la projection mémoire réelle du processus vulnérable (SPOILER) obtenue après exploitation réussie de la cible :

```
00400000-00533000 r-xp 00000000 20:09 944        /apps/media
00542000-00649000 rw-p 00132000 20:09 944        /apps/media
00649000-0091b000 rwxp 00649000 00:00 0
00928000-00942000 rw-p 00238000 20:09 944        /apps/media
00942000-00a22000 rwxp 00942000 00:00 0          [heap]
```

**Listing 7.** Projection mémoire réelle du processus

### 3.4 Injection du code arbitraire

Nous supposons que le tas est exécutable. Comme nous disposons du code source de la bibliothèque libupnp, nous pouvons prédire quels paquets UPnP envoyés à ce processus seront alloués dans le tas. Ainsi, nous plaçons du code arbitraire dans le tas à une adresse inconnue.

### 3.5 Localisation du code arbitraire

Nous avons déjà montré que le débordement de pile permettait de modifier arbitrairement la valeur de certains registres pour le reste de l'exécution (listing 5).

La fonction ssdp_request_type1, appelée juste après la fonction vulnérable, prend en seul paramètre d'appel un de ces registres. Elle utilise ce paramètre comme un pointeur vers une chaîne de caractères et y recherche certains marqueurs. Elle retourne un code numérique correspondant au marqueur trouvé, sinon un code d'erreur.

```
enum SsdpSearchType ssdp_request_type1( IN char *cmd ) {
    if( strstr( cmd, ":all" ) != NULL )
        return SSDP_ALL;
    [...]
    return SSDP_SERROR; }
```

**Listing 8.** Code de la fonction ssdp_request_type1

Le processus répond à notre requête UPnP si et seulement si cette fonction trouve un des marqueurs.

Le débordement de pile nous donne le contrôle sur l'adresse où seront recherchés les marqueurs. Nous plaçons dans notre code arbitraire un des marqueurs reconnus. Comme nous connaissons l'adresse de début du tas, nous nous servons de cette fonction pour rechercher l'adresse de notre code arbitraire.

### 3.6 Exécution de code arbitraire

Nous sommes en mesure de placer du code arbitraire dans le tas, de trouver son adresse, et d'y rediriger l'exécution. Ainsi, nous obtenons un shell pour accéder au contenu du firmware.

Du point de vue de la sécurité, nous constatons que :

- tous les processus ont les permissions root ;
- la pile et le tas des processus sont exécutables ;
- le tas n'est pas randomisé.

## 4 Conclusion

Les SmartTV embarquent un OS et des fonctionnalités réseau semblables aux systèmes informatiques modernes. Cela nécessite d'adopter aussi leurs mécanismes de protection contre les attaques logicielles. L'absence de la plupart de ces protections sur notre cible a permis l'exploitation d'une vulnérabilité pour exécuter du code arbitraire à distance. Cela rappelle la problématique de la confiance à accorder à un équipement connecté en réseau dont nous ne pouvons gérer les mises à jour logicielles, ni auditer la sécurité.

## Références

1. Blind return oriented programming (brop). http://www.scs.stanford.edu/brop/.
2. Portable sdk for upnp devices. http://pupnp.sourceforge.net/.
3. Root my tv. http://neophob.com/2010/01/root-my-tv.
4. Rapid7. Security flaws in universal plug and play. https://community.rapid7.com/docs/DOC-2150, 2013.