History of Game Console Security

Jesus Nuñez
Colorado School of Mines Linux Users Group
September 8, 2022
lug.mines.edu/attend
What consoles?
Consoles

1. Atari 2600
2. NES
3. SNES
4. Dreamcast
5. N64
6. Gamecube
7. Wii
8. PS1-3
9. Xbox-360
Why secure consoles?

Simple answer. Piracy.
Piracy = less games sold
less games sold = less money made
Consoles are sold at a loss. Selling a console at hardware cost would make them more expensive and ultimately sell less. More money is made from selling the consoles at a loss and making up the money in game sales.
early security
The light cube

Early security was heavily reliant on physical media. Hardware solutions were very simple.

- Security code or word
- Manuals with *vital* game information
- Puzzles that ship with the game
- Dedicated chip for authentication
- Light refraction!
Early security was heavily reliant on physical media. Hardware solutions were very simple.

- Security code or word
- Manuals with *vital* game information
- Puzzles that ship with the game
- Dedicated chip for authentication
- Light refraction!
Early security was heavily reliant on physical media. Hardware solutions were very simple.

- Security code or word
- Manuals with *vital* game information
- Puzzles that ship with the game
- Dedicated chip for authentication
- Light refraction!
The light cube

Early security was heavily reliant on physical media. Hardware solutions were very simple.

- Security code or word
- Manuals with *vital* game information
- Puzzles that ship with the game
- Dedicated chip for authentication
- Light refraction!
Early security was heavily reliant on physical media. Hardware solutions were very simple.

- Security code or word
- Manuals with *vital* game information
- Puzzles that ship with the game
- Dedicated chip for authentication
- Light refraction!
The cartridges were essentially computers themselves. They contained manually placed bad sectors in their memory. There were intentional duplicate sectors. Sectors were sometimes unlabeled, and also out of order!
Key chip in every cartridge followed by a lock device inside the console.
FIG. 5
START

S1
POWER ON?

NO

YES

RESET MAIN UNIT AND MAINTAIN RESET STATE (LOCK DEVICE)

S2

S3
SYNCHRONIZE LOCK DEVICE WITH KEY DEVICE

MAKE LOCK DEVICE PREDETERMINED OPERATION

MAKE KEY DEVICE PREDETERMINED OPERATION

S4

DATA TRANSFER BETWEEN LOCK DEVICE AND KEY DEVICE

COMPARE RESULTS OF OPERATIONS

S5

S6

S7

COINCIDENCE?

NO

YES

S9

S8
RELEASE RESET STATE OF MAIN UNIT

CONTINUE RESET STATE OF MAIN UNIT

FIG. 6A

LOCK DEVICE
OPERATION START IN RESPONSE TO POWER ON OR RESET

S10

LOCK?

S11

NO

YES

RESET GAME, PPU AND KEY DEVICE

RELEASE RESET STATE OF KEY DEVICE, SYNCHRONIZE BOTH DEVICES

OUT PUT RANDOM NUMBERS

READ DATA

PROCESS OR OPERATE INPUT DATA

OUTPUT RESULT

READ RESULT

COMPARE BOTH RESULTS

S12

S13

S14

S15

S16

S17

S18

S19

S20

S21

KEY DEVICE

KEY?

NO

YES

OUTPUT RANDOM NUMBERS

READ DATA

PROCESS OR OPERATE INPUT DATA

OUTPUT RESULT

READ RESULT

MAKE OPERATION SIMILAR TO OPERATION OF STEPS S19-S28, HOWEVER, NOT INFLUENCE MAIN UNIT TO BE RELEASED OF RESET STATE THEREOF
How to bypass? You don’t. A chip is needed to run games. One can be harvested from a donor board. This was used in the NES, SNES, and N64. The NES and SNES version were cracked in 2006. The one used in the N64 was cracked in 2015.
The Chip in the N64 was special. There was no matching chip on the console itself. Instead it was baked into a chip containing code that first runs when the console is on. On top of these there were 5 variants of the chip in the wild.
First and foremost the disk. It is not normal. The discs have a copy protection ingrained in them and the drive firmware is encrypted. Each disc has an identifier burned right into the plastic which is not possible to replicate with a dvd burner. This region is called the Burst Cutting Area (BCA).
Small Disc

BCA
Dreamcast discs were built different. Sega made a disc called GD-ROMs. A normal dvd player will only read track 1 of the disc that contained a message. Dreamcasts also did not check for signed code.
Discs
Discs
Hypervisor: Bottom level code running in kernel mode. Incharge of verifying everything that runs. Think about it as the manager of a Virtual Machine.

Chain of Trust: An order of programs to run that check on each other to verify integrity.

Firmware: Code that runs inside a micro processor.
Implemented a chain of trust. System Startup runs and hands off to Windows on a ROM chip which then goes into your game. There are checks between every handoff. Main weakness, System Startup code was in FLASH memory (IE Writable/replacable) uh oh.
The playstation one had a simple defense. Region locking as well as disc verification.
All discs had irregular data used to verify if the game is legit. All discs also had the wobble from the Dreamcast this is often known as the watermark.
There were also unreadable sectors in the PS1 where a normal disc reader would not look at.
The PS2 implemented the same wobble from the PS1. The PS2 also constantly asked the Disc for information to validate the use of the correct disc. This was solely aimed at stopping the swapping attack. Sadly all this CD protection failed because the USB was left with essentially a direct path to the CPU.
Microsoft went above and beyond in security. The Xbox 360 had a hypervisor, chain of trust, signing, and fuses. Updates could cause a fuse to blow stopping any downgrading from ever occurring. The DVD drive was locked down in firmware hidden in a chip making it hard to flash and read. Xbox 360 also implements a global key tree where many keys are derived from a per console key. The master key is in the CPU in read only and kept away from anything that can read it. From this key other keys are derived.
360 Boot Process

The boot process was

1BL → CB_A → CB_B → CD → CF → CD → HV → Kernel → Dashboard

1BL is in the CPU and is the Boot Loader it verifies CB_A and runs it.

CB_A loads and decrypts CB_B and verifies it.

CB_B starts a virtual machine disables all debug ports and verifies and runs CD.

CD then verifies CF and loads it and jumps to it.

CF then verifies, loads, and jumps to CG. CG is decrypted using a key from CD. Once this is done it returns to CD which then goes to the Hypervisor.

The hypervisor then verifies the Kernel and Dashboard, starts them and the gaming can begin.
Fuses
Playstation 3

Implements the same as the Xbox 360 minus the fuses. The PS3 shipped with a very special feature! LINUX. With Linux in the console already, the homebrew/Linux community was not targeting the console as they had an avenue already.
A break from security let’s see what Nintendo has been doing!
The wii contains real Gamecube hardware to run gamecube games!
All Gamecube bugs/exploits still present.
Learning from the GameCube, Nintendo actually signed their code and sandboxed the Gamecube code so nothing gets out of gamecube mode.
Below is an image of how the OS handles memory when using Gamecube games. The gamecube only uses 16MB of RAM and the rest is disabled. This ultimately "sandboxed" the Gamecube code from any Wii code. Or so Nintendo thought. Nintendo never cleared the RAM so people took a pair of tweasers and went pin by pin on the RAM chip to enable the RAM and dump the data.
Wii Chain of Trust

Starlet (ARM)
- boot0
- boot1
- boot2

SysMenu IOS

Game/Title IOS

Broadway (PPC)
- System Menu
- NANDLoader
- Game/Title

PPC hard reset

WiCi CHAiN OF TRUST
Lesson in cryptography/cybersecurity. This function halts if it runs into a null byte. Thus you can guess and check byte by byte really fast.
This is a Wii clone. It runs a new OS that isolates games ran in comparison to the Wii which ran games on the hardware. Memory is also isolated (and cleared this time).
The ARM chip still exists to run code/boot but it is bigger better and fixes the issues from before. Games are also signed and no longer use the strncmp function. It also checks everything ran instead of just discs. (The Wii U has Wii hardware).
No longer on PowerPC but instead on the Nvidia Tegra X1 chip. An old chip even at the time of release. The Nvidia provides a lot of features including boot!. Nvidia is bigger and smarter than Nintendo at this so they did it right. The most famous hack for this console involves only a paperclip!
Back to Actual Security
The PS4 has a lot of things. It is x86, FreeBSD, Webkit, and a Security Chip. It however does not have a Hypervisor. Thus to hack the PS4 you need either a FreeBSD hack or a WebKit hack. Many people went to attack WebKit because FreeBSD was heavily modified.
I don’t know much of the security. It has not been entirely discovered/disclosed yet. HOWEVER
The PS5 has been hacked not once but Twice. Using two separate attacks that are undisclosed.
Sadly i do not know anything about these consoles specifically but from an Xbox talk i found they seem to have just done the Xbox360 but actually fixed all its issues. Do keep in mind that NO ONE has broken the Xbox 360 on a software level. It is still perfectly secure software wise.
My favorite Hacks

- The kamakaze hack
- Reset Glitch hack
- Fusee-Gelee
- Browser hacks
Xbox 360s had keys in the DVD drive to unencrypt games and stuff. To stop writing to it, it was read only. A chip was in charge of keeping the read only pin on. So you have a chip that powers a pin that stops you from writing firmware. How do you turn it off?
Guess where the pin was!
Every CPU has a reset pin. This reset pin resets the CPU! The Xbox 360 has a reset pin. It was discovered that if you send a 5 volt signal about 1-5 KHz down the pin you can get the Xbox 360 to run random code. The code it runs is code you deposit in the NAND where the kernel and OS are in. You use a NAND writer to get it in there.
There is a hardware and software issue in the Nvidia Tegra chip. Shorting the pins as seen below will put the switch in recovery mode and allow for unsigned code to run!

There is no fixing this issue without a new hardware revision. The best type of exploit!
Browsers issues

- The PS Vita
- The Playstation 4
- The Playstation 5 most likely
- The nintendo Wii U
- The nintendo 3DS
Lessons learned through history?

- Do not trust any data from over a bus!
- Do not use Flash for code that executes!
- If the memory is not in the CPU or GPU do not trust it.
- Security processor with it's own in silicon ram, storage, etc
- Do not assume data written to HDD is the same as data Read from the same spot just seconds later
- Continuously check for security. (No one time checks)
Lessons learned through history?

- Do not trust any data from over a bus!
- Do not use Flash for code that executes!
  - If the memory is not in the CPU or GPU do not trust it.
  - Security processor with its own in silicon ram, storage, etc
  - Do not assume data written to HDD is the same as data Read from the same spot just seconds later
  - Continuously check for security. (No one time checks)
Lessons learned through history?

- Do not trust any data from over a bus!
- Do not use Flash for code that executes!
- If the memory is not in the CPU or GPU do not trust it.
  - Security processor with it's own in silicon ram, storage, etc
  - Do not assume data written to HDD is the same as data Read from the same spot just seconds later
  - Continuously check for security. (No one time checks)
Lessons learned through history?

- Do not trust any data from over a bus!
- Do not use Flash for code that executes!
- If the memory is not in the CPU or GPU do not trust it.
- Security processor with it's own in silicon ram, storage, etc
  - Do not assume data written to HDD is the same as data Read from the same spot just seconds later
  - Continuously check for security. (No one time checks)
Lessons learned through history?

- Do not trust any data from over a bus!
- Do not use Flash for code that executes!
- If the memory is not in the CPU or GPU do not trust it.
- Security processor with it’s own in silicon ram, storage, etc
- Do not assume data written to HDD is the same as data Read from the same spot just seconds later

- Continuously check for security. (No one time checks)
Lessons learned through history?

- Do not trust any data from over a bus!
- Do not use Flash for code that executes!
- If the memory is not in the CPU or GPU do not trust it.
- Security processor with it's own in silicon ram, storage, etc
- Do not assume data written to HDD is the same as data Read from the same spot just seconds later
- Continuously check for security. (No one time checks)
This presentation was from the **Mines Cyber Security Club**. A mostly-complete archive of our presentations can be found online at [https://oresec.mines.edu](https://oresec.mines.edu).

Individual authors may have certain copyright or licensing restrictions on their presentations. Please be certain to contact the original author to obtain permission to reuse or distribute these slides.