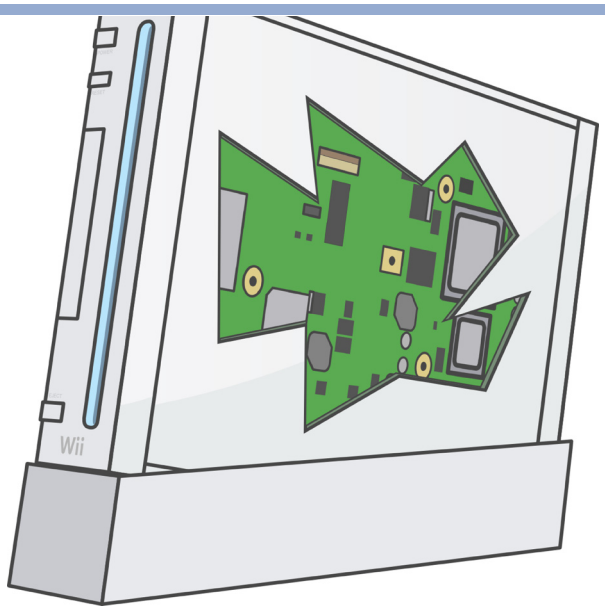
EXHIBIT M

DECLARATION OF RYAN BRICKER IN SUPPORT OF *EX PARTE* MOTION FOR TEMPORARY RESTRAINING ORDER AND ORDER TO SHOW CAUSE RE PRELIMINARY INUNCTION; ORDER OF IMPOUNDMENT

Wii: Hardware, seguridad, hacking y homebrew

Héctor Martín, <hector@marcansoft.com>



Euskal Encounter 16, BEC (Barakaldo), Julio de 2008

Introducción

Nuestras metas:

- Comprender cómo funciona el hardware y el software de la Wii
- Conseguir y mantener la posibilidad de ejecutar software casero en la Wii
- Utilizar la consola para propósitos propios y de todo tipo

Los fabricantes insisten en ponérnoslo difícil...

En esta conferencia veremos:

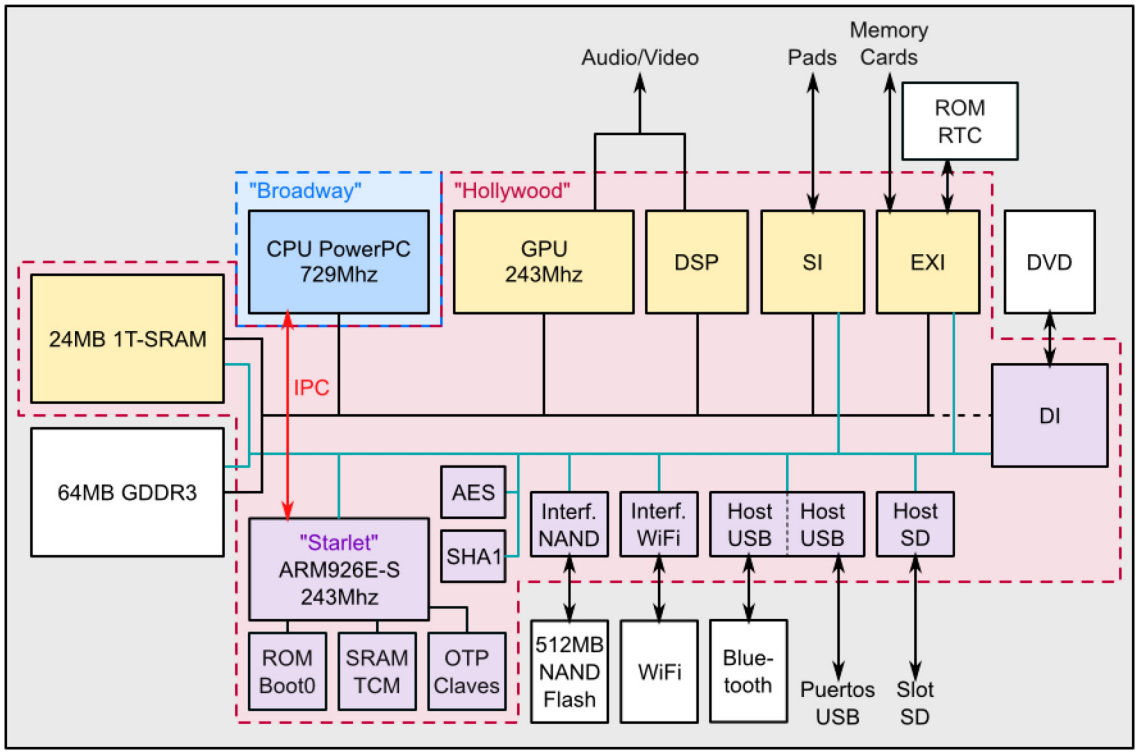
- El hardware de la Wii
- El software de la Wii
- El sistema de seguridad
- Los ataques y exploits
- El futuro de ellos
- Homebrew: herramientas de desarrollo, utilidades, y aplicaciones

¿Qué es la Wii?

Tras las decepcionantes ventas de la GameCube, Nintendo decidió reinventar la consola:

- Partimos desde el hardware de la GameCube (muy respetable)
  - *CPU PowerPC a 485Mhz ("Gekko")*
  - *24MB de memoria principal (1T-SRAM), 16MB de memoria de audio y DVD (DRAM)*
  - *GPU "Flipper" a 162Mhz (desarrollada por ArtX, ahora ATI)*
  - *Unidad lectora de discos ópticos (DVDs modificados)*
  - *Puertos de expansión, mandos, tarjetas de memoria, etc.*
- Actualizamos: la CPU ahora a 729Mhz, y la GPU a 243Mhz
- Y le añadimos lo siguiente:
  - *64MB de memoria GDDR3 (reemplazan a los 16MB de DRAM)*
  - *Lector de tarjetas SD*
  - *USB*
  - *Bluetooth (y mandos inalámbricos - podría hacer una conferencia sólo sobre ellos)*
  - *WiFi*

De GameCube a Wii



Se complican las cosas...

En la GameCube, ejecutar homebrew era relativamente fácil

- El software de los discos no lleva ningun tipo de firma
- La BIOS, aunque encriptada, no lleva firmas y la encriptación es muy mala
- Se pueden usar modchips para reemplazar la BIOS o cargar código desde DVDs

Nintendo parece haber aprendido la lección con la Wii

- Todo el software va firmado
- Todo el software va encriptado
- La criptografía que se usa es moderna, estándar, y segura
- Se establece una cadena de confianza desde el arranque, mediante hardare
- Los datos también van encriptados, incluso las partidas de los juegos
- La memoria FLASH de la consola va encriptada y firmada
- Hay un procesador separado encargado de la seguridad

O al menos esa es la teoría...

El software de la Wii

Hay muchos componentes de software independientes:

- Las etapas del proceso de arranque: **boot0**, **boot1** y **boot2**
- El software que normalmente se ejecuta en *Starlet*, **IOS**
- El sistema de compatibilidad de GameCube, **MIOS** y **BC**
- El menú del sistema
- Juegos cargados desde DVD
- Los Canales del menú (aplicaciones y juegos)
- "Canales" internos (p. ej. el CLUF)

Excepto las etapas de arranque, todos son *títulos* y comparten un sistema de instalación, firmado, y ejecución común

Títulos

Un título es un paquete de software, compuesto por:

- Un **TitleID** o ID de título, un código (o número, según se mire) que lo identifica
  - *Consta de un número de 64 bits, donde los primeros 32 identifican el tipo de título y los ultimos 32 suelen ser un código de cuatro letras ASCII*
  - *Ciertos IDs tienen un tratamiento especial*
- Un **TMD** (*Title Meta-Data*) que describe una versión concreta del paquete y su contenido
- Un **eTicket** (también llamado **ticket** o **tik**), la licencia que tenemos para acceder al software
- Una serie de archivos de datos, los **contenidos**, los cuales se pueden compartir entre títulos para ahorrar espacio
  - *Uno de los contenidos, el **contenido de arranque**, es el ejecutable principal*
- Una cadena de **certificados** que se usa para validar las firmas digitales

El TMD

El TMD es la base que se usa para firmar los contenidos, y contiene:

- El TitleID
- La versión del título
- El IOS a usar, si se aplica
- Un **descriptor de contenido** para cada contenido:
  - *Índice del contenido*
  - *ID del contenido*
  - *Tamaño*
  - *Hash **SHA-1** de los datos del contenido*

Los TMDs llevan firmas digitales RSA de 2048 bits

El eTicket

El eTicket nos da acceso a un título, con posibles restricciones. Contiene:

- El TitleID
- Una **clave** de 128 bits para desencriptar el título
  - *Esta clave a su vez va encriptada*
- Una serie de **límites**, como por ejemplo un límite de tiempo de uso

Los eTickets también llevan firmas digitales RSA de 2048 bits

Se distinguen dos variantes:

- eTickets comunes, para software de libre instalación, del sistema, o juegos de DVD
  - *Su clave está encriptada con una clave común maestra que comparten todas las Wiis*
- eTickets particulares, para software de la Tienda Wii
  - *Su clave está encriptada con una clave generada según un algoritmo basado en una clave privada de la Wii y un número aleatorio. En este caso, el eTicket lleva también este número*

Claves, claves, claves

En la Wii se utilizan varias claves criptográficas. Algunas son comunes a todas las consolas:

- La **Clave común** (AES), que sirve para encriptar las claves de los títulos comunes
  - *ebe42a225e8593e448d9c5457381aaf7*
- La **Clave SD** (AES), que sirve para ofuscar los datos de las tarjetas SD
  - *ab01b9d8e1622b08afbad84dbfc2a55d, IV: 216712e6aa1f689f95c5a22324dc6a98*
  - *Blanqueador MD5: 0e65378199be4517ab06ec22451a5793*
- El **Certificado raíz** (su clave pública RSA) que verifica a todo lo demás

Y otras son privadas y cambian de consola a consola:

- La **Clave NAND** (AES), que encripta el contenido de la memoria NAND FLASH de nuestra consola
- La **Clave NAND HMAC**, que firma la NAND FLASH
- La **Clave privada NG** (ECC), que es la clave que usa nuestra consola para firmar datos de SD y para obtener acceso a títulos protegidos, y la **clave pública NG** (ECC), que va certificada por Nintendo

IOS

IOS es el sistema operativo que se ejecuta en Starlet durante el uso normal de la Wii:

- Es el encargado de arrancar la CPU principal y cargarle el código de arranque
- Contiene los drivers para todos los dispositivos nuevos de la Wii, incluida la memoria NAND FLASH
- Se comunica con la CPU principal por una interfaz inter-procesador
- Realiza todas las operaciones criptográficas de bajo nivel, y es el encargado de la seguridad de la consola
- Se apropia de los 12MB altos de la memoria GDDR3 (esta memoria queda protegida)

Hay varias versiones de IOS instaladas en la consola, ya que cada título siempre usa la misma. Cada versión es un título de IOS distinto, y los llamamos IOS21, IOS30, IOS37, etc.

- Sin embargo, estos títulos tienen versiones propias, y sí se pueden actualizar individualmente.

IOS (cont.)

Algunos componentes de IOS importantes:

- **IOSP**, el kernel, es un sistema operativo (tipo microkernel en tiempo real) sobre el cual corren los demás drivers.
- **FFS**, *Flash FileSystem*, es el driver de la NAND FLASH y del sistema de archivos. Automáticamente encripta y desencripta y comprueba firmas.
- **ES**, *ETicket Services*, es el encargado de administrar los títulos y en particular la instalación en la flash.
- **DI** es el driver de la interfaz de DVD de la Wii, y en modo Wii también realiza el verificado y la desencriptación de los discos (transparentemente).

El proceso de arranque de la Wii

El proceso de arranque está diseñado para garantizar que el software que se ejecuta sea oficial de Nintendo:

- Primero arranca Starlet desde una ROM llamada **boot0**. Esta ROM forma parte del mismo **Hollywood**, y por lo tanto no se puede tocar. boot0 lee boot1 desde los primeros bloques de la NAND y lo comprueba contra una firma SHA-1 guardada en una memoria que sólo se puede programar una vez (en la fábrica).
- **boot1** contiene el código necesario para cargar y verificar boot2 como si fuera un título (boot2 es un título algo raro). Incluye las comprobaciones del eTicket, el TMD, las firmas RSA, etc.
- **boot2** reside en un área reservada de la NAND, y es prácticamente un IOS, pero sólo con el driver de la NAND. Su función es cargar el IOS real que se va a usar para ejecutar el menú del sistema

Este ingenioso sistema le permite a Nintendo cambiar boot1 en consolas nuevas cuando quieran, sin tener que tirar chips Hollywood ya fabricados.

boot2 se puede actualizar por software (es la primera parte del proceso de arranque que puede cambiar después de que la Wii deja la fábrica)

Y ahora, ¿que?

Hasta ahora, todo parece un sistema bastante bien diseñado

- En efecto, la mayoría del sistema de seguridad de la Wii tiene un diseño bastante bueno
- Sin embargo, sufre de bugs y problemas de implementación

Hay varios ataques que se han usado para romper el sistema de seguridad:

- *Twiizer Attack*
- *Fakesigning*, también conocido como las *firmas trucha*
- *Twilight Hack*

Twiizer Attack

En el modo de GameCube, pronto se pudo usar homebrew gracias a los chips

- Los juegos de GameCube no van firmados, así que es fácil insertar código casero
  - *Nos limita al modo de compatibilidad de GameCube, y no podemos hacer nada interesante*

Podemos atacar el hardware para conseguir información interesante

- En el modo de GameCube, se usa la parte baja de la memoria GDDR3 para emular la ARAM de la GameCube
- A la vez, los 12MB altos se reservan para MIOS, pero MIOS apenas usa unos kilobytes
  - *Antes de cargar MIOS, el IOS ha usado parte de esos 12MB para guardar... ¡**claves**!*
- No podemos acceder a esos 12MB directamente...
- ... pero podemos **puentear una conexión** en el chip de GDDR3 para leerlos como si fueran parte de la zona de ARAM

Nace el Twiizer Attack, que nos permitió extraer todas las claves importantes, incluyendo la clave común, gracias a la cual podemos:

- Desencriptar cualquier juego de Wii
- Desencriptar cualquier IOS u otro título común

Fakesigning o firmas trucha

Ahora que podemos ver un IOS en claro, podemos desensamblarlo y analizarlo

Se encontró un curioso problema en la comprobación de firmas de IOS:

- Las firmas digitales RSA de la Wii se crean generando un hash SHA-1 del contenido a firmar, y luego cifrándolo con RSA
- Al comprobar las firmas, se extrae este hash de la firma y se compara con el hash calculado
- Los hash SHA-1 son 20 bytes binarios…

Una grave metedura de pata: se comprueba dicho hash ¡como si fuera texto! Entonces:

- Las comparaciones de texto terminan en cuanto se encuentra un byte a cero
- Hacemos fuerza bruta al hash para que empiece por cero
- Y reemplazamos la firma RSA por ceros
  - *Debido a las propiedades matemáticas de RSA, cuando la firma es cero al comprobar salen ceros*

Con este fallo podemos generar firmas que la Wii considerará válidas. Se conocen como *firmas trucha* ya que el primer programa en hacer uso de ellas fue *Trucha Signer*.

Twilight Hack

Nos queda el problema de conseguir meter nuestro código en algún sitio para que la Wii lo ejecute

- Podemos crear discos caseros, pero para esto hace falta un modchip
- No podemos atacar la red ya que se usa SSL (y no es vulnerable al fallo)
- No hay forma de instalar un canal desde SD, ya que las copias a SD no incluyen el eTicket

Sin embargo, gracias al Twiizer Attack tenemos las claves privadas de nuestra consola

- Estas claves nos permiten generar saves válidos para otras consolas
- Podemos explotar fallos que tengan los juegos
- Estos fallos son muy comunes

Gracias a un fallo en el conocido juego The Legend of Zelda: Twilight Princess, podemos ejecutar código en el PowerPC

- Y desde ahí, hablar con IOS y decirle que instale canales, IOSes, o lo que sea

Con esto nace el Twilight Hack: un save del juego que, explotando el bug, carga un ejecutable desde la tarjeta SD.

- Actualmente es la única forma de empezar a ejecutar software casero en una Wii sin modchip

Contraataques

Nintendo ha ido solucionando algunos de los problemas:

- Las claves ya no se guardan en la GDDR3, sino en una memoria interna de Starlet
  - *Esto da igual, ya que ya tenemos las claves comunes y siempre podemos obtener las privadas usando otros métodos por software*
- En una actualización, Nintendo empezó a arreglar el bug de las firmas
  - *Esto nos imposibilita usar discos que usen el bug en el menú del sistema, pero sigue funcionando Twilight Hack y podemos usar IOSes que aún son vulnerables*
- Recientemente, han intentado detectar y eliminar el Twilight Hack
  - *Lo hicieron tan mal que a las 6 horas teníamos una solución*

Es muy dificil que consigan cerrar todo en actualizaciones futuras:

- Los exploits de los juegos son tan comunes que son prácticamente desechables
- Incluso si arreglan los fallos públicos de IOS, conocemos unos cuantos bugs que se momento no son públicos
- Nintendo tarda meses en producir y probar una actualización de Wii, mientras que nosotros podemos contestar en horas

Estado actual del homebrew

Tenemos un control casi total sobre la consola:

- Podemos ejecutar código en el PowerPC, instalarlo como canales, modificar juegos, etc
- Podemos crear código para Starlet nuevo o parchear el IOS

Sin embargo, todavía dependemos mucho del software de Nintendo:

- Estamos limitados por las restricciones del IOS oficial, a no ser que desarrollemos parches para eliminarlas
- Siempre tenemos que usar el IOS de Nintendo para hacer todo, ya que no existe una alternativa homebrew viable
- Dependemos de una serie de software durante el arranque. Si cualquier elemento del arranque falla (boot2, menú, IOS), la Wii quedará inservible (brickeada)
- Cualquier actualización de Nintendo puede cambiar la situación

Proyecto: firmware de recuperación

La idea: un código personalizado delante de boot2 que implemente un sistema de recuperación

- Si se pulsa un botón al arrancar, el sistema carga un menú desde la tarjeta SD y lo ejecuta (desde el cual se podría restaurar o hacer una copia de seguridad de la NAND)
- Si no se pulsa nada, el arranque es normal

Nos daría inmunidad contra cualquier actualización, siempre y cuando no se toque boot2

Estamos desarrollando una alternativa básica a IOS que de acceso directo a todo el hardware de la Wii, sin restricciones, para usarse en programas tales como:

- El firmware de recuperación
- Un extractor de claves
- Wii Linux

Futuro: parcheador de IOS inteligente

Un parcheador que dinámicamente parchee IOS desde boot2

- Un parche eliminaría la posibilidad de cambiar boot2
- Otro parche se usaría para mantener esta habilidad cuando se vayan cargando otros IOS
- Se puede integrar con el sistema de recuperación
- Podemos arrancar en limpio, sin parches, si fuera necesario
- Más flexible que ir instalando IOS parcheados

Con esto ya tendríamos un control prácticamente total sobre la consola y sus actualizaciones

Homebrew

Tenemos librerías para acceder a casi todo el hardware de Wii

- Gráficos, mandos, pads, red WiFi y ethernet, SD, USB, y mucho más
- El adaptador **USBGecko** crea un puerto serie virtual en la Wii y se comunica con el PC via USB

Herramientas de desarrollo:

- **devkitPPC**, la suite de compiladores para el PowerPC de la Wii
- **libOGC**, la libería que nos da acceso a casi todo el hardware de Wii (y de la GameCube)
- **libFAT**, para acceder al sistema de archivos FAT en SD y USB
- **devkitARM**, para compilar para Starlet

Cargadores:

- El **Canal Homebrew** (*The Homebrew Channel*), un cargador gráfico de aplicaciones desde SD
  - *También carga binarios por WiFi o por USBGecko*

Demostración

Demostración

Links

- Esta conferencia en Euskadi Digital
- El Canal Homebrew y el Twilight Hack
- Blog HackMii (posteamos bushing y yo)
- Comunidad EnTuWii
- Blog de tmbinc
- Wiibrew Wiki