NDH 2k15
Blackbadge Write-up

Jeremy Buet Xavier Bonnetain Nicolas Iooss
de l'équipe h4ck3s*

20 – 26 juin 2015

1 Découverte de la bête

Le badge se présente sous la forme d'un circuit électronique, avec une prise microUSB qui permet de l'alimenter, de le démarrer et de communiquer avec un ordinateur. Il contient une puce Atmega 32u4, un accéléromètre, une led et un buzzer. Au démarrage, 3 notes se font entendre, laissant entendre que le firmware a fini de booter. Il possède également un bouton qui permet de rebooter le badge dans un mode spécial pendant quelques secondes avant de booter sur le firmware.

L'interface microUSB fournit une interface série sur laquelle on peut communiquer. Elle nous propose 3 options par le message
« `Shall we play a game?` » :

- `Quizz`
- `Fortune`
- `Wargame`

On remarque immédiatement la référence à Wargames, aussi taper « Joshua », comme le mot de passe de la backdoor du WOPR dans le film pourrait s'avérer intéressant. On remarque que dès qu'une lettre est tapée, elle est interprétée comme une commande, ne nous permettant pas de taper directement Joshua. En revanche, taper « h » nous débloque une option supplémentaire : « Badge status ». La première exécution donne `DUMMY`, les suivantes `ROOKIE`.

*Afin de ne pas faire de jaloux, l'ordre des auteurs a été déterminé selon le MD5 des prénoms écrits en majuscule.

1.1 Quizz

Le Quizz nous propose une série de questions, en précisant que les réponses doivent commencer par une majuscule. La première question est
`What is the Answer to the Ultimate Question of Life, The Universe, and Everything?`.
La réponse est connue de la plupart des geeks, 42. Question suivante.

`What is the main town of Assyria?`
Une référence plus subtile, à *Monty Python's Holy Grail*. La réponse ne se trouve pas dans le film, mais il suffit d'une petite recherche sur Wikipédia pour trouver Ashur.

`Uncipher this: 413532 541215 244422 1154`
En voilà une question plus difficile. On ne connaît même pas l'algorithme. On remarque tout de même qu'on a 22 caractères, tous compris entre 1 et 5. on a donc 25 combinaisons possibles de deux nombres, soit presque le nombre de lettres de l'alphabet. Après quelques essais, on tombe sur un message qui semble avoir du sens : `POLXBEISGAX`. On sent que le `X` doit être remplacé par `Y`, et `POLYBEISGAY` marche. On se rend alors compte que le chiffre utilisé était justement un carré de Polybe.

`Can you name Gibson's *cold* intelligence?`
La référence à laquelle porte cette question est très floue, Gibson étant un nom assez commun. La question est étrangement formulée, aussi nous parait-il pertinent d'essayer des réponses comme « Yes », « Sure », « No », « Maybe »... Aucune ne marche.
Aussi, Google nous parle beaucoup de renseignement américain, et on note une page présentant des extraits d'une œuvre de SF connue, bien que personne d'entre nous ne l'ait lue, *Neuromancer*, écrit par William Gibson. Ce livre est un des fondateurs du mouvement cyberpunk et comporte plusieurs AI, l'une d'entre elles s'appelant Wintermute. Winter parait bien faire référence au froid, nous l'essayons, il marche `\o/`.

`Who reads Playboy in the movie 'Sneakers'?`
Une question portant sur un film. Le premier réflexe à avoir est d'aller sur IMDb. Ni le synopsis ni les spoilers ne nous aident. Cependant, on a la liste des personnages dans le film. On peut les essayer, par ordre d'importance. « Whistler » s'avère être la bonne réponse.

Là un gentil message nous dit que nous avons gagné le jeu ![1]

Attaquons-nous à présent au jeu suivant.

1. Ce qui me fait dire que j'ai perdu.

1.2 Fortune

Fortune est un petit jeu de pari : nous commençons avec 100 pièces, il faut arriver à 65535 pièces. Pour cela, on parie une somme sur un nombre entre 1 et 10, et un nombre est tiré au hasard. Si nous avons parié sur le bon, nous gagnons la somme, sinon nous la perdons. On voit assez vite que si on respecte les règles, on n'arrivera jamais à 65535 pièces. Tentons d'y aller subtilement. Que se passe-t-il si, par exemple, on laisse le doigt appuyé en continu sur la touche « 2 » ? On s'aperçoit que la somme pariée devient -7282 pièces, et si on perd, on perd -7282 pièces, ce qui revient à gagner 7282 pièces. Voilà qui parait plus efficace pour atteindre la somme voulue. Au terme de plusieurs essais, on se retrouve avec 103 pièces de trop, qu'il suffit de parier et espérer perdre.

Une fois l'objectif atteint, le jeu est validé.

1.3 Wargame

Cette épreuve était plus mystérieuse : si on tapait 8 chiffres, le badge nous renvoyait deux valeurs, l'une constante et l'autre changeant suivant le nombre rentré. On comprend donc que la valeur constante est l'objectif, et la valeur changeante le résultat, bien que l'interface nous dise le contraire.

Une observation de l'ordre de grandeur du résultat nous a fait penser à un `uint32` [2]. On suppose donc avoir une fonction $\phi : [\![0\,;10^{8}-1]\!] \to [\![0\,;2^{32}-1]\!]$. De là, nous nous sommes dit : « Bon, c'est un polynôme de degré au plus 10^8, en avant, calculons les coefficients ».

Nous avons d'abord commencé par des petites valeurs : `00000000 00000001 00000002` ..., et avons vite trouvé que c'était une simple fonction affine, facile à inverser. Malheureusement, l'inverse du résultat voulu ne faisait pas 8 chiffres de long. Pire, la formule n'était valide que pour les 10 premières valeurs !

En continuant à examiner différents résultats nous nous sommes rendus compte que l'on trouvait d'autres fonctions affines en considérant des valeurs du type `00000000 00000010 00000020` ....

De là, nous nous sommes dit que ϕ avait sans doute une expression simple avec la structure $\phi : [\![0\,;9]\!]^{8} \to [\![0\,;2^{32}-1]\!]$. Nous avons pu assez facilement retrouver les 8 termes de degré 1. Ensuite, restait à vérifier qu'il n'y avait pas de termes croisés qui entreraient fourbement en ligne de compte. Quelques

2. Nombre entier positif ou nul sur 32 bits
8. Ceci est un exposant, et non une note de bas de page.
32. Toujours un exposant.

tests avec des nombres sans 0 nous ont fait dire que non.

On a donc pu trouver

$$\begin{array}{rccc} \phi & : & [\![0\,;9]\!]^8 & \to & [\![0\,;2^{32}-1]\!] \\ & & (a_1,...,a_8) & \mapsto & t_0 + \sum_{i\in[\![1;8]\!]} a_i t_i \mod 2^{32} \end{array}$$

$$\begin{array}{rcl} (t_i) & = & \{\ 2083654722,\ 3724747905,\ 171070351,\ 3122836833, \\ 3235238063,\ 2307721793,\ 2845493711,\ 1680712481,\ 3545087727\ \} \end{array}$$

Malheureusement, une fonction comme ça, ça s'inverse beaucoup moins bien qu'une simple fonction affine. Nous avons donc décidé d'effectuer un bruteforce dessus, qui nous a donné le code : `06191783`, qui ressemble à un début de numéro de téléphone.

2 Bon, on a fini ce qu'il a proposé ... now what ?

2.1 À la recherche de l'indice perdu

Une fois les 3 jeux validés, on est toujours en mode `ROOKIE`. On a certainement raté quelque chose qui permette de passer au mode suivant. Pour avoir des idées, on peut s'intéresser au contenu du projet Github permettant de programmer le badge[3]. Celui-ci nous rappelle qu'il y a un accéléromètre sur le badge, qui n'a pas encore été utilisé dans le challenge. En effet, à part l'accéléromètre, le buzzer émet 3 notes au démarrage, la led s'illumine lorsqu'on appuie avec le doigt sur la dent de droite, et il n'y a a priori pas de manière simple d'accéder à la mémoire EEPROM externe.

À quoi peut donc bien servir l'accéléromètre ? Agiter le badge dans tous les sens ne marche pas. Le donner à un chat qui l'agite dans tous les sens ? Pas de succès non plus. Tentons un Konami code, on sait jamais. Il ne marche pas non plus.

2.2 Invocation du code qui fait mal aux yeux

Reste la méthode auxiliaire, à savoir analyser statiquement un dump du firmware. La commande magique pour récupérer le firmware (parce qu'on peut mettre du temps à trouver les paramètres idoines) est, pour ce badge : `avrdude -c avr109 -p atmega32u4 -U flash:r:flash.hex:i`

3. https://github.com/virtualabs/Ndh15Badge-library

Ensuite il est possible d'obtenir le code assembleur AVR 8-bits exécuté avec :

```
avr-objdump -mavr -Iihex -D dump.ihex
```

ou, pour se faire moins mal aux yeux, on peut convertir le dump IHEX en fichier binaire classique et utiliser `strings` :

```
avr-objcopy -I ihex -O binary dump.ihex dump.bin
strings -tx dump.bin
```

On retrouve alors tous les messages déjà vus dans le menu et les jeux, ainsi que quelques chaînes de caractère intéressantes, comme "`m - Dump memory`" et "`Memory> Dumping ...`". On trouve aussi la liste des modes de jeu : `DUMMY`, `ROOKIE`, `NINJA` et `GOD`.

Une fois ceci fait, il n'y pas grand chose qu'il est possible de faire à part tenter de comprendre le code du badge. L'assembleur AVR-8 bits fait assez mal aux yeux, mais après quelques heures de persévérance il est possible de comprendre plusieurs choses dans le code du firmware :

- Le code situé entre `0x0316` et `0x032c` est exécuté lors du démarrage du programme et copie le bloc de données de `0x3bd8..0x4131` (dans la mémoire programme), qui contient entre autres les chaînes de caractères repérées par `strings`, aux adresses de `0x0100` à `0x659` dans la RAM. Ce sont ces adresses qui sont ensuite utilisées pour par exemple afficher des messages.
- La fonction qui s'étend de `0x0636` à `0x06e1` initialise un état de déchiffrement RC4 dans un tableau de 256 octets pointé par la paire de registre `r25:r24` avec la clé de `r21:r20` octets en `r23:r22`.
- La fonction suivante, de `0x06e2` à `0x075e` permet de chiffrer/déchiffrer un bloc de données de longueur `r21:r20` octets en `r23:r22` avec RC4 en utilisant un tableau d'état en `r25:r24`, précédemment initialisé.
- La fonction de `0x0e76` en `0x1091` permet de lire `r19:r18` octets dans un bloc mémoire pointé par `r21:r20`, à la page d'adresse `r22` dans la mémoire externe chiffrée du badge. Le contenu de cette mémoire est chiffré avec RC4 par blocs de 8 octets, chaque bloc utilisant une clé de chiffrement de 9 octets composée de la concaténation de 8 octets constants et d'un octet correspondant à l'adresse de ce bloc de 8 octets. Le début de la mémoire chiffrée contient en fait un entête de 16 octets : 4 octets "NDHC", 4 octets de somme de contrôle (vérifiée au démarrage) et 8 octets utilisés comme partie constante dans la clé de chiffrement.
- La fonction de `0x1470` en `0x15f4` utilise l'accéléromètre pour progresser dans la séquence 1, 0, 1, 0, 2, 0, 2, 0, 3, 0, 4, 0, 3, 0, 4, 0, 7, en repartant du début à chaque fois que la lecture de l'accéléromètre cor-

respond à un autre état que le suivant. Le décodage de cette fonction montre que la lecture des axes X et Y de l'accéléromètre est convertie en 0 pour un mouvement "stable", 1 pour un vers le haut, 2 pour bas, 3 gauche et 4 droite, et que le 7 est un marqueur de fin de séquence... La séquence correspond donc à Haut-Haut-Bas-Bas-Gauche-Droite-Gauche-Droite, qui est une partie du Konami code.

2.3 Invocation réussie !

D'une part, l'analyse statique permet d'une part de déchiffrer le contenu de la mémoire externe. Ce n'est pas forcément ce que l'on cherchait initialement, mais ça peut être utile pour la suite. Une fois le contenu de la mémoire récupéré et déchiffré (en conservant l'entête de 16 octets qui n'est pas chiffré), on obtient ceci :

```
000: 4e44 4843 ab84 795f f6d2 a594 7322 a7fe  NDHC..y_....s"..
010: 0105 ffff 3036 3139 3137 3833 0000 0000  ....06191783....
020: 0334 3200 0641 7368 7572 000c 504f 4c59  .42..Ashur..POLY
030: 4245 4953 4741 5900 0b57 696e 7465 726d  BEISGAY..Winterm
040: 7574 6500 0957 6869 7374 6c65 7200 e3f1  ute..Whistler...
050: 2503 5bd8 f4fc 1bd7 d05f 56d8 b4fb 4867  %.[......_V...Hg
060: 6fba 3458 764e 9646 ce8a 4e3c 442c 3018  o.4XvN.F..N<D,0.
070: fb03 a238 21a2 339d c1ad 157f 1d93 1326  ...8!.3........&
080: 1d24 fa03 bf2f 08f1 e4ef 062c bd62 62c2  .$.../.....,.bb.
090: 2724 6e1d b9d8 449f 51fe 41d9 507b 35f8  '$n...D.Q.A.P{5.
0a0: 966b cbda 3638 3851 deac e964 ebc8 0000  .k..688Q...d....
0b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

D'autre part, on se rend compte qu'il faut effectivement réaliser le Konami code pour déverrouiller une étape, mais apparemment pas comme l'intuition demande de le faire. Et si on tournait le badge avant de faire le Konami code ? Bingo ! ! Si l'on place le badge tel que les dents sont à droite, et les puces vers le haut et qu'on fait le code (Haut-Haut-Bas-Bas-Gauche-Droite-Gauche-Droite), la LED s'allume brièvement en rouge, quelque chose à été pris en compte ! Mieux, si on rappuie sur la dent de droite, au lieu de s'allumer Vert-Rouge-Vert, elle s'allume Rouge-Vert-Bleu. Entrons sur la communication série, on est passé de `ROOKIE` à `NINJA`. Il ne reste plus qu'à tirer les missiles pour passer en `GOD` et avoir le message final `\o/`

3 Le message de la victoire

Une fois en mode `GOD`, le message final s'affiche :

```
======\\o/ VICTORY \\o/======
Send a write-up to: virtualabs+3108b6c37363e4cc@gmail.com
```

En fait, comme nous n'avions pas réussi à entrer le Konami code correctement pour passer en mode `NINJA`, nous avons obtenu ce message différemment, grâce à de l'analyse statique. En effet, les fonctions de chiffrement RC4 ne sont pas que utilisées que pour accéder à la mémoire externe, mais sont également utilisées par la fonction qui s'étend de `0x05a4` en `0x0634`. Cette fonction déchiffre un bloc 91 octets situé à l'adresse `0x53` de la mémoire externe avec une clé RC4 de 12 octets à l'adresse `0x10`. Or cette zone de 12 octets est utilisée de la manière suivante :

- L'octet en `0x10` contient l'état du jeu (0 = `DUMMY`, 1 = `ROOKIE`, 2 = `NINJA` et 3 = `GOD`).
- L'octet en `0x11` contient la dernière question résolue du quizz, 5 quand ce jeu est terminé.
- Les 2 octets `0x12` et `0x13` contiennent le montant courant du jeu Fortune, soit `0xffff` à la fin.
- Les octets de `0x14` à `0x1c` contiennent une copie du code utilisé pour le wargame.

Ainsi, avec la clé `'\x03\x05\xff\xff\x30\x36\x31\x39\x31\x37\x38\x33'`, on peut déchiffrer la zone chiffrée dans le dump de la mémoire externe et obtenir le message final.

Remerciements

Merci à Julien Cretin et aux autres membres de la h4ck3s pour leur aide dans la résolution du challenge.

Merci au staff pour avoir bien voulu nous changer le badge après que nous l'ayons briqué une première fois [4].

Merci aux concepteurs du challenge et plus largement à tous ceux qui ont permis qu'il existe.

4. Oups