History of Game Console Security

Jesus Nuñez
Colorado School of Mines Linux Users Group
October 25, 2021
oresec.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
Simple answer. Piracy.
Piracy = less games sold
less games sold = less money made
Why is piracy such a big problem?

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
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!
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!
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

S2
RESET MAIN UNIT AND MAINTAIN RESET STATE (LOCK DEVICE)

S3
SYNCHRONIZE LOCK DEVICE WITH KEY. DEVICE

S4
MAKE LOCK DEVICE PREDETERMINED OPERATION
MAKE KEY DEVICE PREDETERMINED OPERATION

S5
DATA TRANSFER BETWEEN LOCK DEVICE AND KEY DEVICE

S6
COMPARE RESULTS OF OPERATIONS

S7
COINCIDENCE? NO
YES

S8
RELEASE RESET STATE OF MAIN UNIT
CONTINUE RESET STATE OF MAIN UNIT

FIG. 6A

LOCK DEVICE

S10
OPERATION START IN RESPONSE TO POWER ON OR RESET

S11
LOCK? NO
YES

S12
RESET GAME CP, PPU AND KEY DEVICE

S13
RELEASE RESET STATE OF KEY DEVICE, SYNCHRONIZE BOTH DEVICES

S14
OUTPUT RANDOM NUMBERS

S15
READ DATA

S16
PROCESS OR OPERATE INPUT DATA

S17
OUTPUT RESULT

S18
READ RESULT

S19
COMPARE BOTH RESULTS

S20
COINCIDENCE? NO
YES

S21
RELEASE RESET STATE OF 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
It had a cool wiggle in the CD used for protection. The read head saw the wiggle and each disc had a certain wiggle in the disc. This wiggle is seen as error by any normal DVD reader. The Dreamcast measured this wiggle to determine if the copy was genuine. A normal DVD writer can’t replicate it. This failed ultimately because you could just play a legit game. Then you would just swap the disc with your pirated copy and it would just run!
The gamecube part runs in a sandbox. Implemented Code signing with a per console master key. Each game is SHA-1 signed by Nintendo. If the SHA is good run the game if not don’t. Save files were also SIGNED too.
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.
Xbox OG

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.
Playstation 1

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.
Playstation 2

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.
Key Tree

Root Key

Version 3 SP
- Version 3 SP ERA Devkit
- Version 3 SP SRA Devkit

Version 4 SP
- Version 3 SP Retail

Version 3 SP Retail
- Version 6 OS

Version 3 SP Retail
- Version 7 OS
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.
Consoles since the Xbox 360 and the Playstation 3 implement the same security.
The Xbox Hyper visor was never broken itself. And the Playstation 3 had many other issues not related to it’s hypervisor. Any issues found have been fixed and the systems themselves have been hardened for future consoles.
The Xbox X, Xbox series X, Playstation 4 and Playstation 5 all implement this hypervisor and chain of trust.
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
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
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
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.
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
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.