Nintendo Wii Security Model

Pierre Bourdon

July 8, 2011

Introduction

- Sold more than any other home gaming consoles
- More than 2 percent of all sold consoles have been modded
- That's about 1 million consoles running code not approved by Nintendo
- Let's see how this strange situation happened

Plan

1. Wii goals
   - A full-fledged game console
   - Allowing firmware upgrades
   - A better Gamecube

Plan

1. Wii goals
   - A full-fledged game console
   - Allowing firmware upgrades
   - A better Gamecube

Modern consoles features

- Internet connection (with WiFi)
- Wireless controllers (Bluetooth)
- USB support for accessories
- Downloading games from an online store (Wii Shop)
- SD card support to backup game saves
- All in all, a lot of potential attack vectors

Downloading programs

- You can't rely on physical disc protection to avoid code execution
- There is also a need for cryptography and program signatures
- Everything installed on the console needs to be signed by Nintendo
- This has been a solved problem for at least 20 years... in theory

Plan

1. Wii goals
   - A full-fledged game console
   - Allowing firmware upgrades
   - A better Gamecube

What does the firmware provide?

- The firmware is basically the operating system of the console
- Drivers, support for new features, etc. are provided through upgrades
- Examples: USB keyboards, Wii MotionPlus
- But upgrades may break old code!

Multiple OS versions

- OS versions can be swapped at run time without rebooting
- Each Wii game specifies the OS version it was coded for
- Old OS versions are not removed on upgrades
- ... but then what about security upgrades?

Plan

1. Wii goals
   - A full-fledged game console
   - Allowing firmware upgrades
   - A better Gamecube

A better Gamecube

- Complete backward compatibility with Gamecube software
- One of the main goals when engineering the Wii
- Solves the egg-and-chicken problem with games at console launch
- Let's see how this was achieved

Hardware comparison

- GC CPU: 486MHz PowerPC (codename Gekko)
- Wii CPU: 729MHz PowerPC (codename Broadway)

- GC GPU: ATI 162MHz "Flipper"
- Wii GPU: ATI 243 MHz "Hollywood"

- GC RAM: 24MB system RAM
- Wii RAM: 24MB system RAM + 64MB external RAM

- See a pattern here?

Why is this so important?

- You can’t stay compatible with Gamecube games if you introduce new security features for them
- Nintendo chose to make a special "compatibility mode" where the hardware is basically stripped down
- This works really well... but again, in theory!
- This compatibility mode is the core of the first Wii exploit which I’ll explain later

Plan

2 Wii Internals

- Hidden CPU
- Encrypted data
- Broadway/Starlet interface
- Booting and chain of trust

Introduction

- Everything you wanted to know about the Wii but were afraid to ask
- Most of the informations in this part are not officially confirmed by Nintendo and is the result of research from independant hackers
- Thanks a lot to them for the great work!

Plan

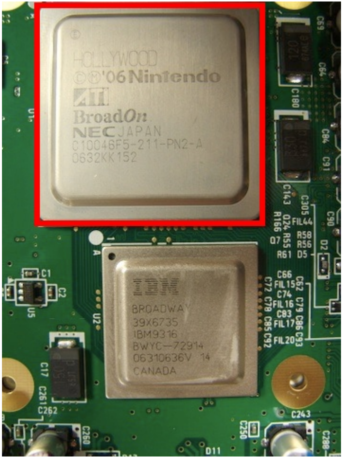
2 Wii Internals

- Hidden CPU
- Encrypted data
- Broadway/Starlet interface
- Booting and chain of trust

Where does the OS run?

- IOS is meant to run even when a game is running
- But the Wii "Broadway" CPU does not support multi-tasking
- Also, the IOS code does not look like PowerPC code at all

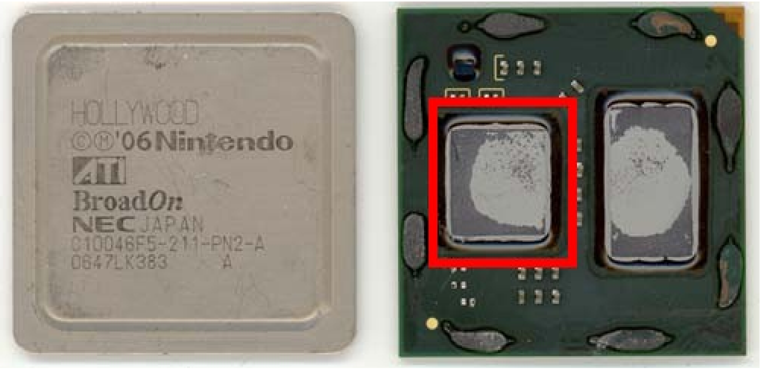
It’s actually in here



Hidden CPU

- The Wii actually embeds 2 CPUs, one for the game and one for the operating system
- The operating system CPU, codenamed "Starlet", is an ARM CPU
- It communicates with the Broadway using a simple `ioctl`-like API
- Nintendo never acknowledged the existence of this CPU to anyone

Closer view


HOLLYWOOD
©Ⓜ'06 Nintendo
ATI
BroadOn
NEC JAPAN
D10046F5-211-PN2-A
0647LK383 A

User space isolation

- Most platforms isolate user mode from kernel mode to avoid security problems
- On a regular PC, ring 0 / ring 3
- On a PS3 or an Xbox 360, the hypervisor which runs the game code
- The Wii program isolation is even better: two separate CPUs which do not directly share memory

Plan

2 Wii Internals
- Hidden CPU
- Encrypted data
- Broadway/Starlet interface
- Booting and chain of trust

Security coprocessors

- The Wii has two security coprocessors which allows for fast encryption and hashing
- The first one: an AES coprocessor which encrypts and decrypts data blocks with a 128 bit key
- The second one: a SHA1 coprocessor which hashes data blocks in very few cycles

The Wii NAND

- The NAND is a 512MB non volatile memory which stored any data needed by the console
- This includes for example IOS versions, game data, downloaded games, etc.
- The encryption key is different on every console and is burned into a chip at manufacturing time
- This way you can't access the data on the NAND easily, you need to first get the keys

One Time Programmable chip

- Also known as the OTP
- It is embedded in the Starlet (which is in the Hollywood chip...)
- Contains all the keys and certificates of the console, as well as some integrity hashes
- Really hard to access without being able to send code to the CPU

Game saves

- Game saves can be backuped on a SD card and then restored
- There must be some kind of integrity control to avoid modified saves
- Modified saves often imply exploits in games (we'll go into further details in the last part of this talk)
- The Wii uses classic asymmetric cryptography (RSA, PKI, etc.)

Game saves signing

- Each console has its own "save certificate" which is used to sign game saves
- This certificate is itself signed by Nintendo's certificate
- This chain of trust allows every console to sign valid game saves and to check if a game save was created on a Wii or was altered
- ... at least until the certificate can be extracted out of the OTP

Software installation

- Wii software is contained in packages called "channels": News Channel, Photo Channel, Internet Channel (Opera), ...
- Each channel is signed and the signature is checked at installation time
- Strangely the signature is not checked at execution time

Disc-based games signing

- Even games on DVD are completely signed to avoid data modifications
- Each byte on the DVD can be checked for integrity and alteration
- This is done through a data structure called a hash tree which allows for hierarchical integrity checks

Plan

2. Wii Internals
   - Hidden CPU
   - Encrypted data
   - Broadway/Starlet interface
   - Booting and chain of trust

IOS primer

- IOS is a "standard" microkernel programmed by BroadOn for the Wii
- It runs exclusively on the ARM CPU
- Each driver is associated with a `/dev/<node>` device file which is used to take commands or data
- For example, `/dev/sha` to communicate with the SHA1 coprocessor

Broadway on the Starlet

- There is a virtual process representing the code running on the Broadway in IOS
- All syscalls made by the PPC code to the Starlet are seen as if they were made by the virtual process
- This process UID depends on the game, so a game can't access another game data

Syscalls from the Broadway

- `open("/dev/sha", MODE_WRITE) / close(fd)`
- `read(fd, buffer, sizeof (buffer))`
- `write(fd, buffer, sizeof (buffer))`
- `seek(fd, where, whence)`
- `ioctl(fd, IOCTL_ACTION, p, sizeof (p), q, sizeof (q))`
- These can all be done synchronously or asynchronously

Everything is a file

- From the Broadway, the only syscalls available are those to manipulate files or device nodes
- Some device nodes provide access to real devices, some are just virtual devices exposing an API through `ioctl`
- For example: `/dev/di` is the disc drive interface
- ... and `/dev/es` is the authentication process which changes the UID of our process to the game UID

Plan

2. Wii Internals
   - Hidden CPU
   - Encrypted data
   - Broadway/Starlet interface
   - Booting and chain of trust

Booting process of the Wii

- The first step is to load boot0 from the OTP and run it on the Starlet
- boot0 loads boot1 from the start of the NAND, checks its SHA1 hash and runs it
- boot1 loads boot2 from the NAND and checks the signature before running it
- boot2 starts up the Broadway and launches the System Menu application

Implications

- boot0 is fixed, SHA1(boot1) is in the OTP, so boot0 and boot1 can't be upgraded with a firmware upgrade
- boot2 is the first piece of software that runs and can be upgraded
- If there is a bug in either boot0 or boot1, it's game over
- Guess what happened... we'll talk about this later

Plan

3 How the Wii got owned

- Finding the keys
- Executing arbitrary code in Wii mode
- Trucha signing bug
- Bannerbomb
- Post System Menu 4.3

Who did this?

- All the Wii reversing and modding efforts are due to only one team
- Team Twiizers: marcan, bushing, sven, segher, crediar
- Their more recent work include pwning the PlayStation 3 :)

First steps

- At the beginning, people did not even know about the Starlet
- The only way to execute code on a Wii was to use the Gamecube emulation mode, which is really limited because Starlet is disabled in that mode
- The first step was to get the keys to decrypt the NAND chip

Plan

3 How the Wii got owned

- Finding the keys
- Executing arbitrary code in Wii mode
- Trucha signing bug
- Bannerbomb
- Post System Menu 4.3

Solution

- Just try to find the keys in RAM when IOS uses them!
- Easy to say, very hard to do

Twiizer attack

- When in Gamecube emulation, IOS is basically "shut down" as it is not needed for anything
- However, it forgets to clean its memory address space before shutting down!
- The keys are still there, we just need to find a way to access them

Twiizer attack, part 2

- The Gamecube emulation mode can only access the first 24MB of RAM
- But you can take an electronic alimentation and modify the data sent on the address bus to the RAM
- Use that to access any address you want in the RAM!
- This way, people could start analyzing what was on the Wii NAND, including the operating system code
- They can also sign fake save games and transfer them on the console

How could Nintendo have avoided that?

- When you've got a really secure memory to store your keys, copying them in RAM is just dumb
- Ideally the SHA1 and AES coprocessors should have been able to read keys directly from the OTP
- Not cleaning up keys after using them is a big mistake

Plan

3 How the Wii got owned

- Finding the keys
- Executing arbitrary code in Wii mode
- Trucha signing bug
- Bannerbomb
- Post System Menu 4.3

Attack vector

- We can now craft game saves and modify them as we want
- Let's try fuzzing that to see how games react!

Tested game

- "That's the first game we actually tried to fuzz"

Great success!

- A too large horse name is `strcpy`-ed on a buffer on the stack
- The stack get smashed
- We can control the return address and jump into a shellcode
- From there we can code things to load an executable from a SD card

Twilight Hack

- Done once again by Team Twiizers
- Working version of this exploit which loads a binary from the SD card
- They also released a framework to easily exploit these kind of game loading errors
- Really famous exploit because it is very simple (buffer overflow with strcpy) and has huge consequences!

What did Nintendo do wrong?

- Definitely not enough testing and fuzzing on the data parsers
- No security features inserted by the compiler (-fstack-protector)
- No security features supported by the hardware (NX stack for example)
- They assumed that nobody would ever get their keys so game saves could be trusted to be valid

Plan

3 How the Wii got owned

- Finding the keys
- Executing arbitrary code in Wii mode
- Trucha signing bug
- Bannerbomb
- Post System Menu 4.3

What is that about?

- Biggest security fiasco ever on the Wii
- Bug in the function checking an executable signature before installing it or before executing boot2
- Could be used to compromise boot2 and patch it with some of our codes
- We break the chain of trust and control whatever is after boot2

Bug explanation

```
char valid_sha1[20], computed_sha1[20];

load_valid_hash(valid_sha1);
compute_hash(computed_sha1);

if (!strncmp(valid_sha1, computed_sha1, 20))
  return OK;
else
  return NOT_OK;
```

Bug explanation, part 2

- `strncmp` stops comparing at the first null byte
- With a lot of maths and RSA analysis, people finally figured out how to craft valid signatures usign this bug
- For the record, `memcmp` should have been used

Most importantly

- This bug is present into boot1
- boot1 cannot be upgraded as its hash is stored in the OTP
- Nintendo has no way to fix this bug!
- All consoles manufactured until at least 2008 have the Trucha bug

What should Nintendo have done?

- More code reviews before releasing code to several million Wii without being to upgrade it?
- There is a lot of public domain signing code that works and is not buggy that Nintendo could have used instead of rewriting their own
- Also, Nintendo were really slow about the boot1 update

Plan

3 How the Wii got owned

- Finding the keys
- Executing arbitrary code in Wii mode
- Trucha signing bug
- Bannerbomb
- Post System Menu 4.3

The end of the Twilight hack

- After two failed tries to stop the Twilight hack, the 4.0 System Menu upgrade "fixed" the Twilight hack exploit
- Actually, they only check the saves you copy to your Wii and don’t let you copy a badly formatted save
- A new exploit was needed to get arbitrary code execution on the Wii

Enters Bannerbomb

- System Menu 4.0 adds a new feature to the Wii: you can save channels (aka. apps) on a SD card and transfer them to your Wii
- On the channel list, the channel has a little animation defined by some kind of scripting language
- Bannerbomb uses an heap overflow in this animation parsing code to execute arbitrary code from the SD card

A real game changer

- Does not require any game! The bug is in the standard software installed on every Wii
- It's really safe and easy to trigger
- It is the most used exploit to install homebrew applications on a Wii

Nintendo's response

- First try to fix Bannerbomb in System Menu version 4.1, fail
- Second try to fix Bannerbomb in System Menu version 4.2, fail
- Third and final fix was finally a success in System Menu version 4.3

Plan

3 How the Wii got owned

- Finding the keys
- Executing arbitrary code in Wii mode
- Trucha signing bug
- Bannerbomb
- Post System Menu 4.3

Current ways to get code execution

- A lot of exploits which all require games
- *Indiana Pwns*, in *Lego Indiana Jones*
- *Smash Stack*, in *Super Smash Bros Brawl*
- *Eri HaKawai*, in *Tales of Symphonia 2*
- *bathaxx*, in *Lego Batman*
- *Return of the Jodi*, in *Lego Star Wars*

Nintendo response

- Currently, none at all!
- System Menu 4.3 is there since June 2010 and no security update was released since then
- Maybe Nintendo finally understand that with all the foundations of their security pwned it's almost impossible to avoid exploits :)
- Even if they fixed the current exploits, finding and exploiting a new one is a matter of hours.

Plan

4 Writing your own exploit

- Finding an exploitable game
- Using the Twiizers savezelda payload
- In Tales of Symphonia 2

A bit of infos about me

- I'm the author of the *Eri HaKawai* exploit
- In this next section I'll talk more precisely about how I found the vulnerable code and how I exploited it to make it a viable way to execute code on a 4.3 Wii

Plan

4 Writing your own exploit

- Finding an exploitable game
- Using the Twiizers savezelda payload
- In Tales of Symphonia 2

Very few games are not exploitable

- Try to avoid very recently released games
- Take a game save and try very large string values to make it overflow
- You'd be lucky to find a game that does not crash :)

What is required for a working exploit

- An overflow which will overwrite the return value of the function without making the function crash before returning
- Enough space elsewhere in the game save to store the shellcode (64KB)
- An emulator to check the address where all those things are loaded in memory

Plan

4 Writing your own exploit
- Finding an exploitable game
- Using the Twiizers savezelda payload
- In Tales of Symphonia 2

Savezelda

- Released with the Twilight Hack
- A "generic" payload which should be placed in the game save and be loaded by the buffer overflow
- Loads a binary from the SD card

Savezelda, part 2

- To compile it correctly you need to know where it will be placed in memory when executed
- Choose an address from the game save memory space
- Finding the right address to load the code and jump on it can be kind of difficult if you only have real hardware

That's all!

- savezelda does everything
- We did not even write PowerPC assembly :)
- This payload is really a great asset for the Wii modding community
- The process of finding an exploitable game and exploiting it takes probably less than 3 hours if you know all the tools

Plan

4 Writing your own exploit

- Finding an exploitable game
- Using the Twiizers savezelda payload
- In Tales of Symphonia 2

Where is the overflow

- When you try to overflow the characters name, it is correctly handled
- When you do the same on monster characters, it looks correctly handled
- When you actually try to do actions on monster characters with overflowed names, strange things happen
- Displaying the monster status smashes the stack

Where to store the payload

- There are large unused spaces in the save file filled with zeros
- The payload is about 30K and can be stored without any problem

How to overflow the buffer

- First approach: insert a lot of FF bytes with the return address at the right offset
- Crashes because local variables are corrupted
- Second try: insert 01 bytes instead of FF bytes
- Curiously this works better :)

Demo time :)

Plan

5 Conclusion

A lot of good ideas

- The Wii security model has a lot of very good ideas
- But also a lot of very poor implementations of their ideas
- If you distribute a product to millions of people, your weakest security link will definitely break some day

You can't avoid hackers

- So maybe you could embrace them?
- The PS3 resisted for a long time because it offered an almost native Linux support
- If you give access to your hardware to hackers, chances are they will take a lot more time to reverse the rest of your security :)
- Homebrew applications are not only dream: XBMC, emulators, etc.

Thanks for your time!

- `http://blog.delroth.net/`
- `@delroth_` on Twitter

- Questions?