HPE iLO 5 security

Go home cryptoprocessor, you're drunk !

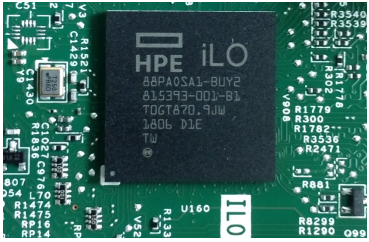
Alexandre Gazet, Fabien Périgaud & Joffrey Czarny

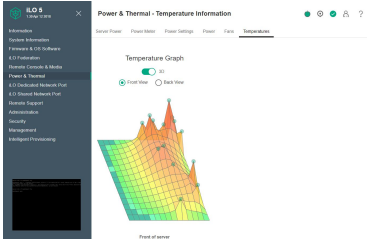
Rennes et les interwebs, 2-4 Juin, 2021

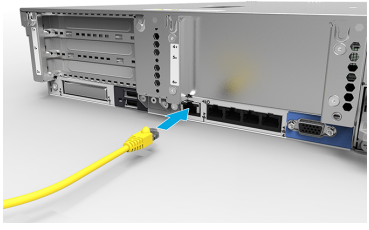
SYNACKTIV

AIRBUS

HPE Integrated Lights Out en bref







Un sujet qui nous est familier :

- Analyse en profondeur de la technologie HPE iLO4 et iLO5 [1] [2]
- `CVE-2017-12542` : exécution de code arbitraire (pré-auth)
- `CVE-2018-7078` : contournement de la vérification cryptographique de la signature des mises à jour
- `CVE-2018-7113` : contournement du secure boot (iLO5)
- Abus du mécanisme DMA pour compromettre l'host depuis l'iLO [3]
- etc.

1. https://github.com/airbus-seclab/ilo4_toolbox
2. Présentations à RECON, SSTIC, ZeroNights
3. https://github.com/Synacktiv/pcileech_hpilo4_service

- Début 2020, nouvelles versions de firmware iLO5 : **2.x**
- Blob de données à forte entropie ⇒ **mises à jour chiffrées**
- Notes d'installation :

  "Upgrading to iLO 5 version 2.10 is supported on servers with iLO 5 1.4x or later installed."

- Versions 1.4x "transitoires" **non chiffrées**

Chiffrement

Objectif(s)

- Retrouver où est implémenté le déchiffrement
- Réimplémenter le mécanisme de déchiffrement
- Mettre à jour nos outils d'analyse de firmware

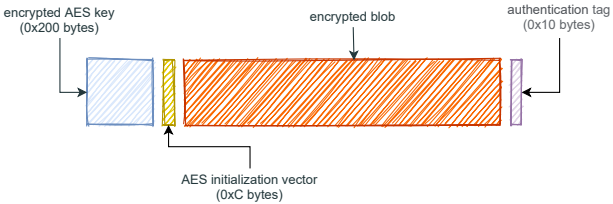
T - 0

Chaîne de boot ?

Bootloaders et kernel sont inchangés entre les versions 1.3x et 1.4x

Firmware Update Manager (FUM)

- Tâche/module en mode utilisateur
- Responsable de la vérification de la signature puis l'écriture des mises à jour sur la flash
- Découverte de la nouvelle fonctionnalité



- Primitives OpenSSL : chiffrement d'enveloppe [4]
- Chiffrement symétrique authentifié des données, AES Galois/Counter Mode (GCM)
- La clé symétrique (AES) est **scéllée** via une clé publique et un algorithme chiffrement asymétrique (RSA)
- $\Rightarrow$ Clé privée RSA requise pour "*ouvrir l'enveloppe*"

4. https://wiki.openssl.org/index.php/EVP_Asymmetric_Encryption_and_Decryption_of_an_Envelope

Clé privée RSA

```
.fum.elf.RW:0006186C 31 71 58 7A+                DCB "FSvA9nEAVQoVhRPcXA1rmHo32x5wopf2WpAd5awmUl5nxr0GmU42qf0CAwEAAQ=="
.fum.elf.RW:0006186C 52 4D 36 0A+                DCB 0xA
.fum.elf.RW:0006186C 71 6E 78 39+                DCB "-----END RSA PUBLIC KEY-----",0xA,0
.fum.elf.RW:00061B74             ; char RSA_PRIVATE_KEY[3435]
.fum.elf.RW:00061B74 2D 2D 2D 2D+RSA_PRIVATE_KEY DCB "-----BEGIN ENCRYPTED PRIVATE KEY-----",0xA
.fum.elf.RW:00061B74 2D 42 45 47+                                         ; DATA XREF: fum_decrypt_hpimage+2C↑o
.fum.elf.RW:00061B74 49 4E 20 45+                                         ; fum_decrypt_hpimage+34↑o
.fum.elf.RW:00061B74 4E 43 52 59+                DCB "MIIJrTBXBgkqhkiG9w0BBQ0wSjApBgkqhkiG9w0BBQwwHAQIjhdNQLNz8zoCAggA"
.fum.elf.RW:00061B74 50 54 45 44+                DCB 0xA
.fum.elf.RW:00061B74 20 50 52 49+                DCB "MAwGCCqGSIb3DQIKBQAwHQYJYIZIAWUDBAEqBBAEJcEYMX1ZMZFCj4/IdBSrBIIJ"
.fum.elf.RW:00061B74 56 41 54 45+                DCB 0xA
.fum.elf.RW:00061B74 20 4B 45 59+                DCB "UO2BH0h/huyP2JBd7bmEVyEhqK87PYg1FI6EuvxXekgCwG/567YPvzJZxYDvdssr"
.fum.elf.RW:00061B74 2D 2D 2D 2D+                DCB 0xA
```

- PEM base64, PKCS♯8
- Clé privée RSA protégée par une passphrase
- OpenSSL : `PEM_read_bio_RSAPrivateKey`
- ⇒ Une fonction callback fournit la passphrase

```
int __fastcall pem_password_cb(char *buf, unsigned int size, int rwflag, void *u)
{
  key_mask[0] = 0;
  key_mask[1] = 0xCE;
  key_mask[2] = 0;
  key_mask[3] = 0xD00000;
  key_mask[4] = 0x86C900;
  key_mask[5] = 0x9A0000;
  key_mask[6] = 0x700000;
  key_mask[7] = 0x190000;
  if ( size < 0x20 || rwflag == 1 )
    return 0;
  HW_SECRET[0] = MEMORY[0x1F200D8];
  HW_SECRET[1] = MEMORY[0x1F20B00];
  HW_SECRET[2] = MEMORY[0x1F20B08] & 0xFFFFFFF0;
  HW_SECRET[3] = MEMORY[0x1F20B0C];
  HW_SECRET[4] = MEMORY[0x1F21810];
  HW_SECRET[5] = MEMORY[0x1F21840];
  HW_SECRET[6] = MEMORY[0x1F21850];
  HW_SECRET[7] = MEMORY[0x1F21890];
  for ( i = 0; i < 0x20; ++i )
    buf[i] = *((_BYTE *)key_mask + i) ^ *((_BYTE *)HW_SECRET + i);
  return 0x20;
}
```

- Plage d'adresses `0x1F2xxxx` ≈ registres de configuration du SOC (System-On-Chip) iLO5 mappés dans la tâche userland.
- ⇒ Le buffer `HW_SECRET` est une clé "matérielle"

Extraction de la clé matérielle

Objectif(s)

- Exploiter une vulnérabilité sur iLO 5
- Lire les registres du SOC
- Réimplémenter le mécanisme de déchiffrement
- Mettre à jour nos outils d'analyse de firmware

T + 2 jours

- CVE-2018-7105 par Nicolas Iooss[5]
- Impacte iLO4 et iLO5
- "*Remote execution of arbitrary code, Local Disclosure of Sensitive Information*"
- Vulnerabilité de type format string dans le shell restreint SSH propriétaire, post-authentication
- Code d'exploitation disponible pour iLO4
- ⇒ Portage du premier stage de l'exploit pour iLO5

5. @fishilico : "*Add SSH exploit for CVE-2018-7105*"
https://github.com/airbus-seclab/ilo4_toolbox/commit/430bfb95

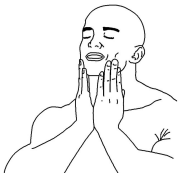
Difficultés

1. Exploit dans la tâche `ConAppCli` ⇒ secrets du SOC non mappé
2. Primitive R/W mémoire avec des contraintes :
   - Adresses avec des octets nuls interdites
   - Tout comme certains charactères spéciaux (“\n”, “\r”, etc.)

Esthète mémoire

1. Patch/hook d'un pointeur de fonction pour appeler une primitive de l'OS (≈ `mmap`)
2. Double mapping (adresses virtuelles-physiques) pour supprimer les octets nuls

```
[+] dumping iLO HW keys:
[+] MMU: memory mapping magic:
  >> patch 0x2008@0xb8264
  >> patch 0x1008@0xb824c
  >> patch 0x18@0xb818c
  >> patch 0xc00000@0xb8181
[+] command hooks:
  >> hook 0x70158@0xb0bec

>> 0xbf7fffc3@0x1f200d8
>> 0x01851c0d@0x1f20b01
>> 0x32f26410@0x1f20b08
>> 0x08000621@0x1f20b0c
>> 0x8000009f@0x1f21810
>> 0x81001012@0x1f21840
>> 0x810010dc@0x1f21850
>> 0x81001121@0x1f21890
```



Extraction de la clé matérielle via une format-string over VPN

⇒ **Firmware 2.x déchiffrés avec succès !!!**

```
0)  Secure Micro Boot 2.02, type 0x03, size 0x00008000
1)  Secure Micro Boot 2.02, type 0x03, size 0x00005424
2)           neba9 0.10.13, type 0x01, size 0x00005644
3)              neb926 0.3, type 0x02, size 0x00000ad0
4)           neba9 0.10.13, type 0x01, size 0x00005644
5)              neb926 0.3, type 0x02, size 0x00000ad0
6)   iLO 5 Kernel 00.09.60, type 0x0b, size 0x000d6158
7)   iLO 5 Kernel 00.09.60, type 0x0b, size 0x000d6158
8)                 2.10.54,  type 0x20, size 0x001dd9dc
9)                 2.10.54,  type 0x23, size 0x00f2ad0b
a)                 2.10.54,  type 0x22, size 0x004e7f2
```

Wait !!!!

- 3 images userland
- Précédement 2 images seulement : `main` (type 0x20) et `recovery` (type 0x22)
- **(double) facepalm : le nouveau type 0x23 est chiffré...**
- Le type 0x20 est maintenant minimaliste, mais **non chiffré**


MARIO
058700
×20
WORLD
1-4
TIME
200
THANK YOU MARIO!
BUT OUR PRINCESS IS IN
ANOTHER CASTLE!

Seconde couche de chiffrement

Objectif(s)

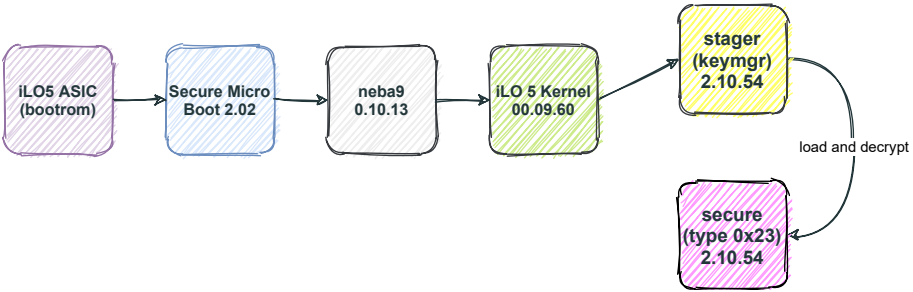
- Comprendre le second étage de chiffrement
- Réimplémenter le second étage de déchiffrement
- Mettre à jour nos outils d'analyse de firmware

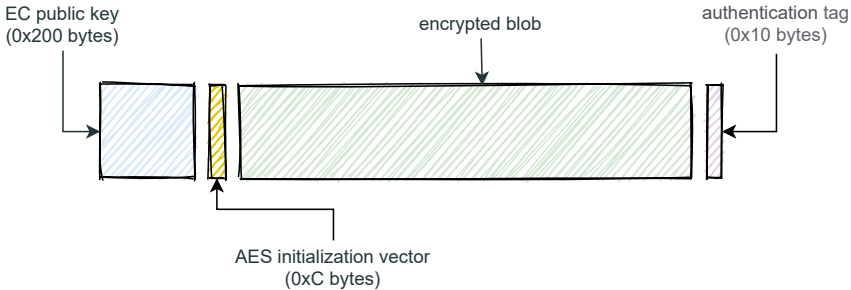
T + 1 semaine

```
---------------[ Shared modules ]---------------

> mod 0x00 -           libINTEGRITY.so size
> mod 0x01 -                   libc.so size
> mod 0x02 -             libopenssl.so size

-----------------[ Tasks List ]-----------------

> task 01 - path keymgr.elf - size 0x00013588
```

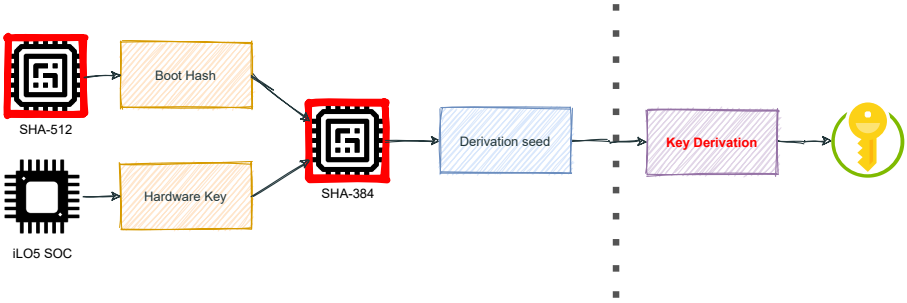
- Image mono-tâche
- 3 libs dont OpenSSL

Une chaîne de boot “augmentée”



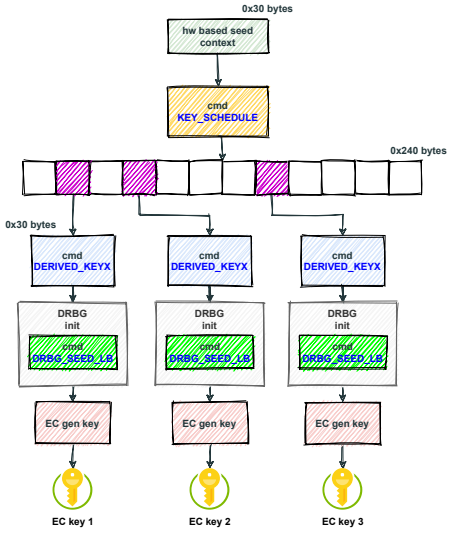


- **Une clé privée reconstruite lors du démarrage de `keymgr`**
- Primitives OpenSSL : `ECDH_compute_key`, Elliptic Curve Diffie-Hellman
- Utilisée avec la clé publique de l'entête pour dériver un secret partagé
- Chiffrement symétrique authentifié des données, AES-GCM



keymgr : un beau challenge à reverser

- **Facteur X : un cryptoprocesseur**
  - Opérations utilisées : SHA384, AES-CTR, AES-GCM
- Deux grandes étapes :
  1. Dérivation d'un seed à partir de valeurs hardware
  2. Une fonction de key scheduling



"Commandes", wrapper sur le crypto-processeur :

- Étape de dérivation (SHA384) paramétrée par une clé :
  - "KEY_SCHEDULE"
  - "DERIVED_KEYX"
  - "DRBG_SEED_LB"

Comment générer une clé déterministe ?

- OpenSSL's EC_KEY_generate_key
- Remplacement du PRNG
- Graine déterministe (issue de la commande DERIVED_KEYX)

Objectif
Générer cette clé en offline ? (sans le hardware iLO)

Fail Hard. Fail Fast.

- Réimplémentation complète en C à partir de l'analyse statique
- Extraction des valeurs hardware via la 1day fmt-string
- $\Rightarrow$ **échec du déchiffrement**

La surenchère

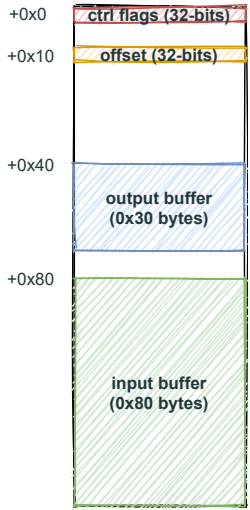
- Interfaçage avec le cryptoprocesseur
- Validation pas-à-pas des étapes

Talking to the unknown

Objectif(s)

- Valider notre compréhension des échanges avec le cryptoprocesseur
- Réimplémenter le second étage de déchiffrement
- Mettre à jour nos outils d'analyse de firmware

T + 2 semaines



Control flags

- `SHA384_DIGEST_DATA_START`
- `SHA384_DIGEST_MORE_DATA`
- `SHA384_DIGEST_DATA_END`
- `SHA384_DIGEST_CLEAR_OUTPUT`

Offset

- Auto-incrémenté lors d'écritures dans le buffer d'input
- Taille des données d'entrée en bits

Comment ?

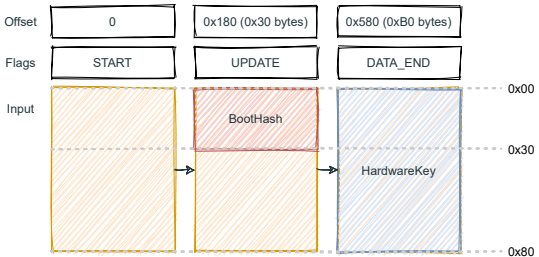
- Lire et écrire la mémoire est suffisant

1-day à la rescousse !

- Ré-utilisation de la vulnérabilité SSH
- Mapping du cryptoprocesseur dans l'espace mémoire
- Intéraction via les primitives de lecture / écriture

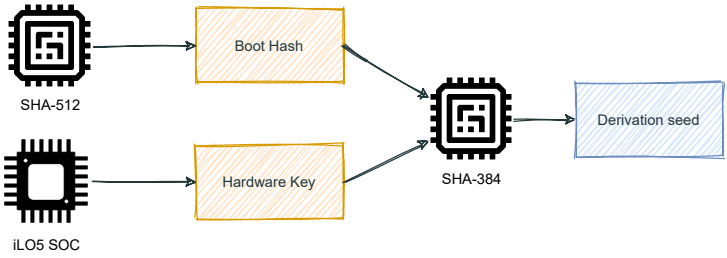
Bug #1 : SHA384_DIGEST_MORE_DATA (UPDATE)

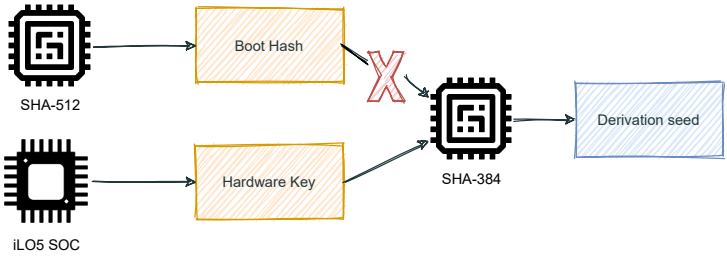
- Sémantique attendue : mise à jour de l'état interne du digest avec le contenu du buffer d'entrée
- Sémantique observée : si le buffer d'entrée n'a pas été complètement rempli, les données sont "ignorées"



Attendu : `SHA384( BootHash || HardwareKey )`

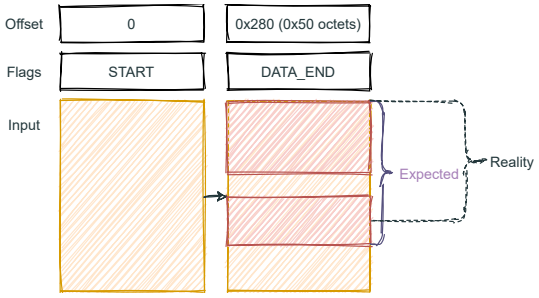
En réalité : `SHA384( HardwareKey || HardwareKey[:len(BootHash)] )`


SHA-512
Boot Hash
iLO5 SOC
Hardware Key
SHA-384
Derivation seed


Boot Hash
SHA-512
Hardware Key
iLO5 SOC
SHA-384
Derivation seed

Bug #2 : écriture non contiguë du buffer d'input

- Registre `Offset` incrémenté du nombre de bits écrits, **sans considération de la position de l'écriture dans le buffer**
- Le champ `Offset` est directement utilisé pour lire les données de façon contiguë



Attendu : `SHA384( INPUT[:0x60] )`

En réalité : `SHA384( INPUT[:0x50] )`

Est-ce une victoire ?

- Notre implémentation est mise à jour pour prendre en compte ces 2 bugs
- **Nouvel échec, la clé calculée est toujours invalide**

Debugging 101

Objectif(s)

- Rechercher des informations de debug pour comprendre nos échecs
- Récupérer ces informations
- Corriger notre implémentation

T + 3 semaines

Messages de debug

- État intermédiaire

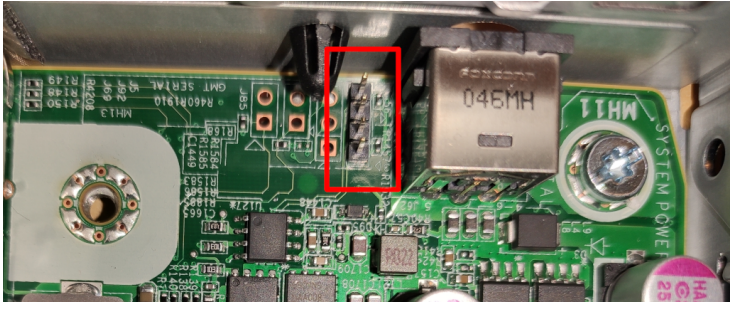
```
coproc_crypto_cmd(coproc_status, "KEY_SCHEDULE", 0, 0, 0, &DRBG_BUFFER_POOL, 0x240);
EVP_EncodeBlock(tmp_buffer, &DRBG_BUFFER_POOL, 0x21);// base64 encode
printf("Key Schedule Validation: %s\n", tmp_buffer);
```

- Clé publique finale

```
ec_key = EC_KEY_POOL_.ec_key1;
bio = BIO_new_fp(FD_STDOUT, 0);              // dump key to stdout
PEM_write_bio_EC_PUBKEY(bio, EC_KEY_POOL_.ec_key1);
```

Objectif

- Les informations de debug sont affichées sur la sortie `UART`
- Retrouver l'`UART` sur le `Microserver`



Au démarrage

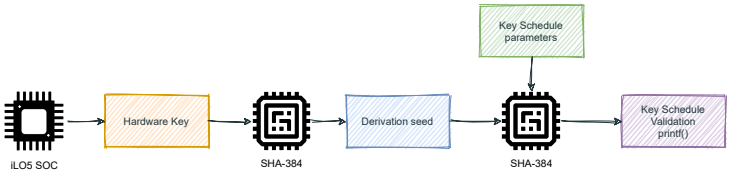
```
Key Schedule Validation: l03wN5OaDle9gyhfEJShR5jv0sKBOtfT25uk2U/vjxRA
-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE4StRIN6NFi6XOOOaNMLePDm0mYmXIpDf
03KrjkhWjZW8z3QNeyUXVNxHayZEKFL6Xk6vjkYYeJNdqg9yEzIOa2GK2emSgp4D
RNgUyUpix0jq5+lluKXWUyFQ6rBJ45Dr
-----END PUBLIC KEY-----
```

Dans notre implémentation

```
Key Schedule Validation: enRnbrYwZKTWBJF955yiy5VIbe4ROBE4g1EO5dOrYcVl
```

- Échec très tôt dans la chaîne de dérivation
- L'étau se resserre

Tout se passe ici



Théorie

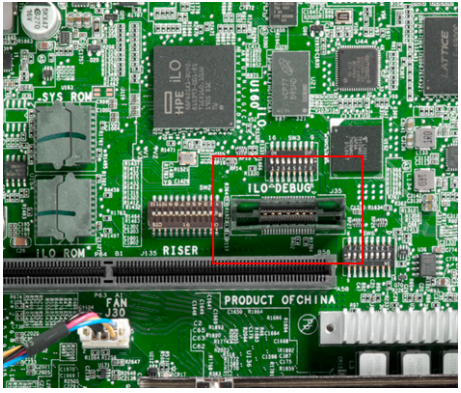
- Les constantes dans les registres hardware ont bougé entre les versions 1.x et 2.x
- Il nous faut de meilleures capacités de debug

Hardware debugging

Objectif(s)

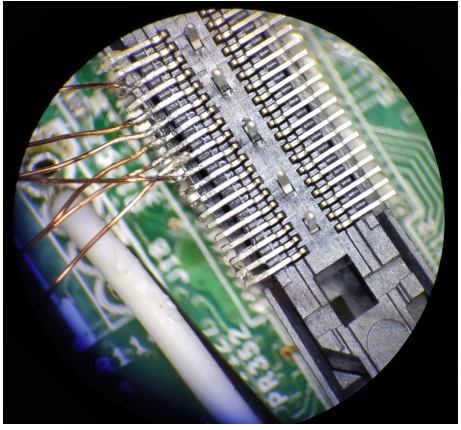
- Rechercher des ports de debug
- Utiliser cet accès pour relire les registres
- Corriger notre implémentation

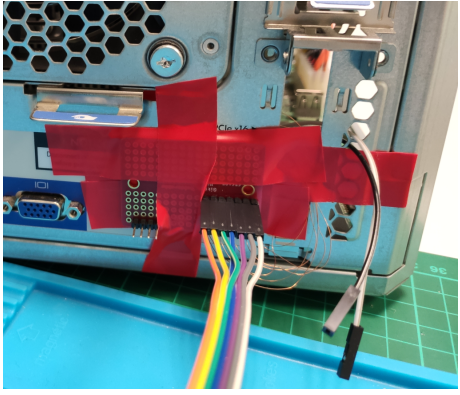
T + 3 semaines



Sur la carte mère du Microserver Gen10

- Repéré sur le site d'HPE
- Port `MICTOR`
- Permet théoriquement du `JTAG`





Finalement, un adaptateur correct a été acheté :)

Énumération JTAG

```
Device ID #1: 0101 1011101000000000 01000111011 1 (0x5BA00477)
Device ID #2: 0000 0111100100100110 11110000111 1 (0x07926F0F)
```

Résultats médiocres

- Problèmes avec `TDI`
- Pas de solution trouvée $\Rightarrow$ abandon de la solution hardware
- On est des “softeux”, cherchons une solution logicielle

0day hunting

Objectif(s)

- Rechercher une vulnérabilité dans les firmwares 2.x
- Relire les registres du SOC
- Corriger notre implémentation

T + 4 semaines

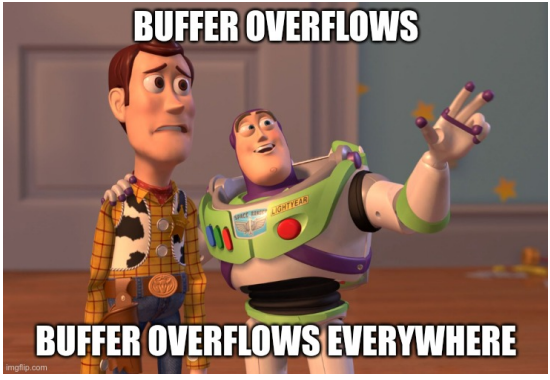
Cible : firmwares 2.x

- Interface CHIF
- Plusieurs tâches joignables depuis l'hôte
- Image `recovery` en clair : tâche `blackbox` présente

Tâche `blackbox`

- Nombreux gestionnaires de commandes
- Commande 5 : menu de debug
  - Commandes en mode texte
  - Sortie affichée sur l'UART
  - Exemple :

```
bb fdump (file)        — Hex dump 'file'. Absolute path necessary
```



Utilisation massive de fonctions dangereuses

- `sprintf`
- `strcpy`

Aucune mitigation

- NX
- ASLR
- Biscuits de pile

```
root@debian:/home/synacktiv# python -i bb_exploit.py
[*] Run interactively with "python -i"
>>> bb_exploit_dump_users()
```

Dump i:/vol0/cfg/cfg_users_key.bin

```
0x00000000  3c bf b8 ae 1d 51 ea a8 98 2a f7 42 cb 21 21 78  <....Q...*.B.!!x
0x00000010  a6 fb 8f 98 49 a6 73 41 a1 56 db 1d 92 a4 f1 f8  ....I.sA.V......
```

IV

```
0x00000000  a7 e2 95 f6 28 a7 95 48 4c 0d 4e 76 07 04 78 0b  ....(..HL.Nv..x.
```

```
[01][03ff][Administrator] Administrator / QVW77R6R
[04][000b][UserName2] user2 / S3curePass
[03][0003][Username1] user1 / p@ssw0rd
[02][03ff][admin] admin / [redacted]
>>>
```

Lecture des valeurs “hardware”

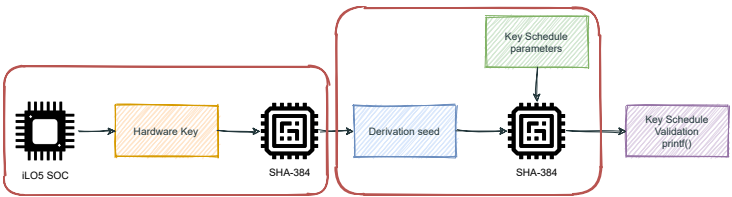
$\Rightarrow$ Valeurs identiques aux anciens firmwares

Lecture des valeurs “hardware”

⇒ Valeurs identiques aux anciens firmwares

Rip du code de `keymgr`

- Exécution de deux blocs de code
- Comparaison du buffer de sortie du cryptoprocesseur à notre implémentation



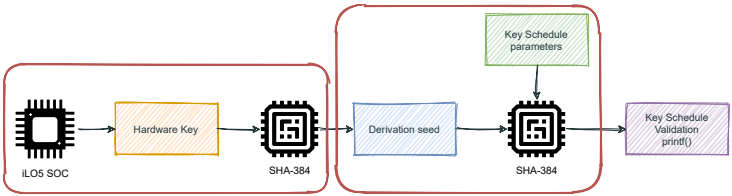
Résultats identiques

Lecture des valeurs “hardware”

⇒ Valeurs identiques aux anciens firmwares

Rip du code de `keymgr`

- Exécution de deux blocs de code
- Comparaison du buffer de sortie du cryptoprocesseur à notre implémentation



Résultats identiques

Nouvelle théorie

- Notre contexte d'exécution est différent de `keymgr`
- Nous devons déboguer/instrumenter `keymgr`

Secure boot bypass

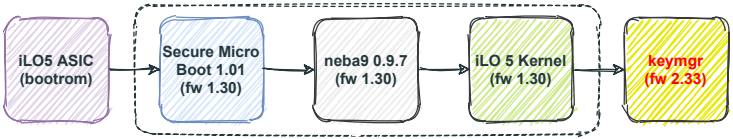
Objectif(s)

- Créer un firmware modifié
- Ajouter des hooks de debug
- Comprendre nos erreurs
- Corriger nos implémentations

T + 6 semaines

Bypass du Secure Boot

- CVE-2018-7113 : permet de charger une image userland non signée
- Le plan : charger un `keymgr` modifié via un ancien noyau vulnérable

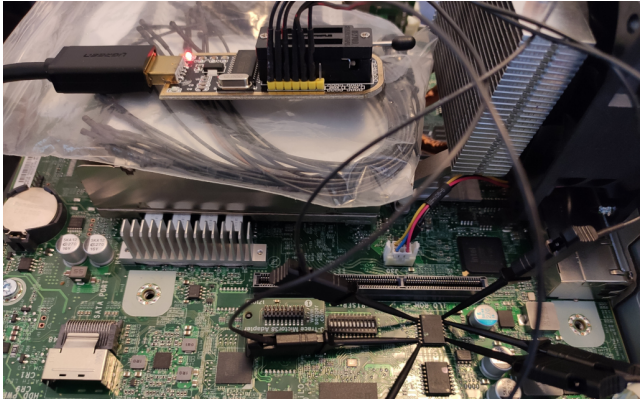


`keymgr` démarre !

- Affiche le même "*Key Schedule Validation*" que pour un boot normal
- Modification mineure : démarre toujours

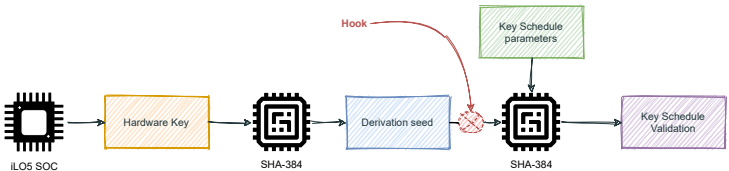
Ecriture d'un firmware arbitraire

- Soit via une vulnérabilité : nécessite de reflasher un firmware valide à chaque fois
- Soit matériellement : plus simple, plus rapide



Installation d'un hook

- `printf()` de debug
- En amont de l'opération de key schedule
- Dump de l'entrée en base64 sur l'UART



- Sur l'UART :

```
Key Schedule coproc status:
v0lzOt0Taev0yrxBtI3b33av1zWCZkwtEpnCI+WFCa7cZyqm0KmQT8+sxyduqPya
```

Valeur récupérée

```
BC E9 73 3A DD 13 69 EB
F4 CA BC 41 B4 8D DB DF
76 AF D7 35 82 66 4C 2D
12 99 C2 23 E5 85 09 AE
DC 67 2A A6 D0 A9 90 4F
CF AC C7 27 6E A8 FC 9A
```

Notre implémentation

```
8F D6 EA 44 DD 13 69 EB
F4 CA BC 41 B4 8D DB DF
76 AF D7 35 82 66 4C 2D
12 99 C2 23 E5 85 09 AE
DC 67 2A A6 D0 A9 90 4F
CF AC C7 27 6E A8 FC 9A
```

La valeur est un hash SHA-384. Quelle est cette magie ?

Race condition

- Absence d'utilisation des mécanismes de synchronisation du cryptoprocesseur
- La boucle de copie commence avant la fin des calculs du cryptoprocesseur, sur un état intermédiaire du SHA-384

```
SHA384_DIGEST_FLAG_MODE = SHA384_DIGEST_DATA_END;// sha384.digest()
for ( m = 0; m < 0xC; ++m )
  *&derivation_seed[4 * m] = SHA384_DIGEST_OUTPUT[m];
*&SHA384_DIGEST_FLAGS = SHA384_DIGEST_CLEAR_OUTPUT;
v6 = coproc_crypto_key_schedule(derivation_seed);
```

Vérification

- Implémentation de SHA-384 en Python
- Récupération de l'état intermédiaire juste avant le `.digest()`

```
>>> sha384(HardwareKey + ctx_HardwareKey[:0x30]).hexdigest()
FINAL STATE : bce9733a2cb047ecb74cd8d408507dbb7937cd5a4599ca655920362216d8a65a9
aa562b74b50e3bfdf5a26e88d15cece31574ffe3f5fa8217c9516988038505d
'8fd6ea44dd1369ebf4cabc41b48ddbdf76afd73582664c2d1299c223e58509aedc672aa6d0a990
4fcfacc7276ea8fc9a'
```

Notre outil de déchiffrement est fonctionnel

- Déchiffrement de la première enveloppe
- Déchiffrement de l'image userland
  - 4 nouvelles tâches
  - Assez peu de différences
  - "Tout ça pour ça :)"

L'outil est disponible !

https://github.com/airbus-seclab/ilo4_toolbox

Conclusion

Objectif(s)

- Savourer la victoire
- Restaurer l'équilibre dans la force

T + 2 mois

- Capacité d'analyse de nouveaux firmwares recouvrée
- Chiffrement du firmware :
  - Valeur ajoutée discutable pour l'utilisateur final
  - Implémentation particulièrement complexe
  - Pas d'utilisation d'un secure element
  - Utilisation buggée du cryptoprocesseur ?
- iLO5, les mêmes faiblesses inhérentes au système :
  - Secure boot cassé
  - Absence totale de mitigation moderne

- HPE iLO5, systèmes critiques du SI :
  - Patchez
  - Isolez
  - Surveillez
- Correctif disponible :
  - **2.41**[6] (26 Mars 2021)
  - "*Critical - HPE requires users update to this version immediately.*"
- Bulletin de sécurité `HPESBHF04133`[7]

6. https://support.hpe.com/hpesc/public/swd/detail?swItemId=MTX_9e149bcaae774cc190a26ea98e
7. https://support.hpe.com/hpesc/public/docDisplay?docId=hpesbhf04133en_us

- Mark Menkhus et le PSRT Hewlett Packard Enterprise
- Nos équipes **Synacktiv** et **Airbus** pour leurs précieux retours et conseils
- Nos relecteurs du comité de programme SSTIC
- Raphaël, Xavier pour l'opération à cœur ouvert sur le serveur ML110 :)
- Crédits DOOM guy face : `Reinchard2 @ ZDoom board` [8]

8. https://forum.zdoom.org/viewtopic.php?f=46&t=48921

Merci pour votre attention

Questions ?

Pour nous contacter :
fabien [dot] perigaud [at] synacktiv [dot] com - @0xf4b
alexandre [dot] gazet [at] airbus [dot] com
snorky [at] insomnihack [dot] net - @_Sn0rkY