Le Reset Glitch Hack : un nouvel exploit sur Xbox 360

Soumis par Yoshee
28-08-2011

Le développeur et hacker français, GliGli, est fier de vous livrer aujourd'hui via les forums Xbox-Hacker.org, ce que la majorité des fans de homebrews attendait depuis plus d'un an : un nouvel exploit sur Xbox 360. Cet exploit baptisé "Reset Glitch Hack" nécessite l'installation d'une puce et est compatible avec tous les modèles Slim et certaines FAT : Zephyr et Jasper (le support des cartes mère Falcon viendra ultérieurement lorsque l'auteur en aura une sous la main).

Pour faire (très) simple, la puce envoie une série d'impulsions électriques au processeur afin de déstabiliser la console et lui faire croire qu'un CB modifié est authentique (correctement hashé et signé). Cette opération ne réussi pas à chaque fois, mais elle se répète jusqu’à son succès. Une fois que le CB modifié/hacké est validé par la console, il possède les droits suffisants pour lancer du code non signé, dans notre cas XeLL, le Xenon Linux Loader. Si vous voulez plus de détails, vous pouvez lire l'explication de l'auteur ci-dessous et voir les sources du hack disponibles ICI.

Les avantages de ce Hack :

- Toutes les consoles seront compatibles sauf les Xenons

- Il n'est pas patchable, en effet le CB intervient tellement tôt dans le processus de boot de la console qu'il ne peut être révoqué.

Ses Inconvénients :

- Nécessite l'installation d'une puce.

- Le temps de boot est variable et peut aller jusqu'a 2-3 minutes.

Lien de la vidéo (si elle ne s'affiche pas)

La suite : l'explication complète du Hack par GliGli et traduite en français.

****

* The Xbox 360 reset glitch hack *

****

Introduction / quelque faits importants

==========

Tmbinc l'a dit lui-même, les approches logicielles pour lancer du code non signé sur la 360 ne marchent pas dans la plupart des cas, elle est conçue pour être sûre d'un point de vue logiciel.

Le processeur commence par lancer du code à partir de la ROM (1bl), qui ensuite charge du code signé en RSA et crypté en RC4 à partir de la NAND (CB).

Le CB initialise ensuite le moteur de sécurité du processeur, sa tâche sera d'encrypter et vérifier le hash de la mémoire physique DRAM en temps réel. De ce que nous avons trouvé, il utilise l'AES128 pour le cryptage et un hachage (Toeplitz?) robuste. Le cryptage est différent à chaque démarrage, car il dépend d'au-moins :

- Un hash du fuseset entier

- La valeur du compteur de temps du CPU (timebase)

- Une valeur totalement aléatoire générée par une partie hardware dédiée du processeur. Sur les FAT, ce RNG peut être électriquement désactivé, mais il y a une vérification sur "l'apparence aléatoire" (un comptage des bits à 1 en fait) dans le CB, il attend jusqu'à avoir ce qui lui semble être une valeur aléatoire correcte.

Le CB peut ensuite lancer un moteur logiciel basé sur une sorte de "bytecode" simple qui aura pour tâche d'initialiser la DRAM, le CB peut alors charger le bootloader suivant (CD) depuis la NAND, et le lancer.

Pour faire simple, le CD chargera alors un kernel à partir de la NAND, le patchera et le lancera.

Ce kernel contient une petite partie de code privilégiée (hyperviseur), quand la console tourne, c'est le seul code qui aura assez de droits pour lancer du code non signé.

Dans les kernels 4532/4548, une faille critique est apparue, et tous les hack 360 connus nécessitent d'utiliser un de ces kernels et d'exploiter cette faille pour lancer du code non signé.

Sur les 360s actuelles, le CD contient un hash de ces 2 kernels et arrêtera le processus de boot si vous essayez de charger l'un d'eux.

L'hyperviseur est une relativement petite partie de code à vérifier pour trouver une faille et apparemment aucune nouvelle version n'en contient qui pourraient autoriser le lancement de code non signé.

D'un autre coté, Tmbinc a dit que la 360 n'avait pas été conçue pour résister à certaines attaques hardware tel que le timing attack et le "glitching".

Le Glitching ici est essentiellement l'action de cibler certains bugs processeur de manière électronique.

C'est la manière que nous avons utilisée pour pouvoir lancer du code non signé.

Le glitch du reset en quelques mots

==========

Nous avons découvert qu'en envoyant de petites impulsions de reset au processeur pendant qu'il était ralenti ne le remettait pas à zéro, mais changeait plutôt la manière dont le code tournait. Il semble que ceci soit très efficace pour que les fonctions comparatrices de mémoires des bootloaders renvoient toujours l'information "pas de différence". Les fonctions comparatrices de mémoires sont utilisées entre-autres pour comparer le hash SHA du bootloader suivant avec celui stocké, et permettant s'ils sont identiques de le lancer. Nous pouvons ainsi mettre un bootloader qui ne passera pas la vérification de hash dans la NAND, glitcher le précédent et se bootloader se lancera permettant le lancement de presque tout code.

Détails pour le hack des fats

==========

Sur les FAT, le bootloader que nous faisons glitcher est le CB, afin de pouvoir lancer le CD que nous voulons.

Cjak a découvert qu'en activant le signal CPU_PLL_BYPASS, l'horloge du CPU ralentissait beaucoup, il y a un point de test sur la carte mère qui est une fraction de la vitesse du CPU, il est a 200Mhz lorsque le Dashboard est lancé, 66.6Mhz lorsque la console démarre et 520Khz lorsque le signal est activé.

Nous procédons donc de cette manière:

- Nous activons CPU_PLL_BYPASS autour du code POST 36 (hex).

- Nous attendons que le POST 39 démarre (le POST 39 est le comparateur de mémoire entre le hash stocké et le hash de l'image), et démarrons un compteur.

- Lorsque le compteur atteint une valeur précise (c'est souvent autour de 62% de la longueur totale du POST 39), nous

envoyons une pulsation de 100ns sur le CPU_RESET.

- Nous attendons un moment et ensuite nous désactivons CPU_PLL_BYPASS.

- La vitesse du CPU redevient normale, et avec un peu de chance, à la place d'obtenir une erreur POST AD, le processus de boot se poursuit et le CB lance notre CD modifié.

La NAND contient un CB "zero-paired", notre payload dans un CD modifié et une image SMC modifiée également.

Un glitch est par nature pas fiable, nous utilisons une image SMC modifiée qui reboot infiniment (en comparaison l'image d'origine redémarre 5 fois et ensuite la console est RROD) jusqu'à ce que la console aie démarrée correctement.

Dans la majorité des cas, la console boot en moins de 30 secondes après le démarrage.

Détails pour le hack des slims

==========

Le bootloader que nous faisons glitcher est le CB_A, de cette manière nous pouvons lancer le CB_B que nous voulons.

Sur les slims, nous n'avons pas été capables de trouver une piste sur la carte mère pour CPU_PLL_BYPASS.

Notre première idée a été de retirer le cristal 27Mhz maître et générer notre propre horloge à la place, mais il s'agissait d'une modification complexe, et qui n'a pas donné de résultats probants.

Nous avons ensuite regardé sur d'autres moyens de ralentir l'horloge du CPU et nous avons découvert que la puce HANA possède des registres PLL configurables pour l'horloge 100Mhz fournissant les paires différentielles CPU et GPU.

Apparemment ces registres sont écrits par le SMC au travers du bus I2C.

Le bus I2C est accessible librement, il est même accessible depuis un header (J2C3).

Ainsi la puce HANA devient notre arme de choix pour ralentir le CPU (désolé tmbinc, tu ne peux pas toujours avoir raison, elle n'est pas ennuyeuse et il s'agit bien d'un bus intéressant

Nous procédons donc de cette manière :

- Nous envoyons une commande i2c vers l'HANA afin de ralentir le CPU au code POST D8 .

- Nous attendons que le POST DA démarre (POST DA est le comparateur de mémoire entre le hash stocké et le hash de l'image), et nous démarrons un compteur.

- Lorsque le compteur atteint une valeur précise, nous envoyons une pulsation de 20ns sur le CPU_RESET.

- Nous attendons un moment et ensuite nous envoyons une commande i2c en direction de l'HANA afin que l'horloge du CPU revienne à son état normal.

- La vitesse du CPU redevient normal, et avec un peu de chance, à la place d'obtenir une erreur POST F2, le processus de boot se poursuit et le CB_A lance notre CB_B modifié.

Lorsque le CB_B démarre, la DRAM n'est pas initialisée, nous avons donc décidé de n'appliquer que les patchs suivants pour qu'il puisse lancer n'importe quel CD :

- Activation permanente du mode "zero-paired" afin de pouvoir utiliser une image SMC modifiée.

- Non-décryptage du CD, et attente d'un CD en clair dans la NAND.

- Pas d'arrêt du processus de boot si le hash CD n'est pas correct.

Le CB_B est crypté en RC4, la clé provient de la clé CPU, alors comment patcher le CB_B sans connaitre la clé CPU?

Le cryptage RC4 c'est en gros:

crypté = texte-clair xor flux-de-clé-pseudo-aléatoire

Ainsi si nous connaissons le texte clair et crypté, nous pouvons connaitre le flux de clés, et avec lui, nous pouvons encrypter notre propre code. Cela fonctionne ainsi :

flux-de-clé-pseudo-aléatoire-supposé = crypté xor texte-clair

nouvelle-encryption = flux-de-clé-pseudo-aléatoire-supposé xor patch-texte-clair

Vous pouvez penser qu'il s'agit d'un problème du style de l'oeuf et la poule, c'est à dire comment obtient-on le texte clair en premier lieu?

Facile: nous connaissons le texte en clair des CB provenant des consoles FAT, et nous avons supposé que les premiers bytes du code seraient les même que le nouveau CB_B, nous avons ainsi pu encrypter une petite partie de code pour dumper la clé CPU et décrypter le CB_B!

La NAND contient le CB_A, un CB_B patché, notre payload dans un CD en clair(non crypté) modifié, et une image SMC modifiée.

L'image SMC est modifiée pour avoir un nombre de reboot infini, et pour l'empêcher d'émettre périodiquement de commandes I2C pendant que nous envoyons les nôtres.

Vous n'avez peut-être pas encore réalisé, mais le CB_A ne contient pas de vérification sur les EFUSEs de révocation, ce hack est donc non patchable.

Bémols

======

Rien n'est parfait, il y a donc quelques bémols sur ce hack :

- Même si le Glitch trouvé est plutôt fiable (Taux de réussite de 25% par essai en moyenne), il se peut que la console prenne quelques minutes pour lancer du code non signé.

- Ce taux de succès est lié au hash du bootloader modifié que nous souhaitons lancer (CD pour les FAT et CB_B pour les slims).

- Il requiert un matériel rapide et précis permettant d'envoyer la pulsation de reset.

Notre intégration actuelle

==========

Nous utilisons un CPLD Xilinx CoolRunner II (xc2c64a), parce qu'il est rapide, précis, reprogrammable, pas cher et fonctionne avec deux niveaux de tension différents en même temps.

Nous utilisons l'horloge standby 48Mhz de la 360 pour le compteur du glitch. Pour le hack Slim, le compteur tourne même à 96Mhz (incrémenté sur les fronts montants et descendants de l'horloge)

Le code du CPLD est écrit en VHDL.

Nous avons besoin de connaitre le code POST actuel, notre première mise en application utilisait l'intégralité des 8 bits du port POST à cet usage, mais nous sommes maintenant capables de détecter les changements d'un seul bit du POST rendant le montage plus simple.

Conclusion

========

Nous avons essayé de ne pas inclure de MS copyrighté dans le pack de hack diffusé.

Le but de ce hack est de lancer XeLL et d'autres logiciels libres, je (GliGli) NE promeus EN AUCUN CAS le piratage ni quoi que soit qui s'en rapproche, je veux juste avoir la possibilité de faire ce que je veux avec le matériel que j'achète, y compris lancer mon propre code natif dessus.

Crédits

=====

GliGli, Tiros: Reverse engineering et développement du hack

cOz: Reverse engineering, beta test.

Razkar, tuxuser: beta test.

Cjak, Redline99, SeventhSon, tmbinc, tous ceux que j'ai oubliés... : Précédent reverse engineering et/ou du travail de hack sur la 360.

Ce hack comme l'a expliqué l'auteur n'a pas vocation à promouvoir le piratage, qui n'est d’ailleurs pas possible actuellement puisqu'il ne permet que le lancement de XeLL, lui même offrant la possibilité de lancer du code libre comme les homebrews libXenon, Linux etc ....

Source : Merci à Razkar de Logic-Sunrise pour la news !

Nota : comme vous l'aurez remarqué, cette news est une copie relue de notre source. En effet devant la pertinence de

l'originale, nous avons préféré paraphraser le contenu.