How the Nintendo Wii Security Was Defeated

By vedard

Published: 2023-08-06 · Archived: 2026-04-05 18:46:26 UTC

This is the story and the technical details of how the hacker group named Fail0verflow (formerly known as Team Twiizer) discovered and exploited numerous vulnerabilities to defeat the security mechanisms of the Nintendo Wii.

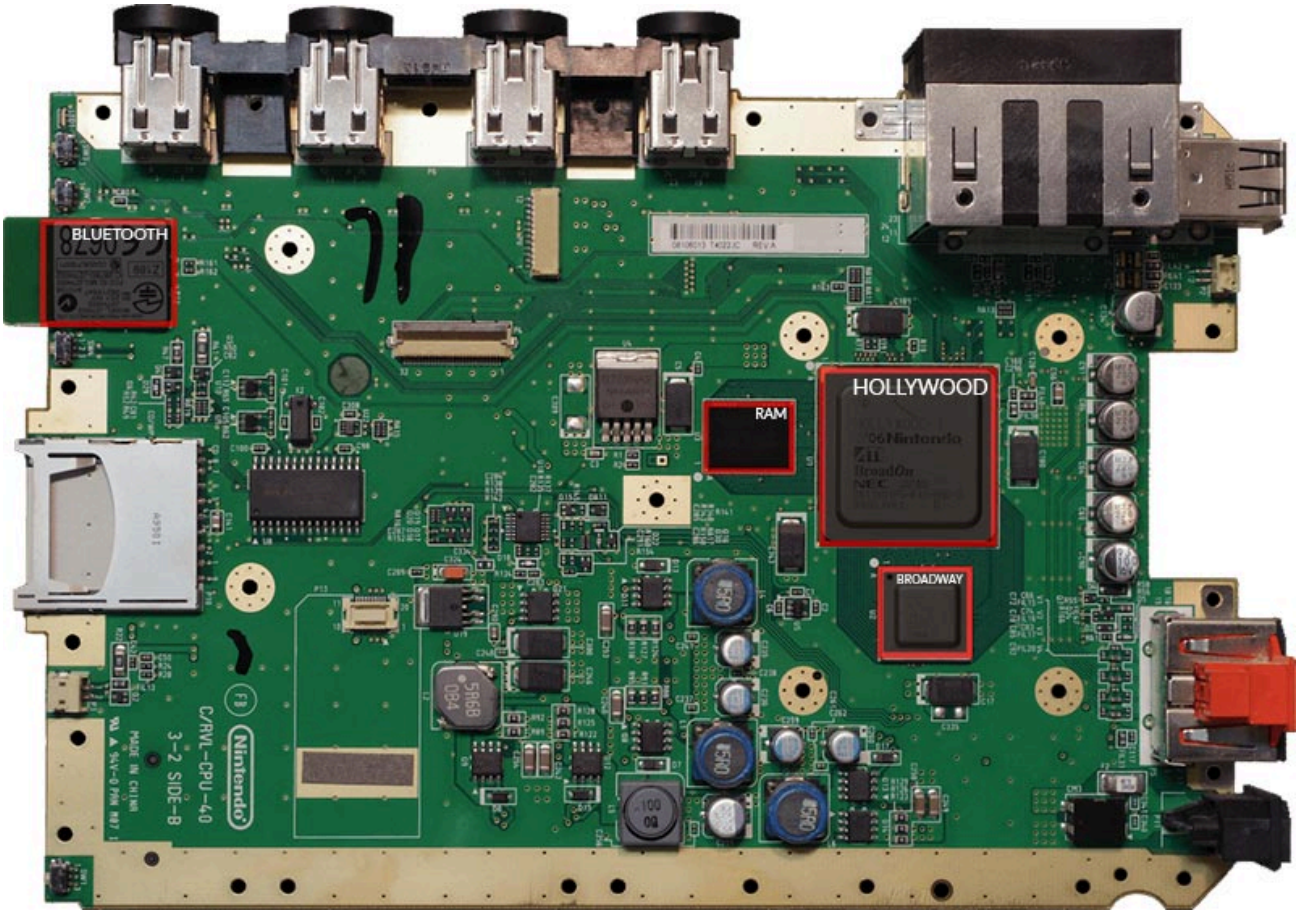
The Architecture of the Nintendo Wii

Before delving into the security flaws of the console, it's essential to understand the design of the Wii and the security mechanisms implemented by Nintendo.

Hardware

The Nintendo Wii was released towards the end of the year 2006 and features the following hardware.

- **Storage**: The Wii includes 512 MB of NAND flash memory, serving as internal storage for system software, applications, game saves, and user settings. The majority of data stored in the NAND flash is encrypted using a unique key specific to each console.
- **CPU**: The Wii's processor is the "Broadway" CPU, a 729 MHz processor manufactured by IBM using the "PowerPC" microarchitecture. It handles running games and applications on the console.
- **GPU**: The Wii is equipped with the "Hollywood" GPU, manufactured by ATI, responsible for handling graphics processing.



The Nintendo Wii's motherboard

The Secret Coprocessor

Hidden inside the GPU Hollywood is a second CPU nicknamed as "Starlet." This ARM processor serves as an IO bridge for games executed on the Broadway processor and the Wii hardware components (NAND flash, wireless chip, disk drive, etc.). Starlet runs its own program called IOS, which stands for Input/Output System. This program is responsible for all the security mechanisms of the Wii, with hardware implementations of cryptographic functions such as SHA1 (hashing), AES (encryption), and RSA (signing).

Initially, the existence of the Starlet CPU wasn't known. Developers would access its functionality through the Nintendo's SDK completely unaware of its presence. It wasn't discovered until a year and a half after the Wii's release when hackers stumbled upon ARM instructions within the binaries distributed by Nintendo during a system upgrade.

Title and Ticket

All the games and applications for the Wii are referred to as "Titles." When these are distributed, either through Wii discs or the Wii Shop, they are encrypted with an AES key called the "Title Key." This key is provided with the game upon purchase in the form of a "Ticket." The ticket, in turn, is encrypted with AES using the master key called the "Common Key," which is universal and stored securely in the console. In addition to encryption, the titles are also signed with an RSA certificate, and the private key for this signature is known only to Nintendo, preventing modifications.

Titles run on the Broadway processor, which is much more powerful and suitable for games. However, this processor cannot directly communicate with the Wii's hardware. For instance, if a game intends to utilize the Wii's storage to save a player's

progress, it must do so through the IOS API and won't be able to save everything everywhere. IOS acts like a firewall to restrict what the game processor can do.

The System Menu on the Wii is the primary graphical interface, allowing the player to select a title to launch. It is not an operating system but simply another title that runs barebone on the Broadway processor with "elevated" permission. This allows it, for example, to request IOS to install, delete, or launch a title.

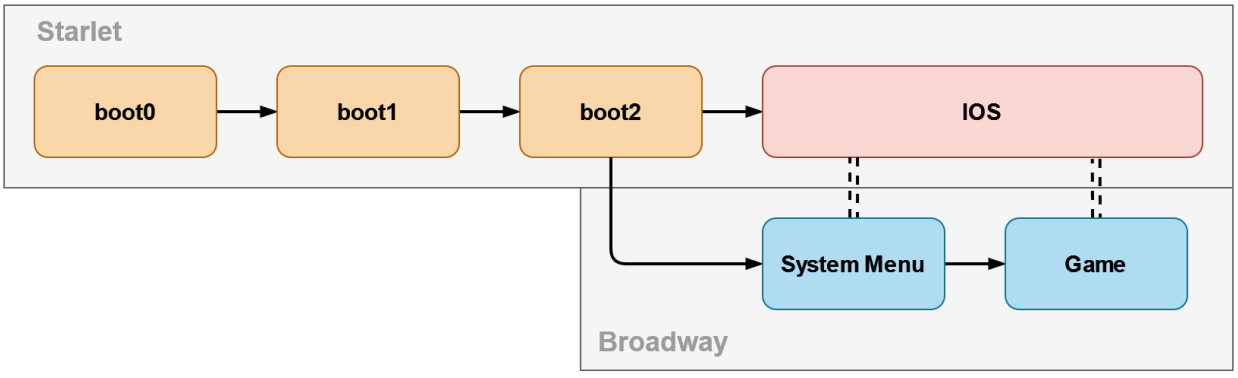
The System Menu

It's worth noting that only one title can run at a time on the Broadway processor, so IOS will terminate the System Menu to launch a new title.

The Boot Process

During the power-on process of the Nintendo Wii, the Starlet CPU executes a boot loader chain before the Broadway CPU is activated.

- **boot0**: Small boot loader situated in a read-only memory (ROM) within the Hollywood chip. Its purpose is to decrypt and validate boot1 using an AES key and SHA1 hash stored in the OTP chip.
- **boot1**: Located at the beginning of the NAND storage, boot1's role is to decrypt, verify, and load boot2.
- **boot2**: Responsible for loading the System Menu from the NAND storage and the appropriate version of IOS.



The Wii's boot process

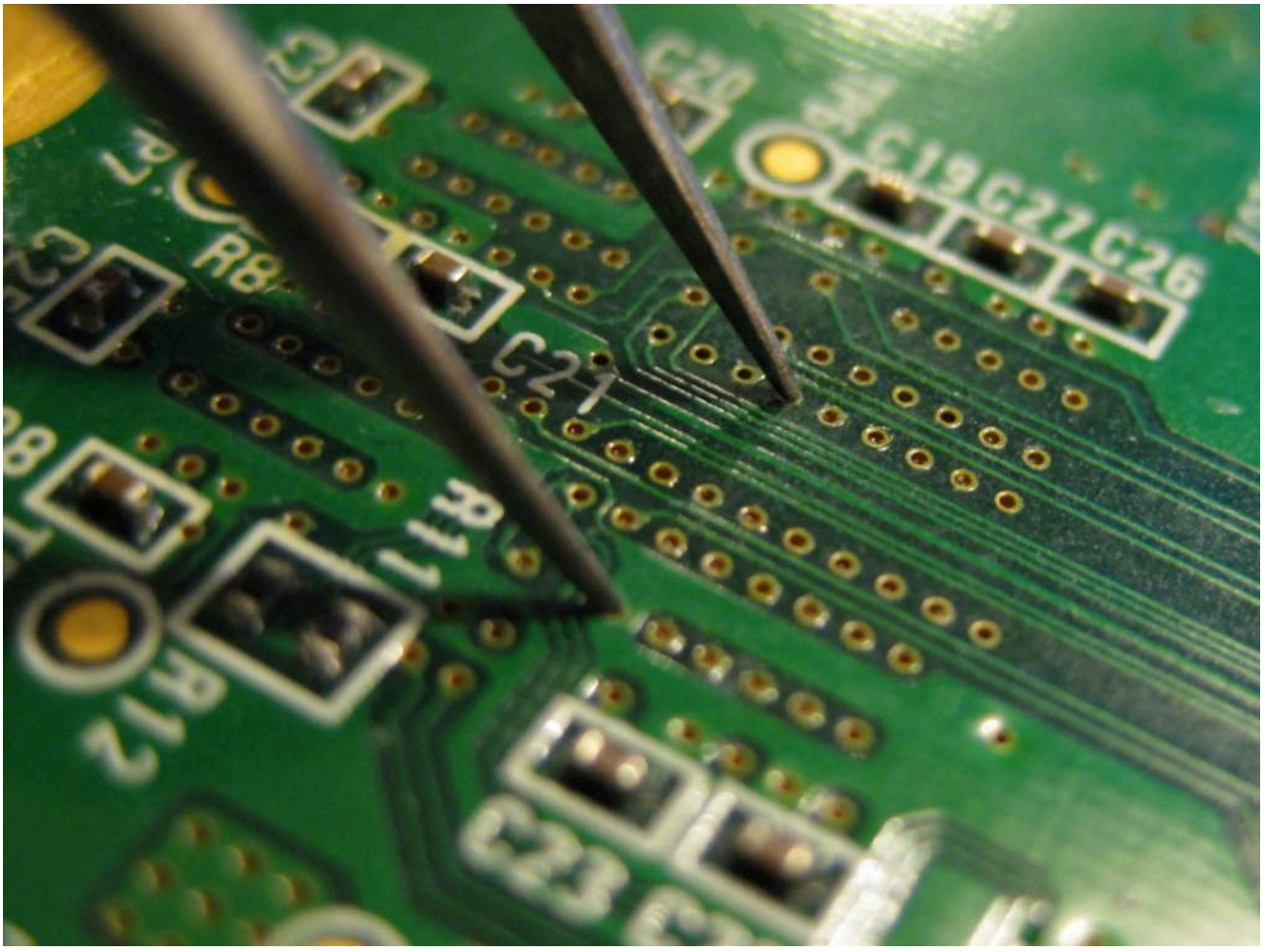
All boot loaders are stripped-down versions of IOS, so the same security flaw can sometimes be found across different boot loaders. Furthermore, Nintendo cannot remotely update boot0 and boot1. This operation can only be done at the factory.

Exploitation and vulnerability

The Tweezer Attack

The Nintendo Wii offers full backward compatibility with the Gamecube. GameCube titles are not signed, and when they are launched, the Broadway processor reboot into a restricted “Sandbox” mode, disabling most new functionality from the Wii and limiting access to only 16 MB of the 64 MB memory.

In 2007, Fail0verflow discovered that it is possible to expose other sections of the RAM by shorting specific lines beneath the chip using a pair of metallic tweezer. This attack slide the 16 MB of accessible memory, allowing them to sequentially dump portion of the memory via a serial port soldered onto the Gamecube controller connector.



A pair of tweezer shorting an address line underneath the RAM chip

Since the sandbox was never intended to access the other 48 MB of memory, this space remained unclear and contained the IOS code along with some cryptographic keys. This includes the famous “Common Key” mentioned earlier, which is used to decrypt tickets and titles.

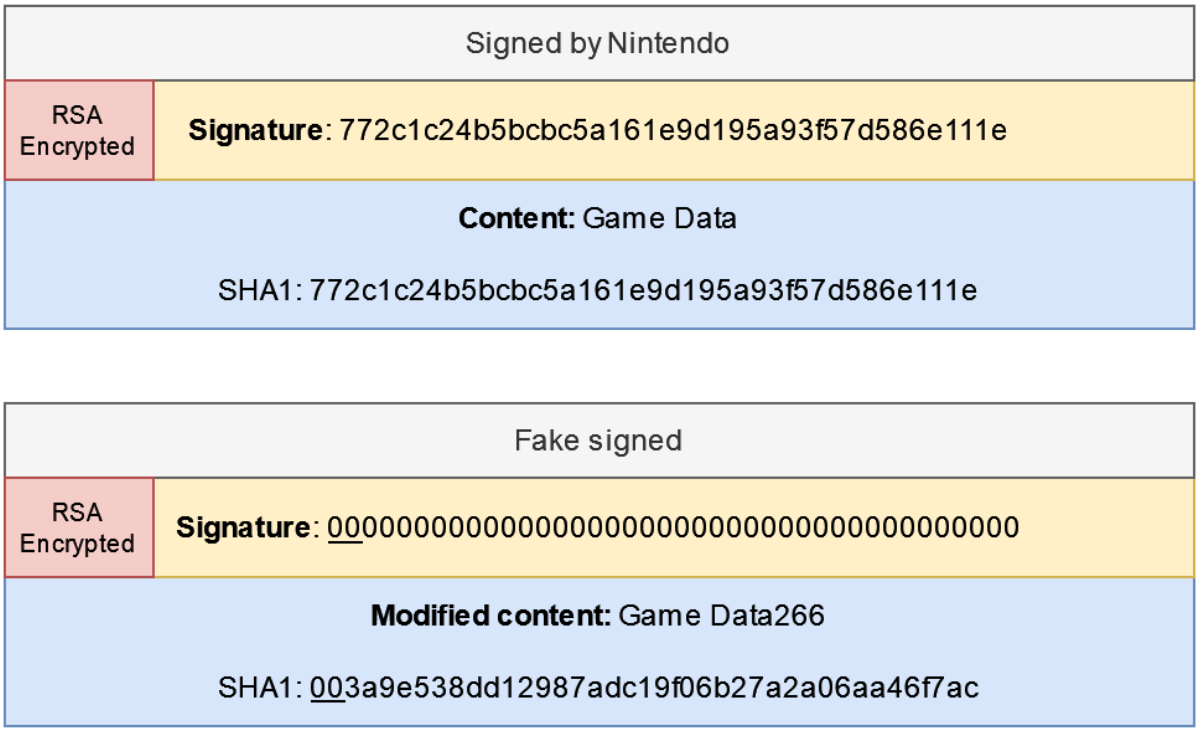
The Signing Bug

By carefully examining the IOS machine code extracted through the tweezer attack with a disassembler, a serious flaw was discovered in the module responsible for validating the signature of a Wii title.

Nintendo uses RSA to sign a Wii title, where a SHA-1 hash of the game content is encrypted with Nintendo’s private key and distributed as a signature. The Wii validates the title by decrypting the signature using the public key and comparing it to the actual hash of the content proving that it had not been modified.

However, the Wii used a string comparison function (similar to `strncmp`) to compare the hashes instead of a function for binary values. And since strings in C are terminated by a null byte (`0x00` or `\0`), the comparison can end early with a successful result if both the signature and the hash start with the null byte.

1. The signature of a title can easily be replaced with zeros. This is because RSA by design uses an exponent operation to encrypt data, and 0 to the power of anything is always 0.
2. The hash can quickly be brute-forced by adding padding to the game content.



Comparison of signed title vs a fake signed title

Here's a Python script demonstrating how to brute force the first byte of SHA1 hash:

```
import hashlib

original_content = b"Game Data"
padded_content = original_content
padding = 1

while not hashlib.sha1(padded_content).digest()[0] == 0x00:
    padded_content = original_content + str.encode(str(padding))
    padding += 1

print(padded_content) # b'Game Data266'
```

The vulnerability is present everywhere, and as a consequence, it allows for the execution of unsigned games, the installation of custom titles or custom IOS, and even the modification of boot2. It was eventually patched in IOS by system update and in boot1 at the factory only for new Wii manufactured after some point in 2008.

The Twilight Hack

In “The Legend of Zelda: Twilight Princess,” players have the opportunity to rename their horse. Fail0verflow discovered that while the interface limits the number of characters for the name, the check is not implemented when loading up a save file and can be exploited to cause a buffer overflow.

Save files can be imported from an SD card, but these files are encrypted and signed by IOS to prevent players from easily tampering with them (e.g., adding lives, unlocking items). However, with the knowledge of the keys obtained from the tweezer attack, it becomes possible to craft a correctly signed save file.

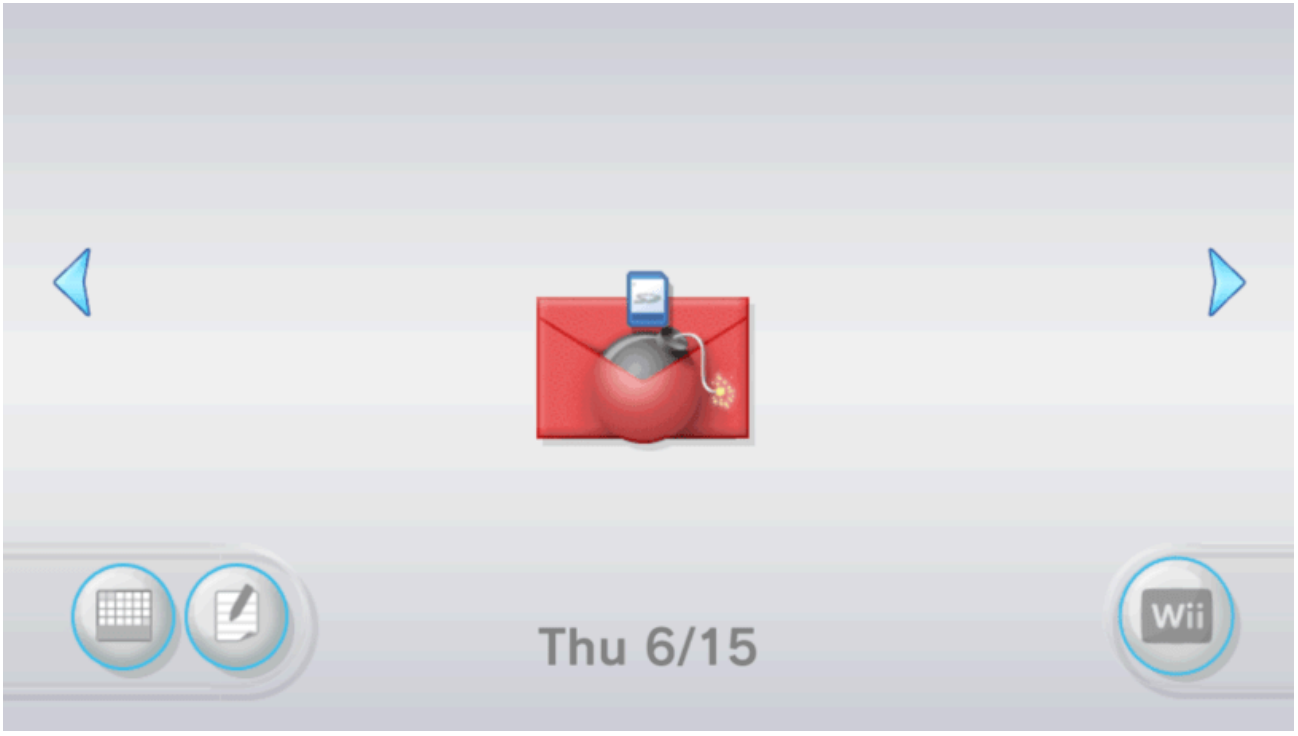
Triggering the buffer overflow to crash the game is pretty simple. However, running custom code is far more challenging.

Twilight Princess with debugging tools enabled

With the signing bug, Fail0verflow was able to patch the game to re-enable all debugging functions, allowing them, for example, to view the content of memory registers. Ultimately, they managed to create the right payload to achieve code execution simply by loading a saved game. It was the first publicly available exploit and opened doors for many similar exploits to be discovered in other games. It was, however, patched by Nintendo a year later.

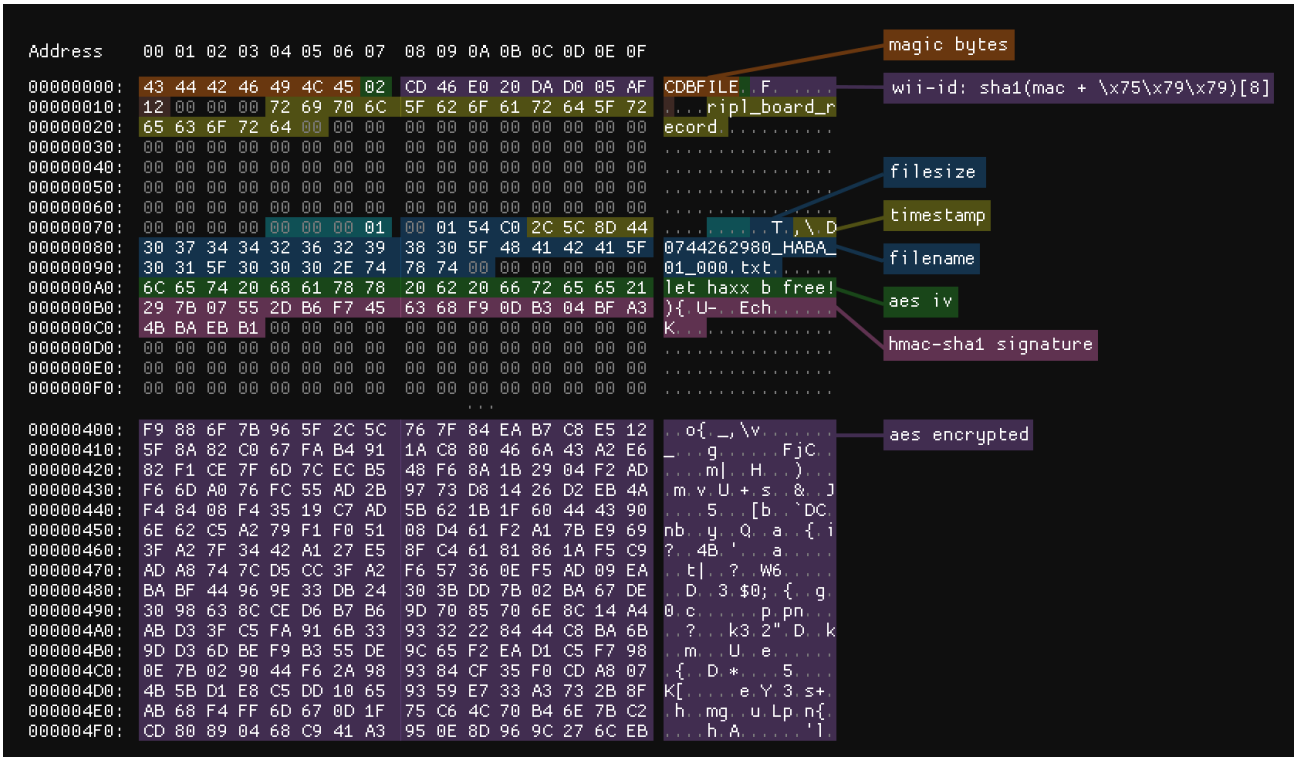
The LetterBomb

The “LetterBomb” exploit is comparable to the “Twilight Hack,” as it enables the execution of codes on the Broadway processor. However, unlike the latter, it’s not necessary to possess a specific game, since it exploits a vulnerability in the Message Board functionality directly accessible from the System Menu.



How the LetterBomb message appears on the Message Board

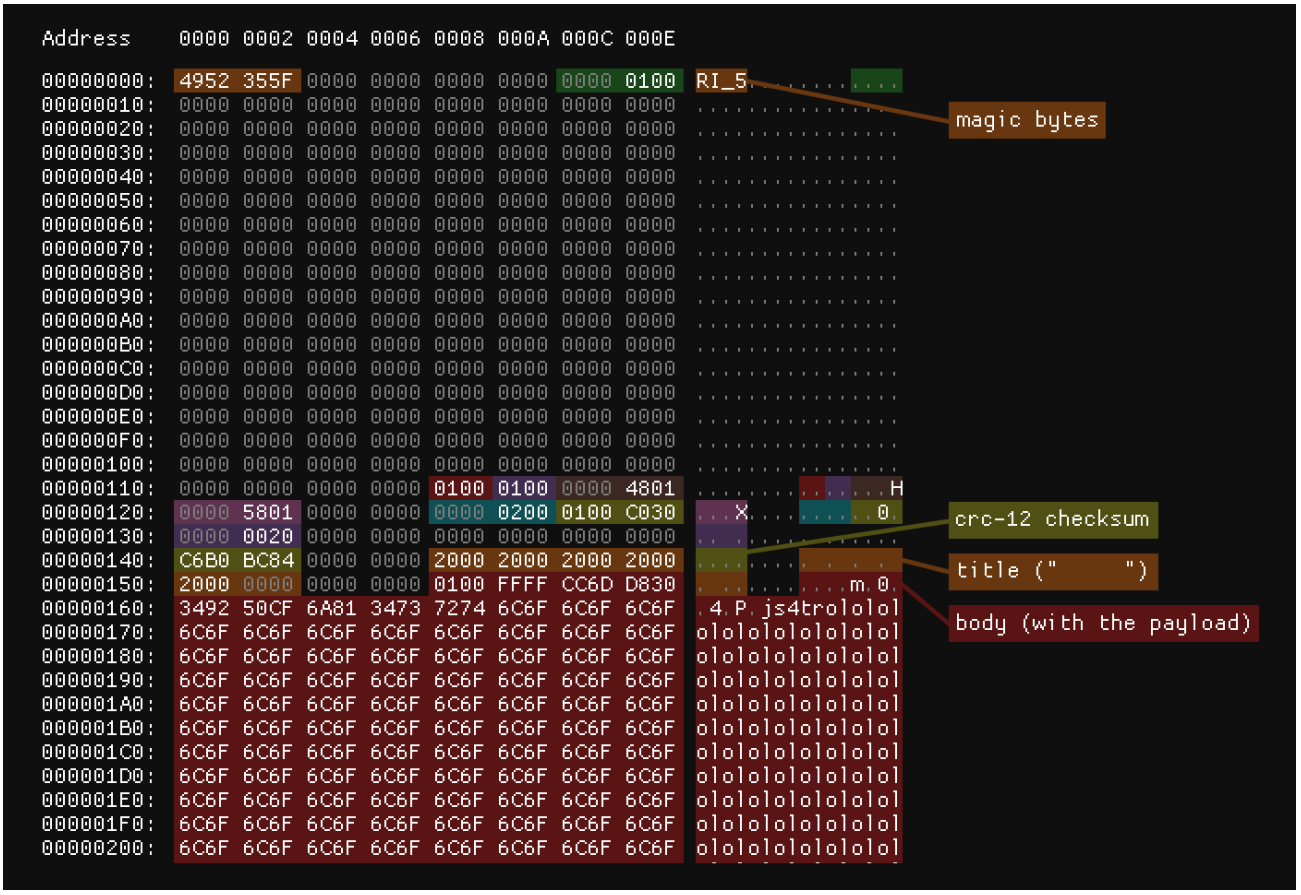
The Wii Message Board allows users to save short messages on a calendar. These messages can be stored on the Wii’s internal storage or on an SD card. They are saved in the binary format illustrated below.



A hex dump of the Letterbomb message

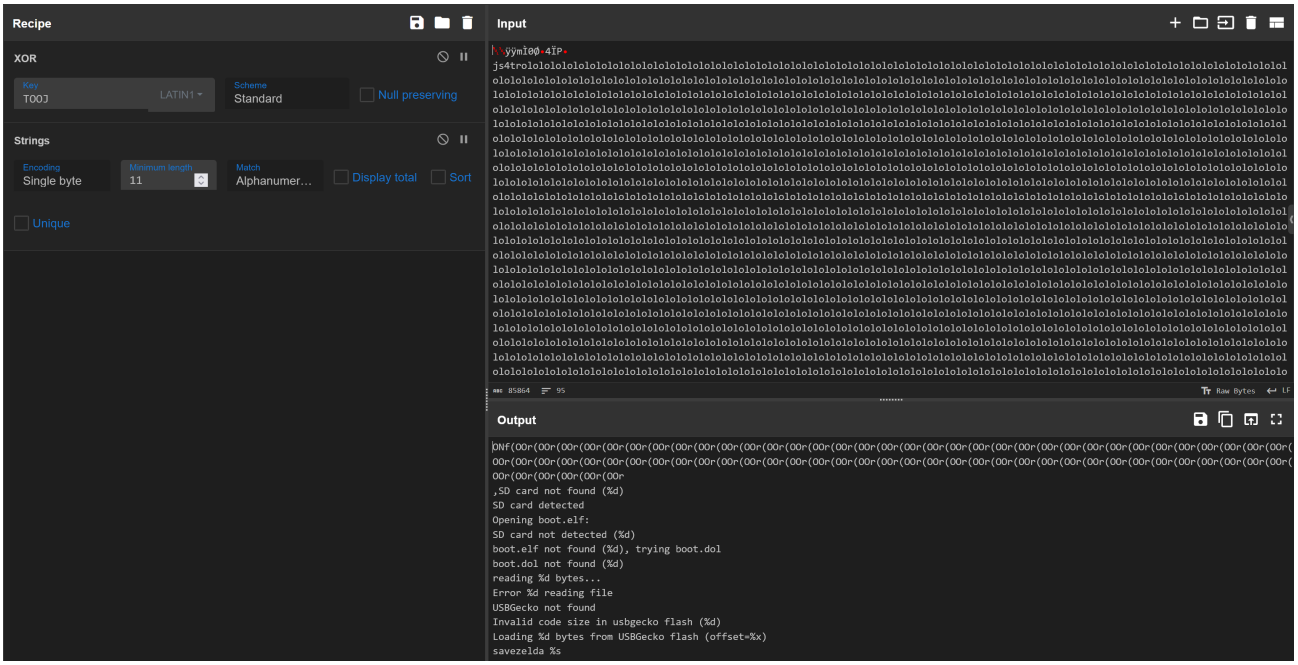
Inside, you can find a signature and encrypted content. This is a form of obfuscation that Nintendo uses to prevent users from easily altering these files. However, the encryption key is simply composed of 16 null bytes, and the Wii ID is used to sign the file. The Wii ID is simply a number derived from the MAC address and a constant `\x75\x79\x79`.

Within the encrypted content, there is the title and body of the message. It's within the message body that a Buffer Overflow vulnerability exists, exploitable if a long body is inserted into the file.



A hex dump of the decrypted part of the LetterBomb

The payload below, created by Fail0verflow, is slightly obfuscated but allows the loading and execution of a file (`boot.elf`) located on the SD card.



The strings from the deobfuscated payload

To this day, the LetterBomb remains one of the preferred vulnerabilities for installing the Homebrew Channel.

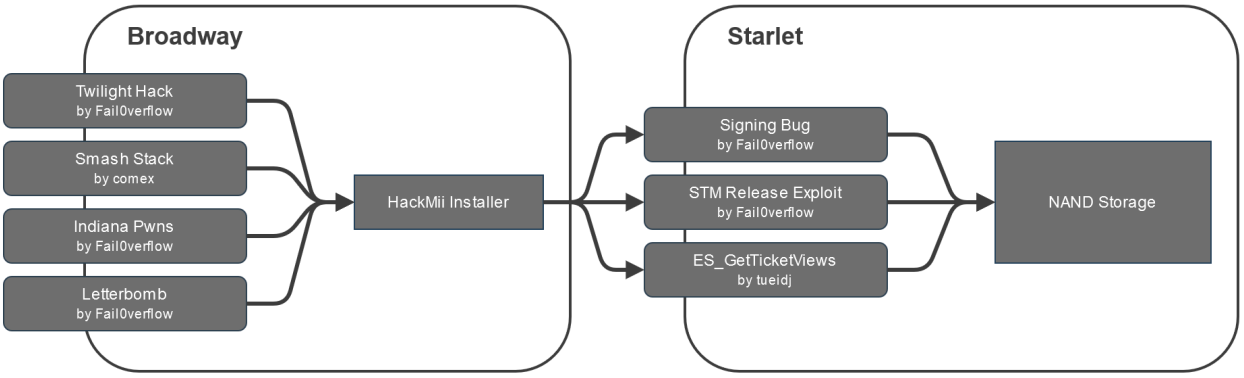
The Homebrew Channel

It is tedious to run an exploit every time you wish to launch a custom application. That's why Fail0verflow created the Homebrew Channel, an application that can be installed on the Wii and serve as launcher accessible directly from the System Menu.

The Homebrew Channel

The installation process consists in:

1. Exploiting a vulnerability in a title (e.g.: LetterBomb) to gain access to the Broadway processor
2. Launching the HackMii Installer from the SD card (`boot.elf`)
3. Exploiting a vulnerability in IOS to gain access to the Starlet CPU
4. Installing the Homebrew Channel



The various vulnerability chains to be exploited to install the Homebrew Channel

The early versions of HackMii Installer could simply utilize the Signing Bug to install the Homebrew Channel. However, after Nintendo patched it, Fail0verflow had to discover new security flaws in IOS.

The STM Release Exploit

For some time, the HackMii Installer utilized an exploit in the STM module of IOS to disable the signature verification, allowing the installation of the Homebrew Channel. The STM module is responsible for handling various hardware functions, such as the front LED and power buttons. Games could register a callback to received notification when for example the reset button was pressed.

The vulnerability lies in the mechanism to unregister from this callback (`STM_UnregisterStateEvent`), the IOS code is missing a return statement, meaning that if there is no callback registered the function would continue unintentionally and dereference a NULL structure. This unintended behaviour leads to writing a value of 0 to an address that can be controlled by the Broadway processor.

```
1  void STM_UnregisterStateEvent(ios_message *imm_msg)
2  {
3      struct ios_message *the_hook_msg = hook_msg;
4
5      printf_disabled("Release\n");
6      if (!the_hook_msg) {
7          AckMessage(imm_msg, -6);
8          // <------- Missing a return statement here
9      }
10     hook_msg = NULL;
11
12     // if the_hook_msg is null,
13     //    0 is written to an address from the pointer at 0x00 (null) + 0x18 (offset of buffer_out)
14     *(u32*)the_hook_msg->buffer_out = 0;
```

```
    AckMessage(the_hook_msg, 0);
    AckMessage(imm_msg, 0);
}
```

By changing the address to the return address in the stack, the exploit would trick `STM_UnregisterStateEvent` to return and execute code located at address 0, which is modifiable by the Broadway processor. Although the STM module had limited permission in IOS, it still possessed the capability to make syscalls. `get_kernel_flavor` is a syscall that can write specific values to any address with kernel permission.

```
void get_kernel_flavor(u32 *a, u16 *b)
{
    *a = 3;
    *b = 0;
}
```

With the ability to write the value 0 to any address, it becomes possible to disable signature verification and enable the installation of any software. The exploit was eventually patched by Nintendo, and as a result Fail0verflow released a detail explanation here: https://hackmii.com/2010/01/the-stm-release-exploit

It is challenging to determine precisely how HackMii Installer works today because its code has multiple layers of obfuscation to prevent Nintendo from discovering and fixing the vulnerabilities exploited. However, the latest release notes refer to a vulnerability discovered by tueidj, which appears to be related to the `ES_GetTicketViews` function.

The ES_GetTicketViews Exploit

There is a specific syscall in IOS `void syscall_54(int factory_mode)` that enable the `HW_AHBPROT` setting. When activated, this setting grants Broadway full access to the hardware components (Wireless, Storage, etc.) without the need to communicate with IOS. It's used at the factory to test the Wii's hardware, but in the case of the HackMii Installer, this setting would enable software to be installed directly to the storage without being validated by IOS.

The problem is that this syscall can only be invoked from a particular IOS module the E-Ticket Services (ES). This where the function `ES_GetTicketViews(u64 titleID, tikview *views, u32 cnt)` comes into play.

The purpose of the function is to write tickets into the memory address specified by the pointer, IOS ensures that this is a valid address range. The upper bound of the address range is determined by the following formula: `START_ADDRESS + TICKETVIEW_COUNT * TICKETVIEW_SIZE`

Each ticket view occupies 216 bytes of memory. By requesting 536 870 912 tickets, the upper bound (unsigned int32) overflows to exactly 0. As a result, IOS will not validate the address before writing the ticket.

To take advantage of this vulnerability, the `tikview` pointer can be set to a specific location on the ES stack. When the function returns, it will execute the code at the start of MEM1, a location that can be modified by the Broadway processor with instructions to invoke `syscall_54` giving it full control to install the Homebrew Channel.

Here's a simplified pseudocode implementation for the ES_GetTicketViews exploit:

```
payload[] =
{
    0xE3A00001, // mov r0, #1
    0xE6000A90, // call syscall_54(1)   (Enable HW_AHBPROT)
    0xE3A00065, // mov r0, #101         (return 101)
    // Additional instructions to return to the correct place...
};

// Backup the original content of MEM1
memcpy(backup, MEM1, sizeof(backup))

// Copy the payload to MEM1
memcpy(MEM1, payload, sizeof(payload))

// Try different location on the ES stack till a success
for (stack = 0x20111000; stack < 0x20112500; stack += 4)
{
    ES_GetTicketViews(title_id, (void*)stack, 536870912)

    if (ret == 101)
        break // Exploit successful
}

// Restore the original content of MEM1
memcpy(MEM1, backup, sizeof(backup))
```

Closing thoughts

The Nintendo Wii was designed with multiple security mechanisms to prevent the execution of unauthorized software, but it was weaknesses in the implementation that opened the door to hackers. Right now, it’s really easy to hack the Nintendo Wii, just copy some files to the SD card and click on the LetterBomb message and voilà. It’s all done automatically, but only thanks to the impressive work of Fail0verflow.

I highly recommend listening to their presentation from 2008 on the subject which inspired me to write about their journey.