# 17 Mistakes Microsoft made in the Xbox Security System

Michael Steil
http://www.xbox-linux.org/

29.12.2005

# Xbox Security

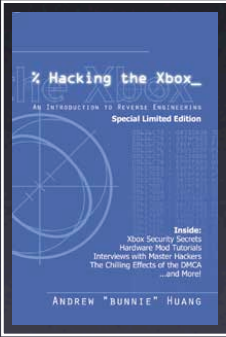
December 2002 (19C3)
*Andy Green, Franz Lehner, Milosch Meriac, Michael Steil*
**Xbox Hacking & Xbox Linux**

+

December 2003 (20C3)
*Andrew "bunnie" Huang*
**Xbox Hardware Hacking**
*Stefan Esser, Franz Lehner, David Jilli,*
*Franz Lehner, Melissa Mears, Michael Steil*
**Xbox Software Hacking**

+ August 2005
additional research...

+




*Andrew "bunnie" Huang*
**Hacking the Xbox**

=

October 2005

## 17 Mistakes Microsoft Made in the Xbox Security System

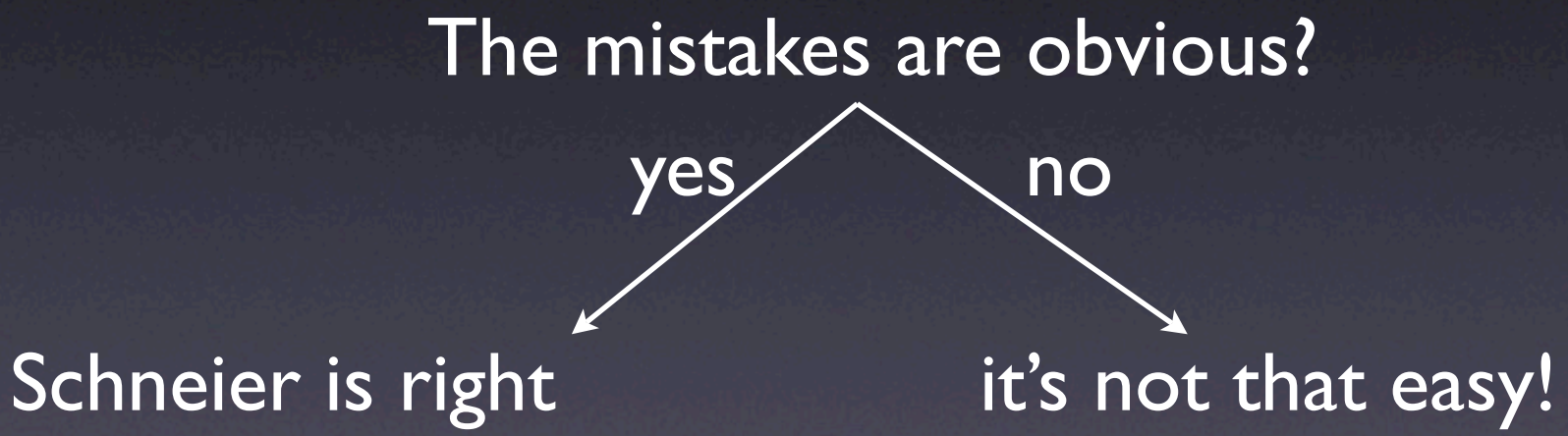
Michael Steil <mist@c64.org>
Xbox Linux Project http://www.xbox-linux.org/

*conclusion*
*summary*


Schneier on Security: X-Box Security
http://www.schneier.com/blog/archives/2005/0
RSS
Google
Schneier on Security: X-Box...
Bruce Schneier
Schneier on Security
A weblog covering security and security technology.
Home
Essays and Op Eds
Computer Security Articles
In the News
Speaking Schedule
Password Safe
Lots of kindergarten security mistakes.
The MD5 Defense »
Interesting article: "The Hidden Boot Code of the Xbox, or How to fit three bugs in 512 bytes of security code."
Microsoft wanted to lock out both pirated games and unofficial games, so they built a chain of trust on the X-Box from the hardware to the execution of the game code. Only code authorized by Microsoft could run on the X-box. The link between hardware and software in this chain of trust is the hidden "MCPX" boot ROM. The article discusses that ROM.
Lots of kindergarten security mistakes.
Posted on August 10, 2005 at 01:00 PM
Weblog Menu
Recent Entries
Computer Crime Hype

# Idea of this Talk

Microsoft's view




Lots of kindergarten security mistakes.

# You are Microsoft!

# Home Entertainment

| SONY | Microsoft® |
| --- | --- |
| Discman, Mini-Disc | Windows Media Audio |
| DVD Recorder | Windows Media |
| Playstation |  |

# A video game console

Seamus Blackley

We need a video game console in less than 2 years!


- PC hardware
- Windows 2000
- DirectX libs
- smaller case

fast and cheap!

# What is the Xbox?

- Celeron III 733, 64 MB RAM
- nVidia GeForce 3½, TV out
- 10 GB HD, DVD-ROM
- Fast Ethernet, USB
- stripped-down Windows 2000 system



The Xbox is a PC!

# What makes a PC?

- x86 CPU?
- VGA-style GPU?
- PCI, AGP, IDE, USB?
- PIC (1982 Interrupt Controller)
- PIT (1982 Timer)
- A20 gate (1984 hack by IBM)

# Security

- Trivial to run Linux?
- No, security system!

| ***Threat*** | ***Effect*** | ***Reason*** |
|---|---|---|
| **Linux** | Xbox as a computer | Xbox sold at loss |
| **Homebrew** | media player, browser | software monopoly |
| **Copied Games** | piracy | obvious |
| **Unlicensed Games** | anyone can make games | missing royalties |

address 4 threats with 1 security system

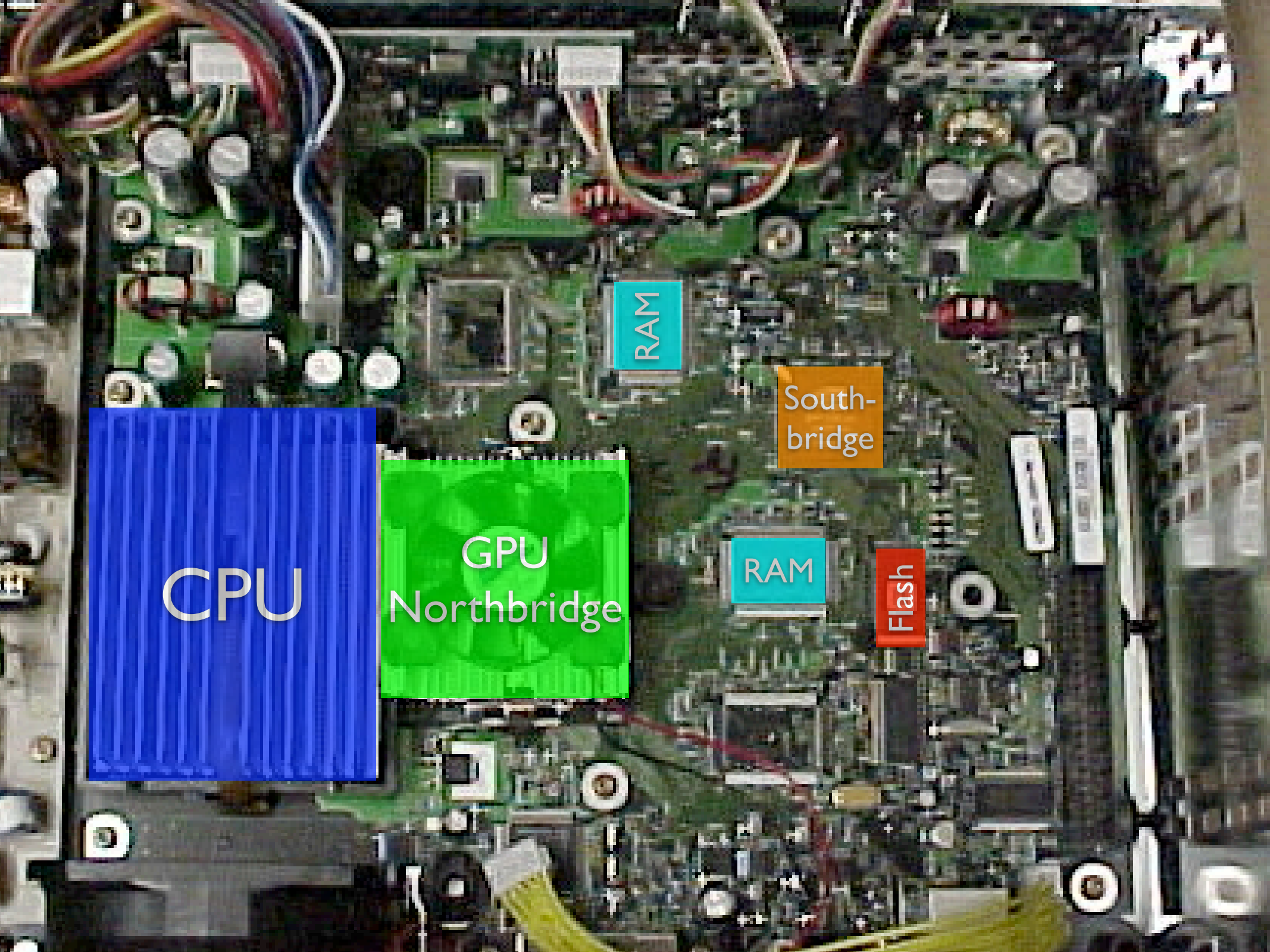
# Only run authentic code

only pass execution to trusted code

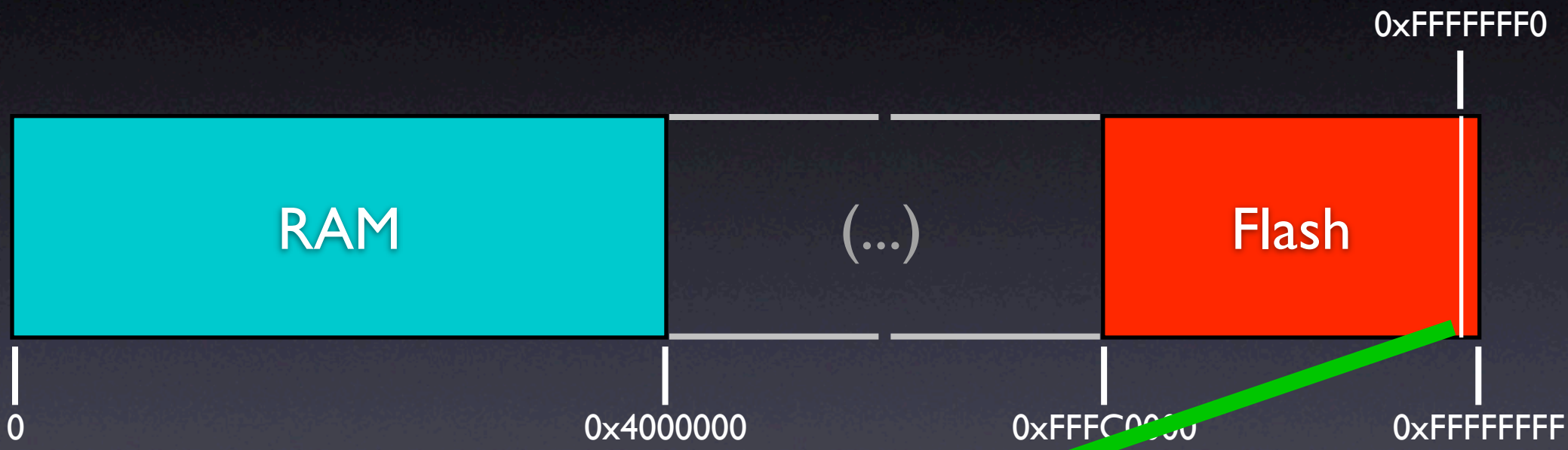
"Chain of Trust"


"Trusted Computing"
by Benjamin Stephan and Lutz Vogel
http://www.lafkon.net/tc/

# Chain of Trust

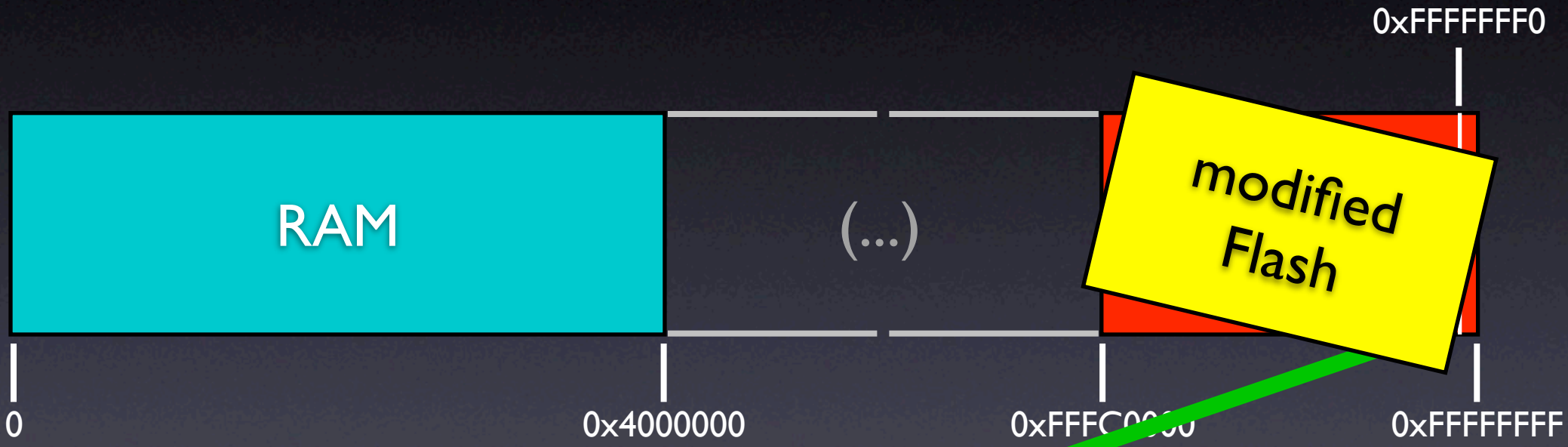

RAM
South-
bridge
CPU
GPU
Northbridge
RAM
Flash

# x86 Startup



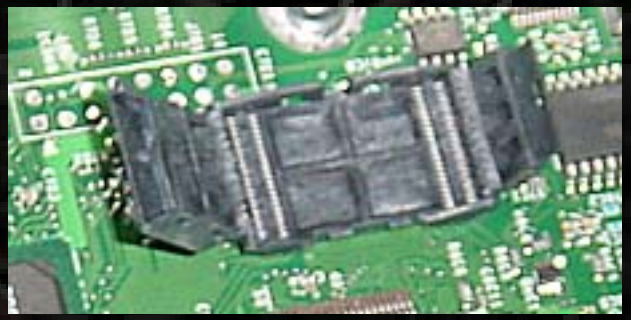

```
FFF0   lgdt offset gdt
FFF4   mov cr0, eax
FFF6   or eax, 1
FFF8   mov eax, cr0
FFFA   jmp 0xFFFC0000
```

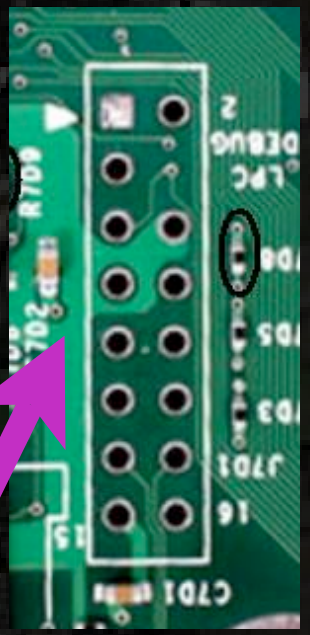
# x86 Startup




```
FFF0   lgdt offset gdt
FFF4   mov cr0, eax
FFF6   or eax, 1
FFF8   mov eax, cr0
FFFA   jmp 0xFFFC0000
```

replace the Flash chip
override the Flash chip
overwrite the Flash chip





RAM

South-
bridge

CPU

GPU
Northbridge

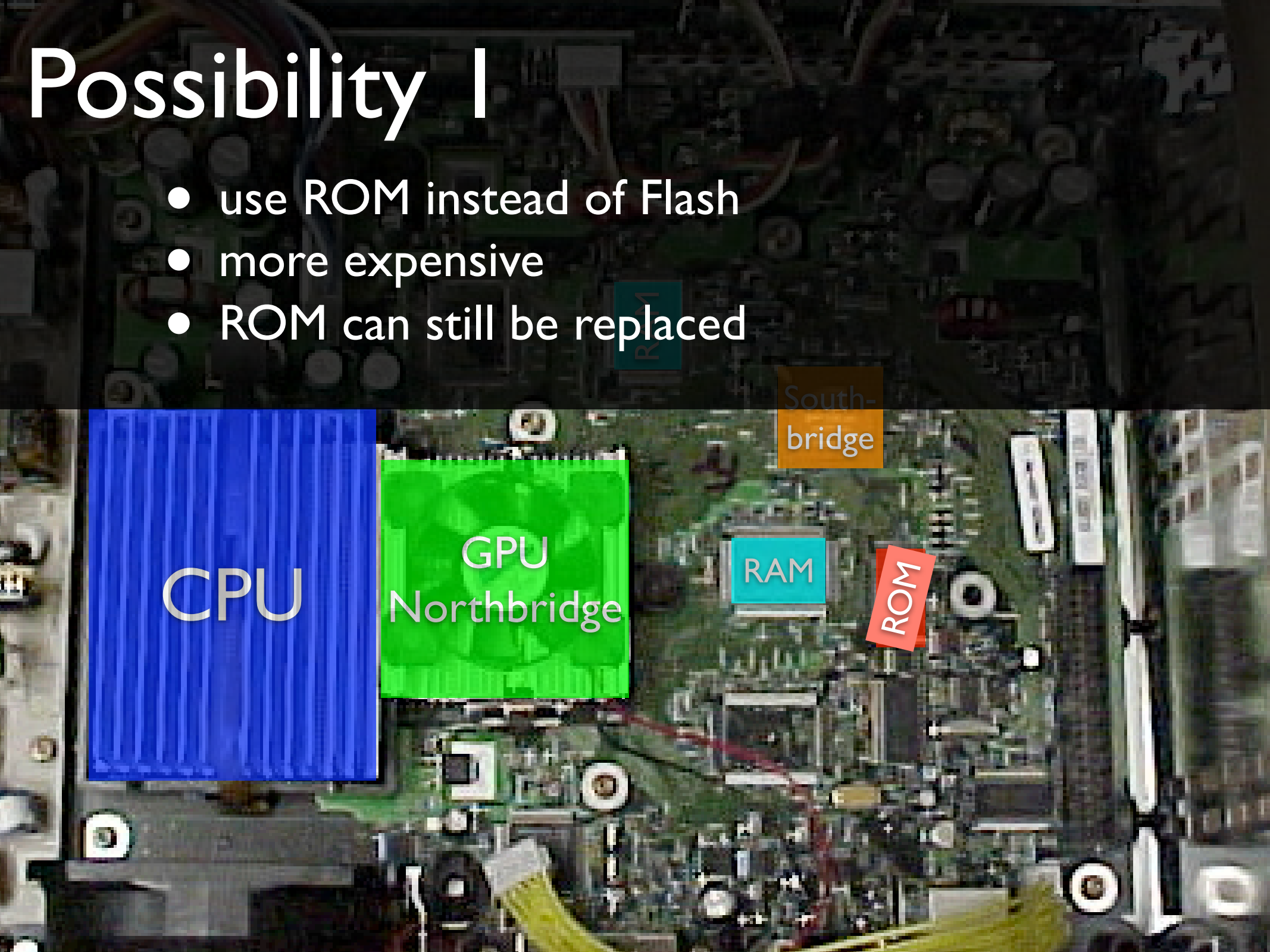
RAM

Flash

we must **not** start from Flash!

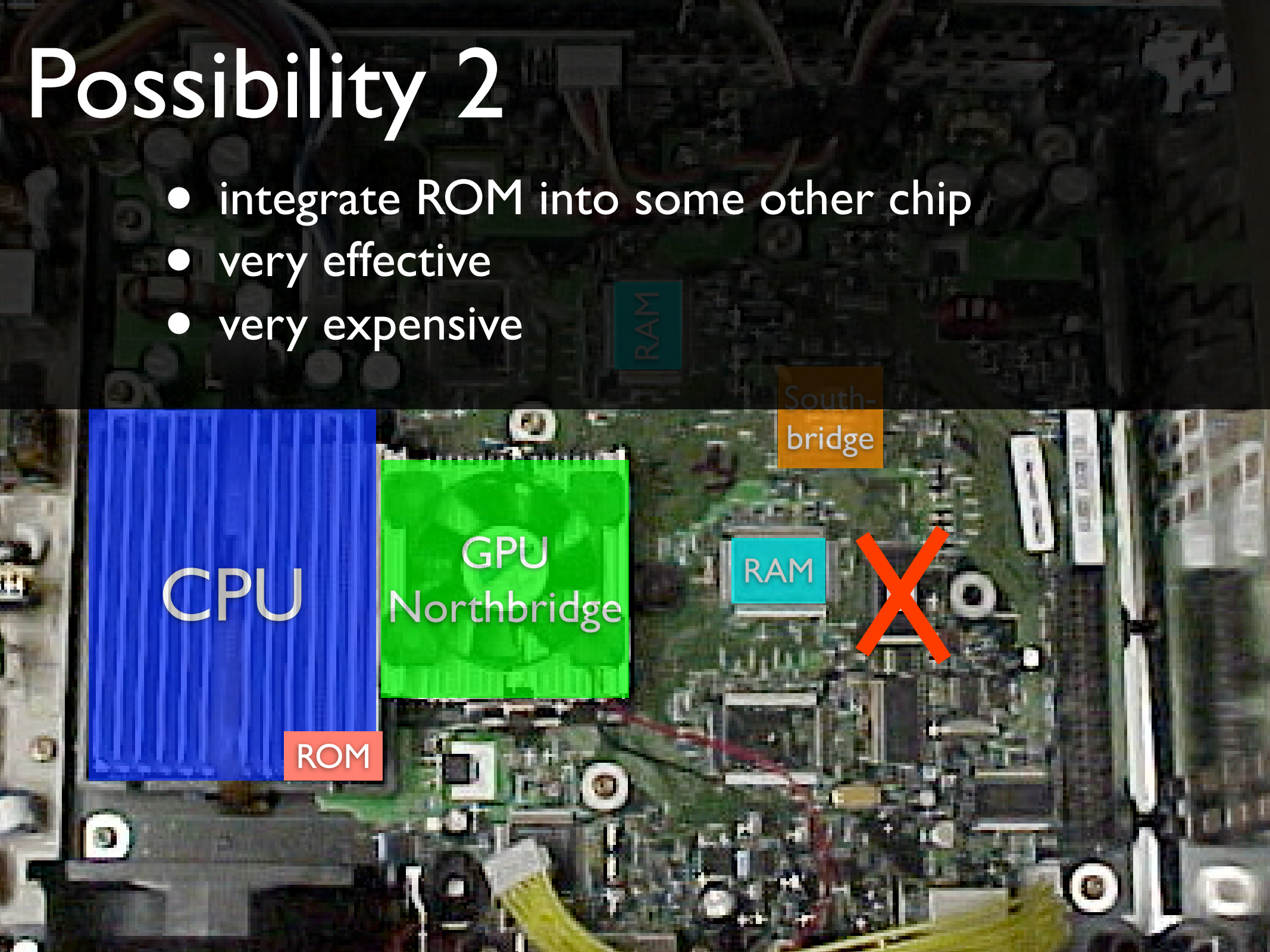
# Possibility 1

- use ROM instead of Flash
- more expensive
- ROM can still be replaced



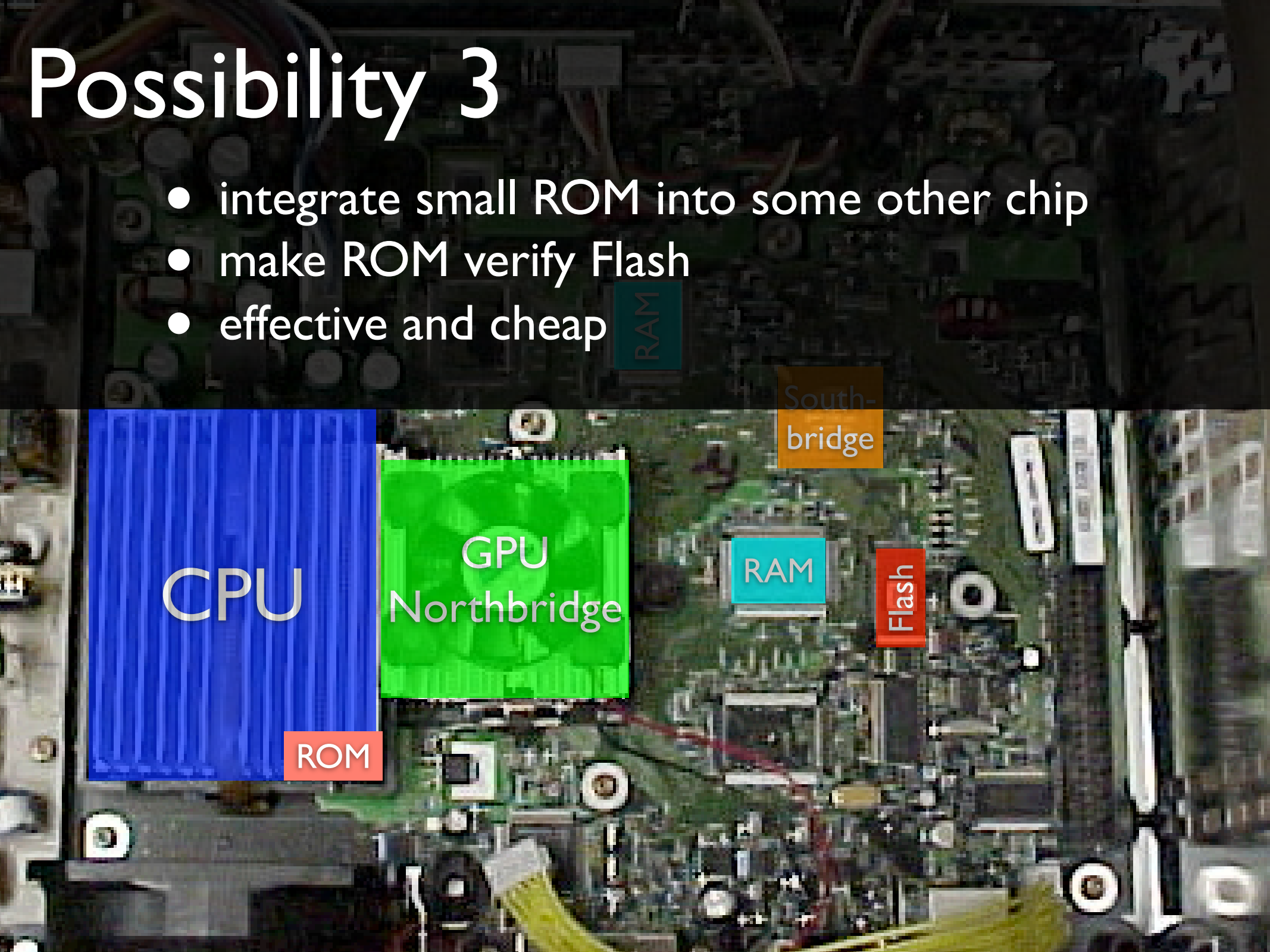
# Possibility 2

- integrate ROM into some other chip
- very effective
- very expensive

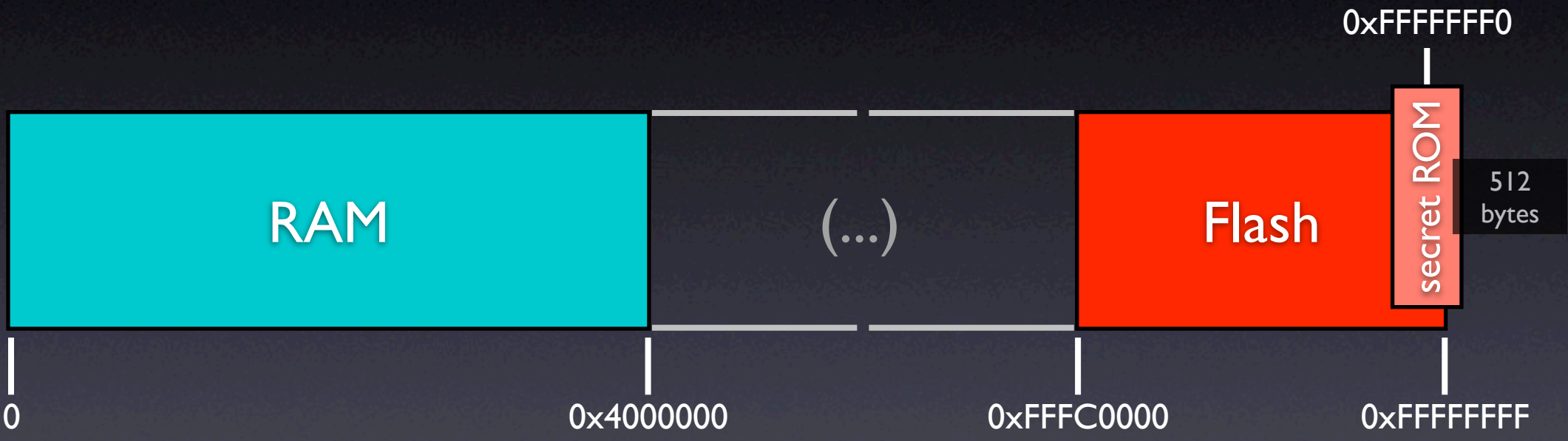


# Possibility 3

- integrate small ROM into some other chip
- make ROM verify Flash
- effective and cheap



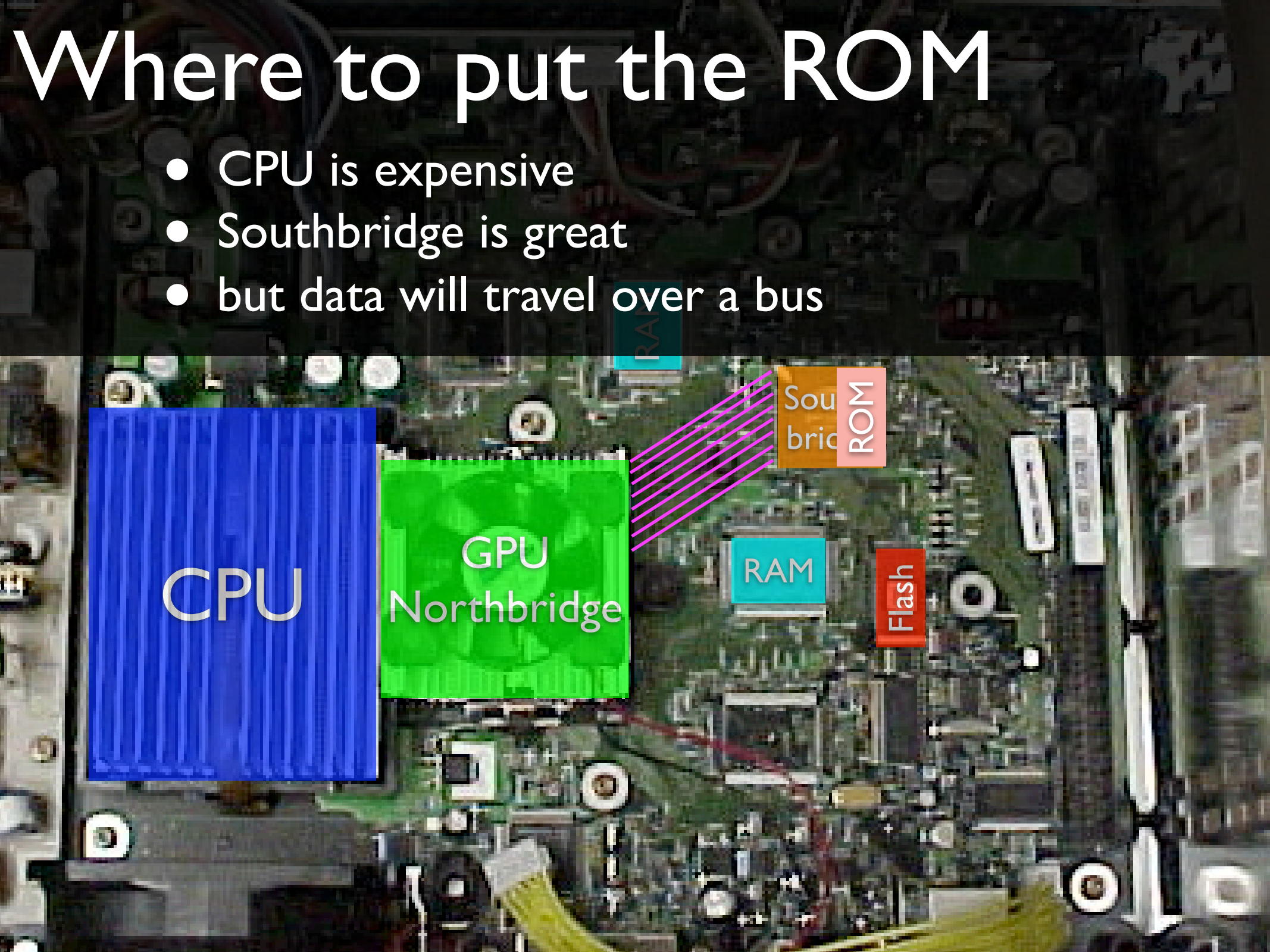
# Secret ROM




- verify Flash integrity
- if okay, pass control to code in Flash

# Where to put the ROM

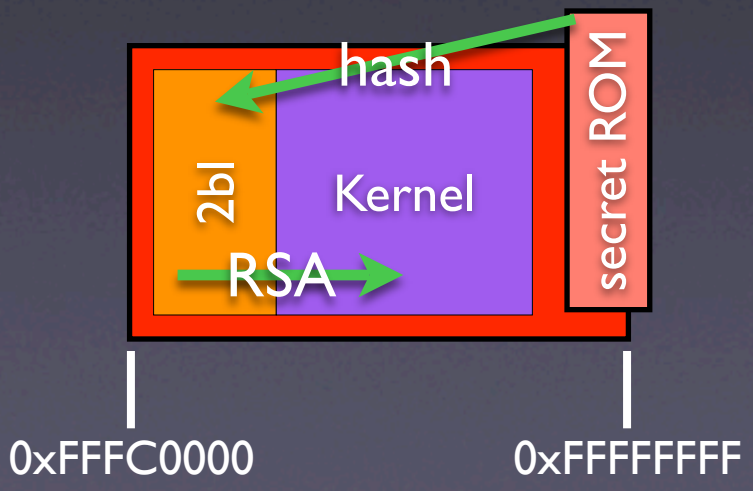
- CPU is expensive
- Southbridge is great
- but data will travel over a bus



# Flash Verification

- Check SHA-1/RSA signature
  - great, but won't fit in 512 bytes
- Check SHA-1 hash
  - we cannot update the Flash

Solution:



# Secret ROM

- initialize RAM (stress test)
- decrypt 2bl
- verify 2bl integrity (hash)
- pass control to 2bl

512 bytes???

# Virtual Machine

- read/write memory
- ports
- PCI config
- and/or
- if-then
- end

"Xcodes"

```
struct {
    char opcode;
    int op1;
    int op2;
} *p;
int acc;

p = 0xFFF00080;

while(1) {
    switch(p->opcode) {
        case 2:
            acc = *((int*)p->op1);
            break;
        case 3:
            *((int*)p->op1) = p->op2;
            break;
        case 4:
            outl(p->op1, 0x0CF8);
            outl(p->op2, 0x0CFC);
            break;
        case 5:
            ...
        case 0xEE:
            goto end;
    }
    p++;
}
end:
```

# Memory Initialization

```
POKEPCI MEM_CNTRL, 200
POKE 0, 0xAAAAAAAA
ACC = PEEK(0)
IF ACC = 0xAAAAAAAA GOTO END
POKEPCI MEM_CNTRL, 195
POKE 0, 0xAAAAAAAA
ACC = PEEK(0)
IF ACC = 0xAAAAAAAA GOTO END
POKEPCI MEM_CNTRL, 190
...
```

# VM Threat

- secret ROM may be revealed
- people may know how to write Xcodes
- Xcodes are not verified for integrity
- people may hack the Xcode interpreter

# Attack 1
# Dump Secret ROM

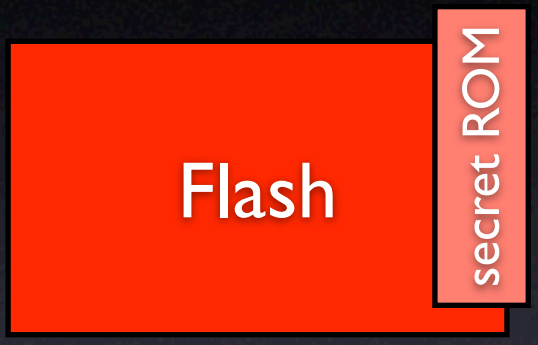
```
A = PEEK(0xFFFFFE00)
OUT 0xC000, A
A = PEEK(0xFFFFFE04)
OUT 0xC000, A
```

- make sure the Xcodes cannot access the secret ROM
- mask high addresses

```
and ebx, 0FFFFFFFh  ; clear upper 4 bits
mov edi, [ebx]      ; read from memory
jmp next_instruction
```

# Attack 2
# Turn off the Secret ROM

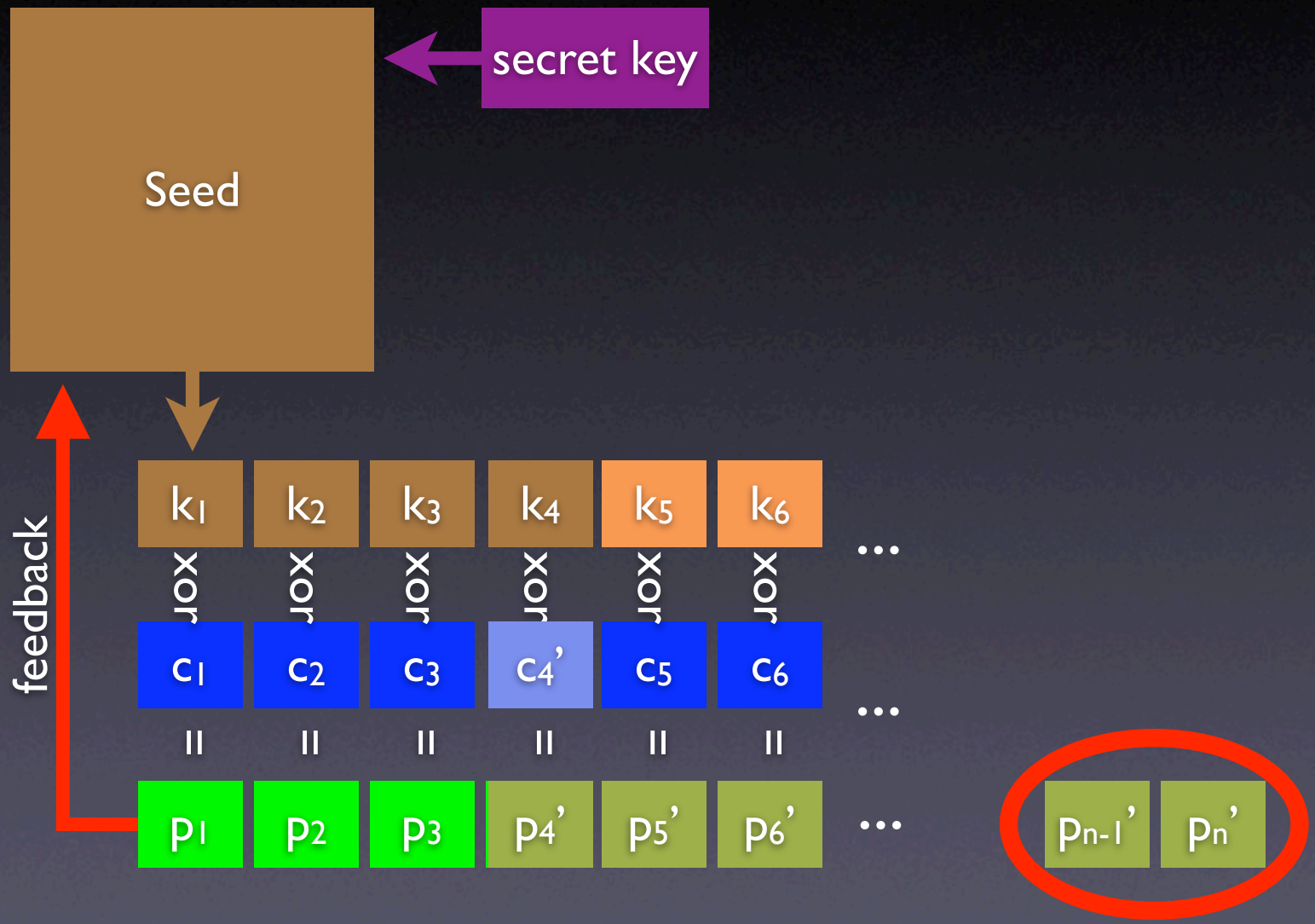
```
POKEPCI(80000880h, 2)
```




- This code will turn of the secret ROM
- We will fall down to Flash
- Check for this code!

```
   cmp ebx, 80000880h          ; MCPX disable?
   jnz short not_mcpx_disable ; no
   and ecx, not 2              ; clear  bit 1
not_mcpx_disable:
```

# Decryption and Hashing




check last few bytes of decrypted data - we get a hash for free

# Panic Code

- what if the Flash check failed?
- panic!
  - blink LED
  - halt CPU
  - disable secret ROM

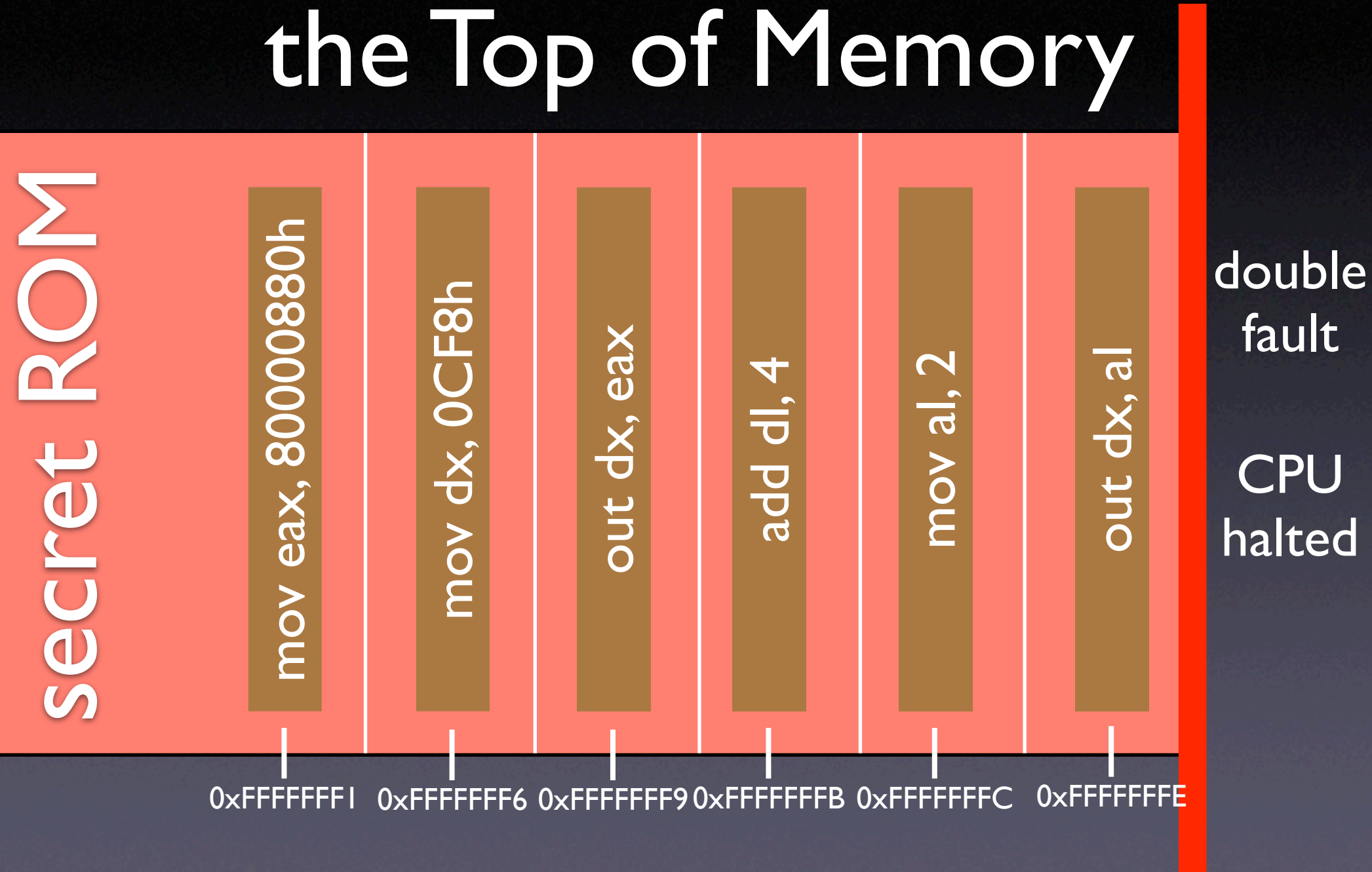
someone could attach special hardware
to dump the secret ROM after a panic

# How to Panic

or: how to crash a car when the motor is off

- Turn off CPU, then secret ROM
  - who can turn off the ROM??
- Turn off secret ROM, then the CPU
  - where is the code to halt the CPU??

# Drive against the Top of Memory



# Summary

- secret ROM inside Southbridge
- cannot be replaced/overridden/overwritten
- very hard to dump
- initializes RAM using secure Xcodes
- decrypts and hashes “2bl” in Flash
- panics if something goes wrong

# You are a hacker!

# Extracting the Secret ROM

- Andrew “bunnie” Huang
  - dumped the Flash ROM
  - put it online
  - got a call from Microsoft’s lawyers
  - removed it from his website

# Analysis

- bunnie
  - looked at where an x86 would start
  - what did he see?
  - zeros?

# Ouch!

- the upper 512 bytes of Flash contain:
- a virtual machine
- RC5 decryption/hash code
- panic code

when changing this code in Flash, nothing happened

it turned out this was an old version of the secret ROM

this code cannot successfully decrypt/verify “2bl”

# The Secret ROM

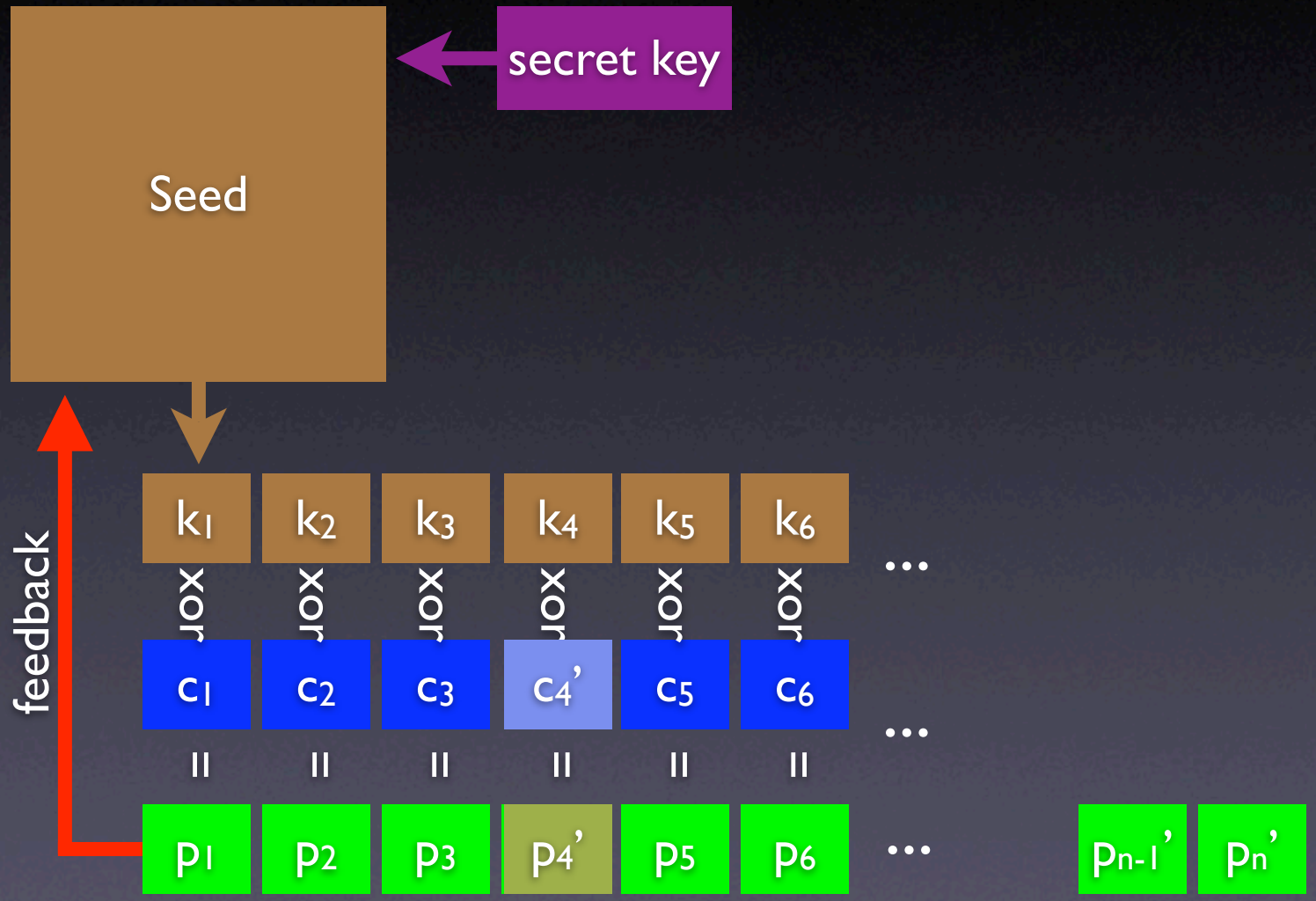
- bunnie
  - found out the CPU didn't boot off Flash
  - figured there had to be a secret ROM
  - sniffed HyperTransport
  - had the 512 bytes of secret code

virtual machine, RC4 decryption, panic code

but 2bl is hashed - what can we do?

# Decryption and Hashing

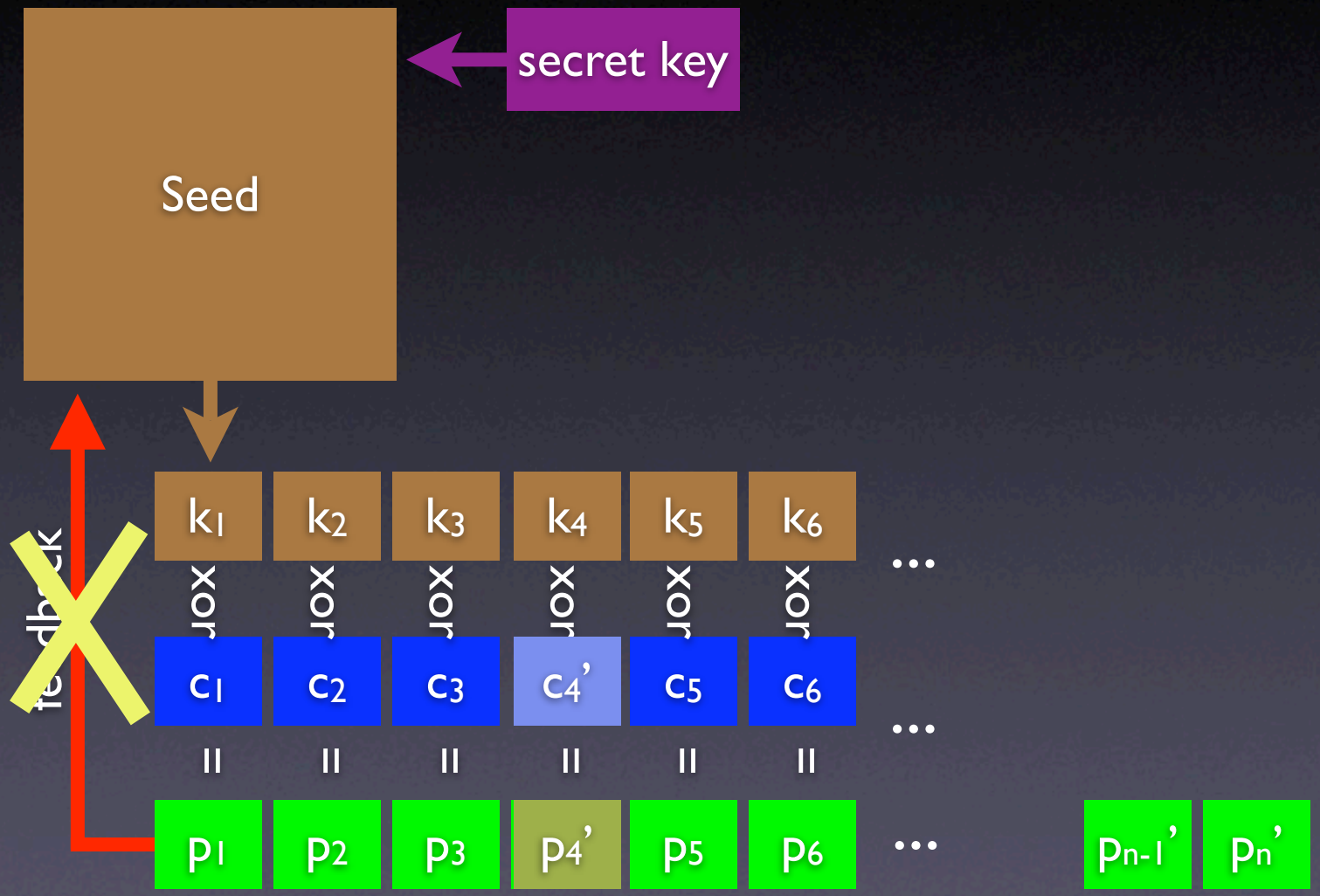
RC5




RC4 can **not** be used as a hash!

# Decryption and Hashing

RC4




RC4 can **not** be used as a hash!

# The Missing Hash

- The RC4 key is in the secret ROM
- So we have the secret RC4 key...
- There is no effective hash
- We can put our (encrypted) code where “2bl” should be

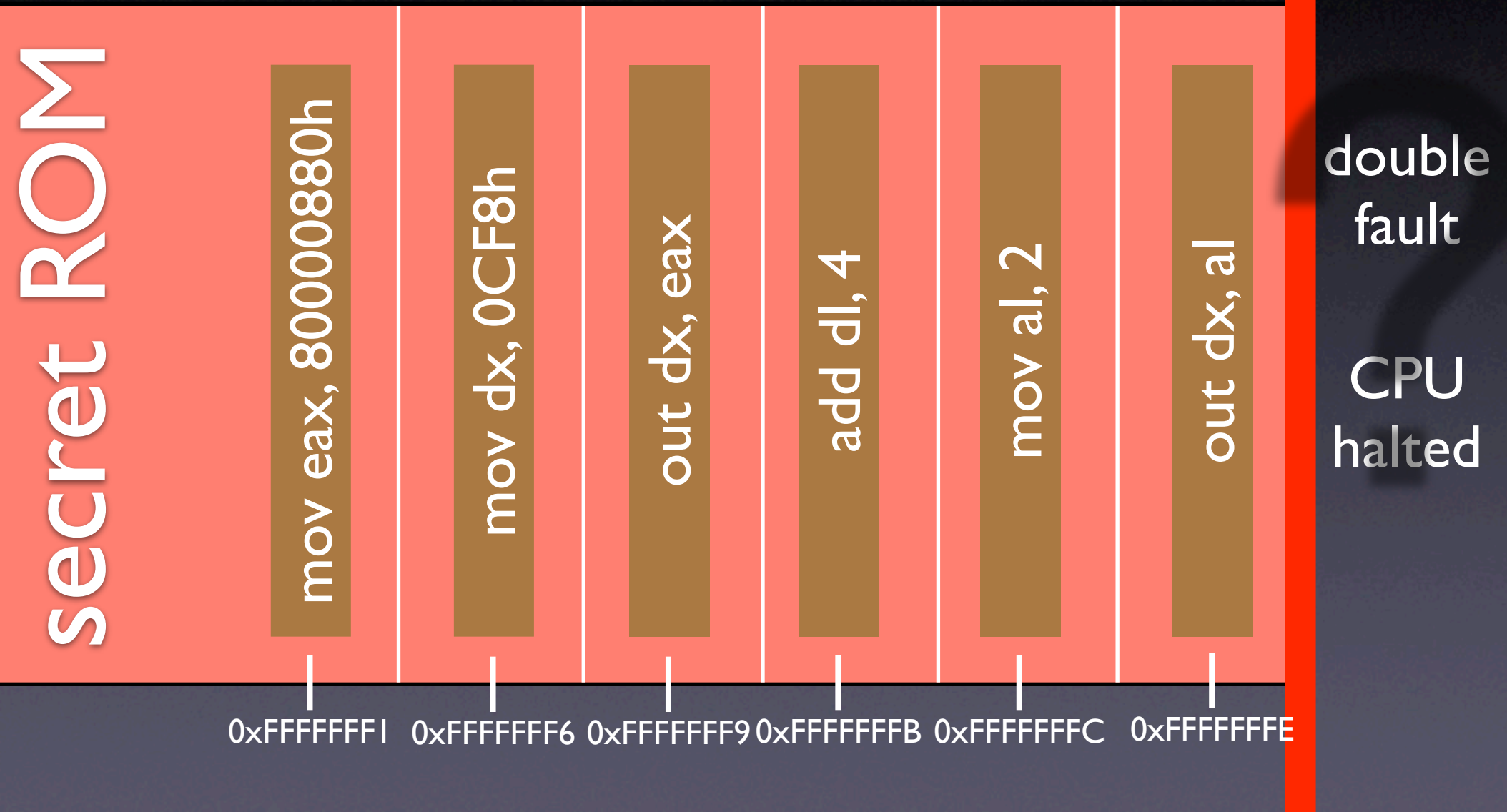
# Modchips

- First generation
  - disable onboard ROM
  - add 31 wire parallel ROM
- Second generation
  - pretend the onboard ROM is empty (ground data line D0)
  - Xbox will boot off external LPC ROM (9 wires!)

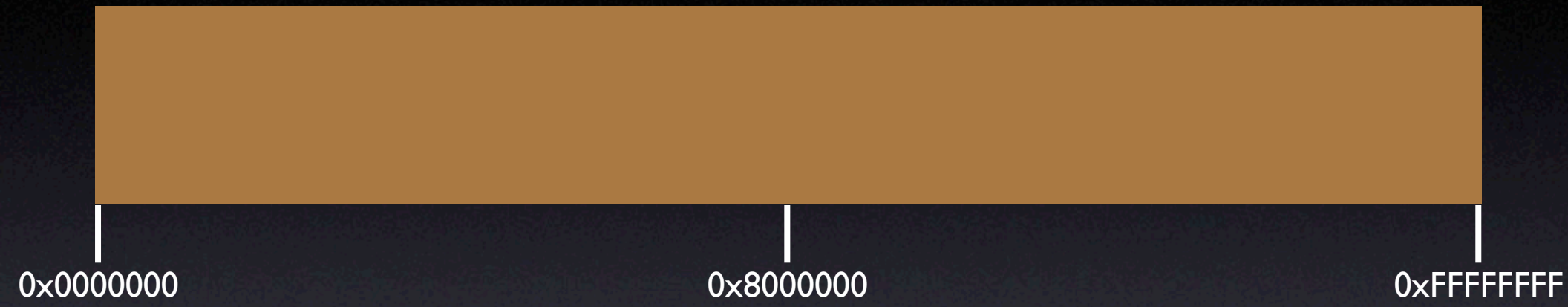
# Xbox Linux Bootloader

- ship the RC4 key with the build tools??
- we must find a better way
- let's look at the secret ROM code...

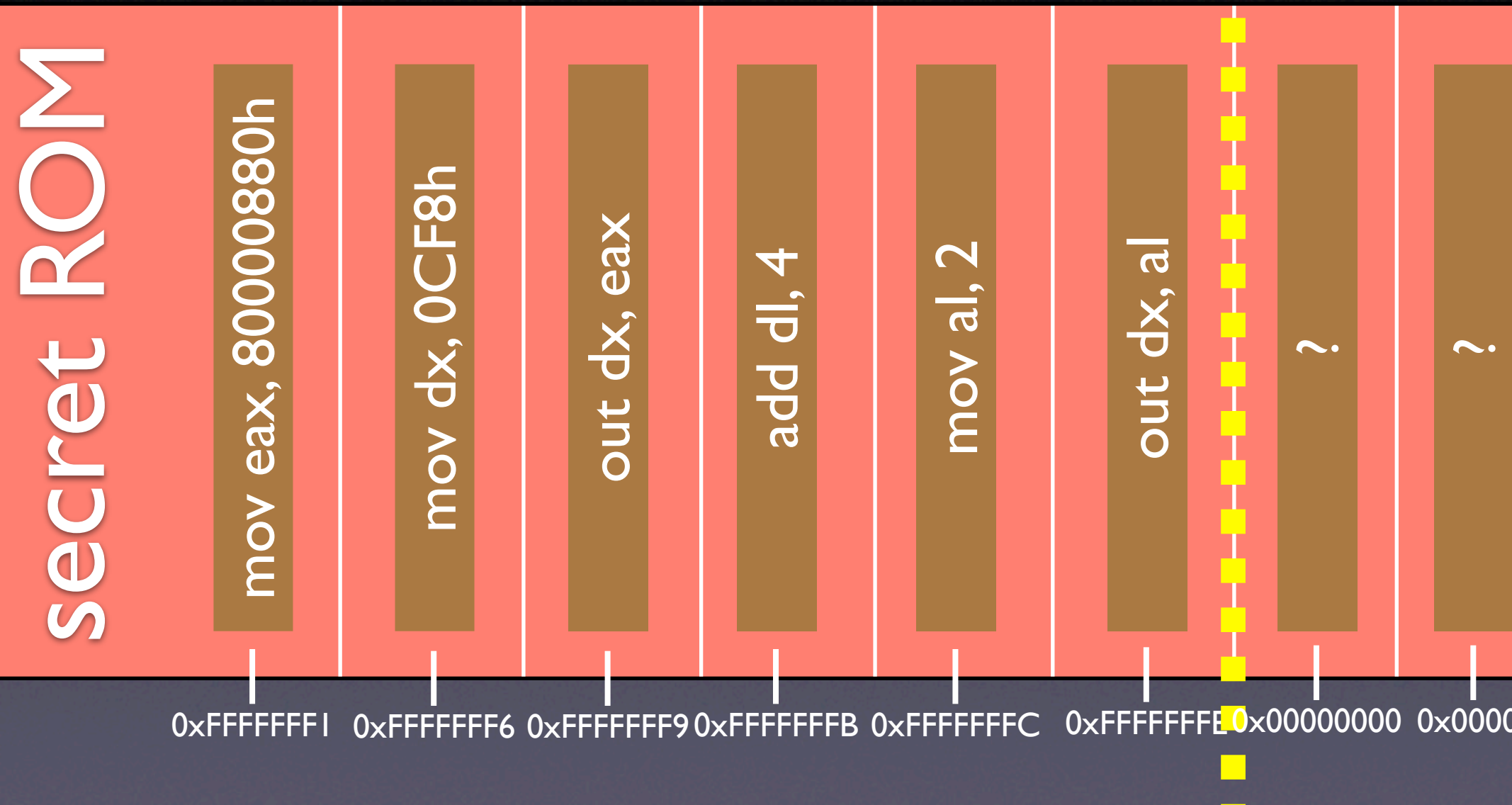
# Panic Code Revisited



# The Earth is a Sphere!




# Memory is a Donut



# Visor Bug

- “visor” thought:
  - perhaps execution rolls over to 0x00000000
  - put code at 0x00000000
  - let “2bl” check fail

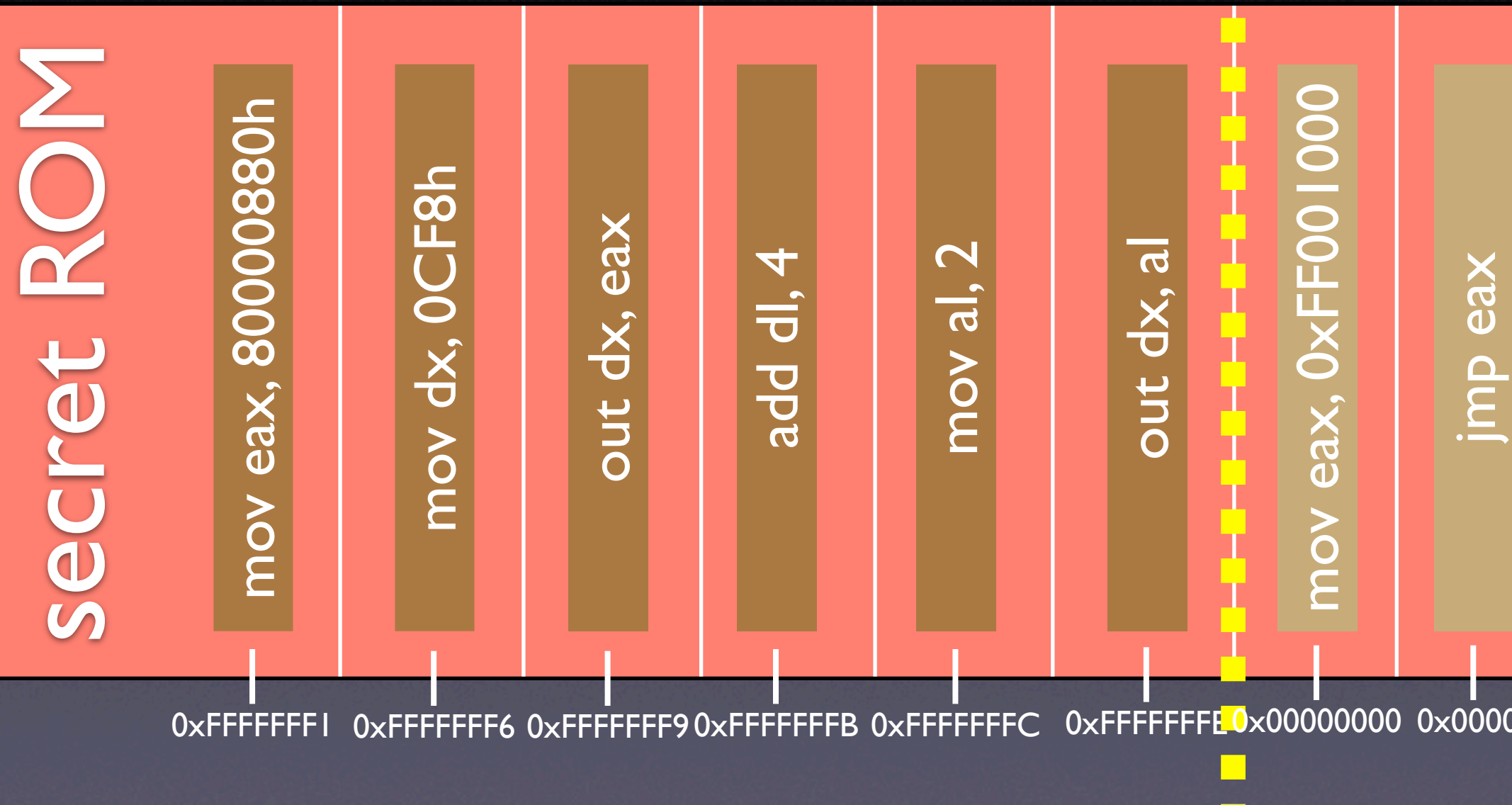
how do we put code at 0x00000000 (RAM)?

# Xcodes!

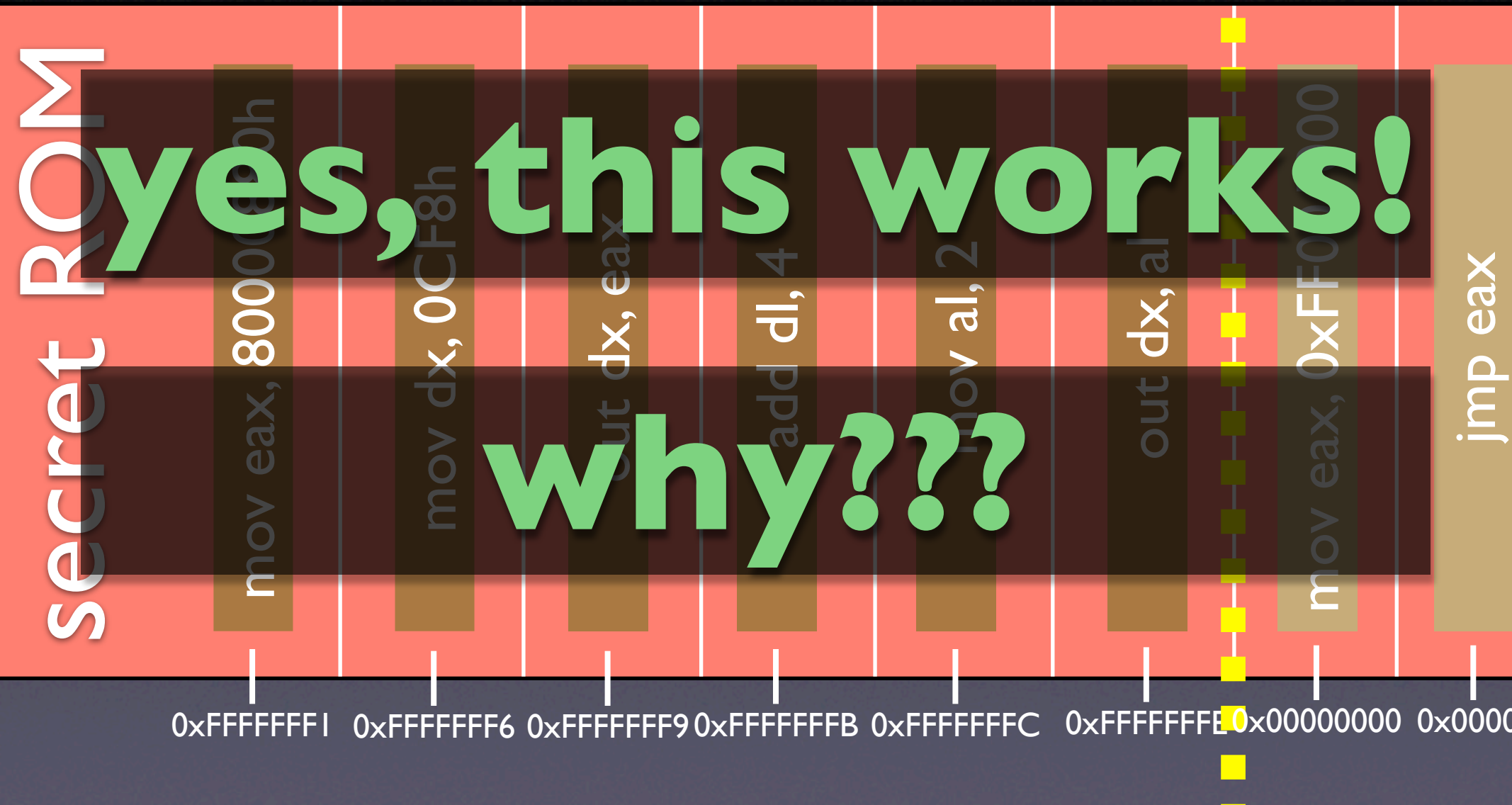
```
POKE 0x00000000, 0x001000B8
POKE 0x00000004, 0x90E0FFFF
```

```
00000000 mov eax, 0xFF001000
00000004 jmp eax
```

# Wrapping Around...



# Wrapping Around...



# A History Lesson

- The 8086 starts off at 0xFFFF0
- Many other CPUs start off at 0
- The 8086 could be used in a computer with ROM at 0

# A History Lesson

- Unconnected addresses will read back FF
- FF-FF-... behaves like NOP
- the 8086 can "NOP" its way through 0xFFFF0-0xFFFFF
- ...and continue execution at 0 - by design
- The Pentium III still does it like this

but this does not explain why Microsoft did it wrong...

# Microsoft's Mistake

- AMD CPUs don't have this behaviour
- They are specified to halt on a FFFFFFFF/00000000 transition
- All Xbox prototypes had AMD CPUs
- They switched to Intel

  "in the last minute"...

# “mist” Hack

```
POKEPCI(80000880h, 2)
```

```
   cmp ebx, 80000880h           ; MCPX disable?
   jnz short not_mcpx_disable   ; no
   and ecx, not 2               ; clear  bit 1
not_mcpx_disable:
```

Flash

secret ROM

| Bits | Description |
|---|---|
| 0-7 | reg |
| 8-10 | func |
| 11-15 | device |
| 16-23 | bus |
| 24-30 | reserved |
| 31 | must be 1 |

# “mist” Hack

```
POKEPCI(FF000880h, 2)
```

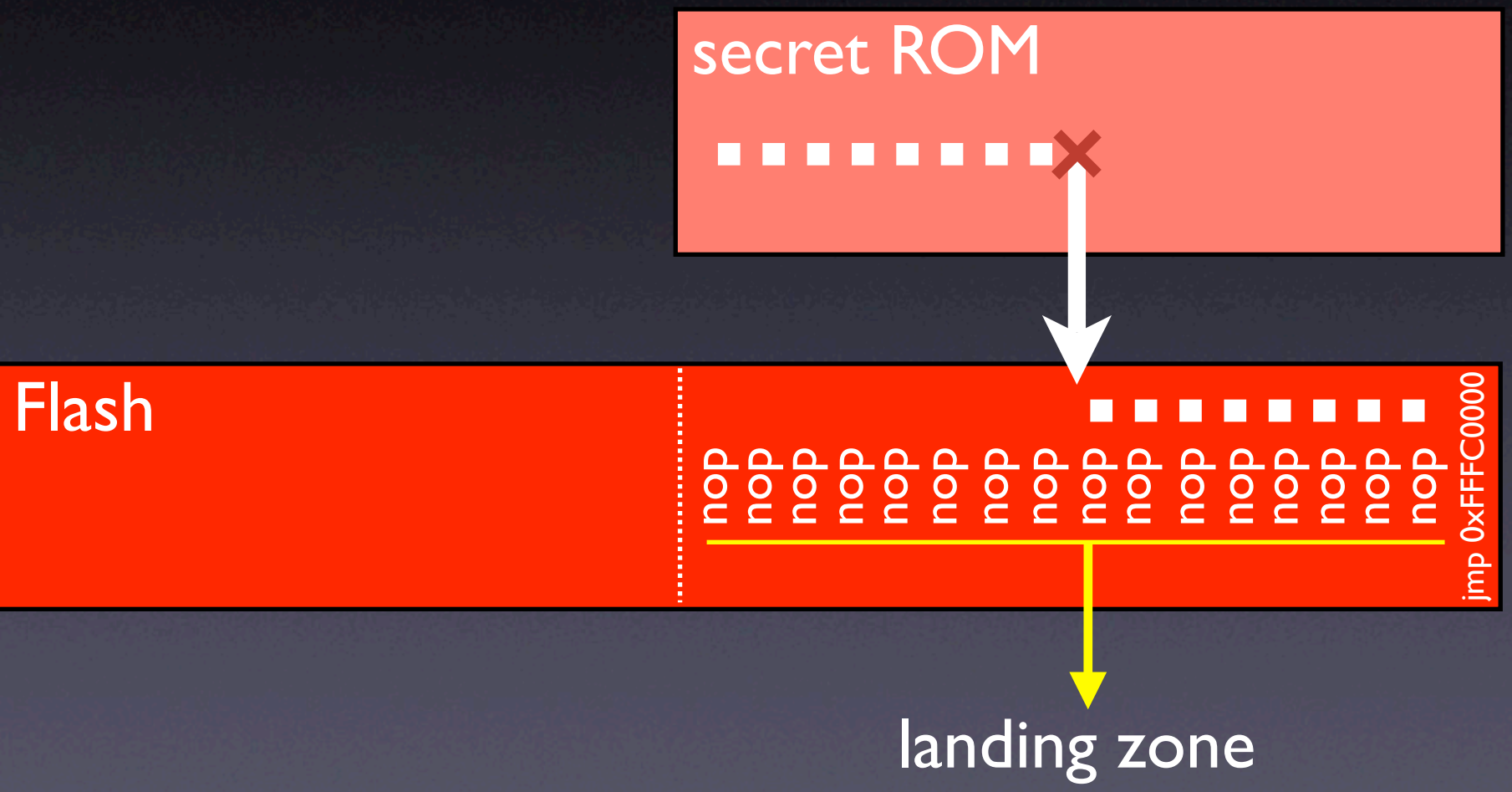
```
   cmp ebx, 80000880h           ; MCPX disable?
   jnz short not_mcpx_disable   ; no
   and ecx, not 2               ; clear  bit 1
not_mcpx_disable:
```

Flash

secret ROM

| Bits | Description |
|---|---|
| 0-7 | reg |
| 8-10 | func |
| 11-15 | device |
| 16-23 | bus |
| 24-30 | reserved |
| 31 | must be 1 |

# Landing

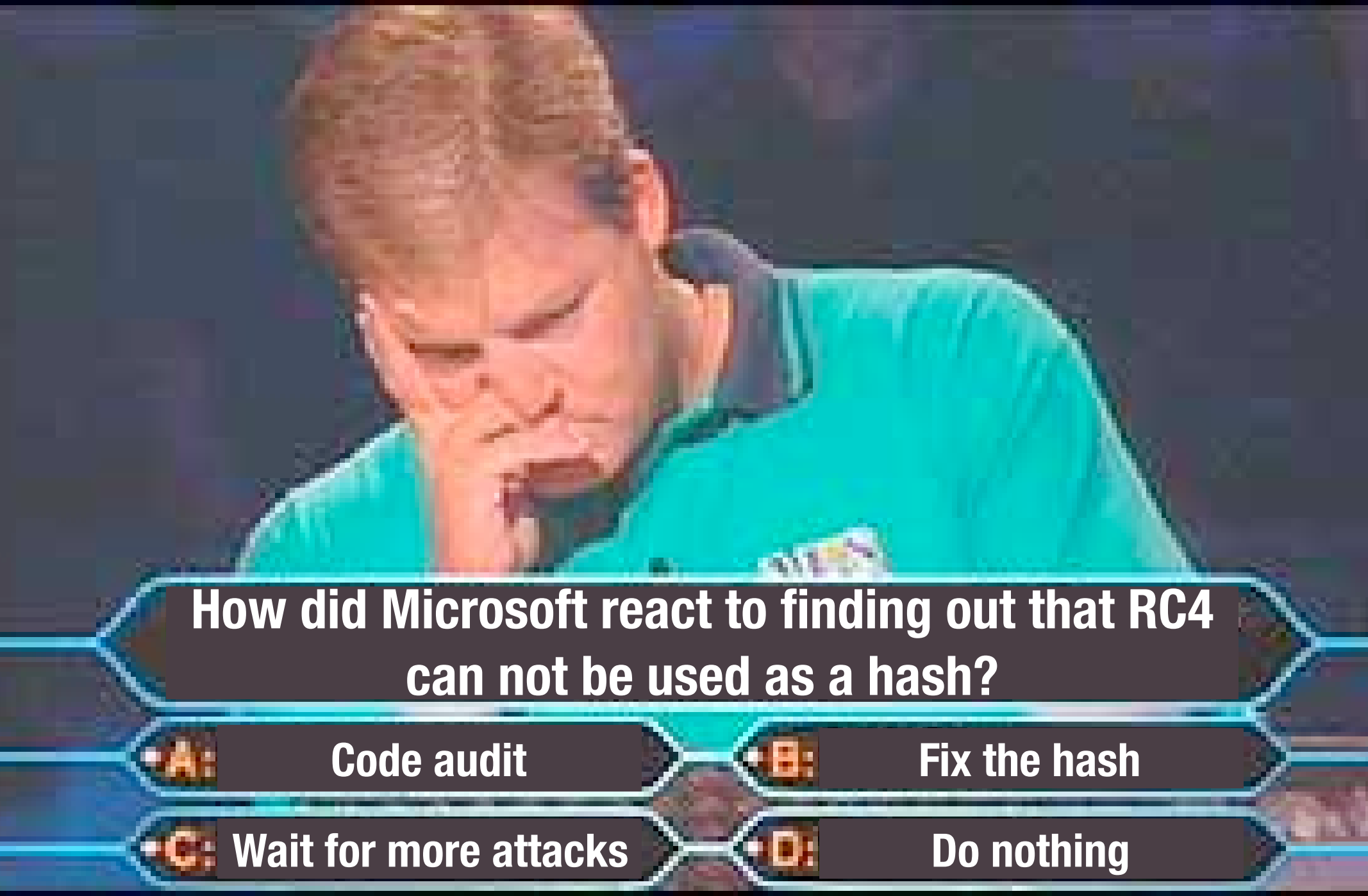

secret ROM
Flash
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
jmp 0xFFC0000
landing zone

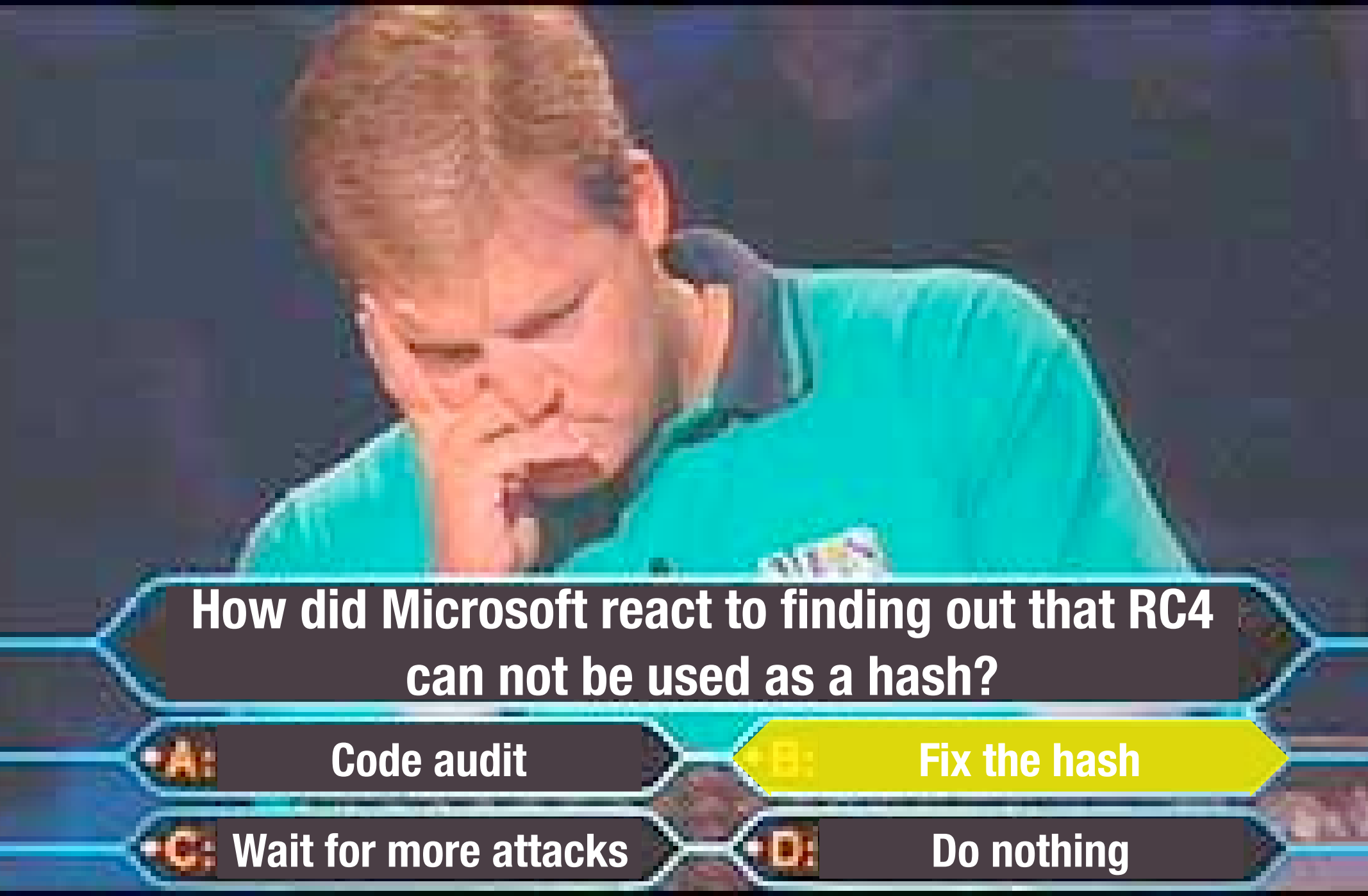
# “mist” II

```
OUTB(0xcf8), 0x80
OUTB(0xcf9), 0x08
OUTB(0xcfa), 0x00
OUTB(0xcfb), 0x80
OUTB(0xcfc), 0x02
```

# Microsoft reacts

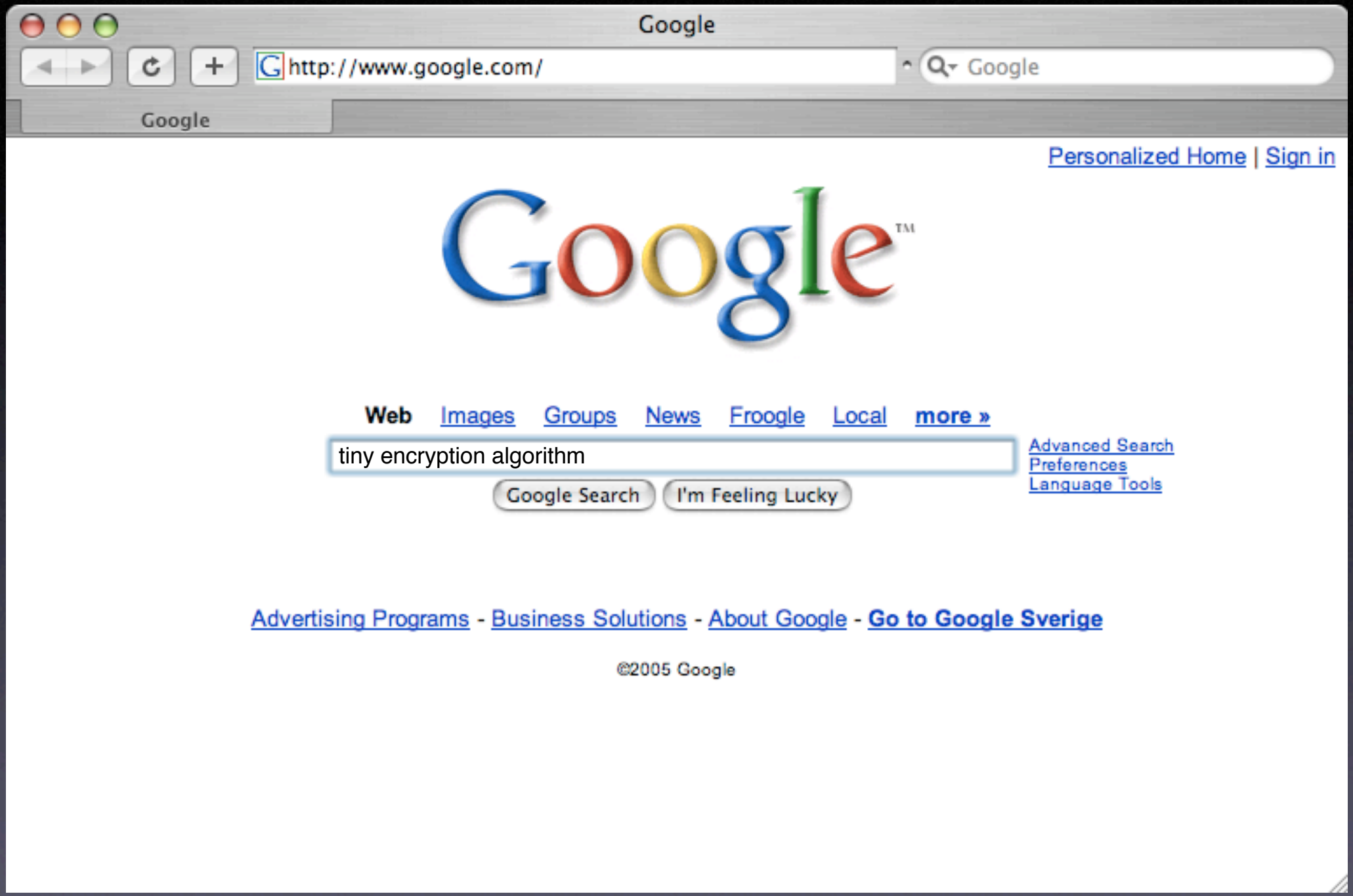
- Just after bunnie found out that RC4 is no hash...
- ... Microsoft reacted.

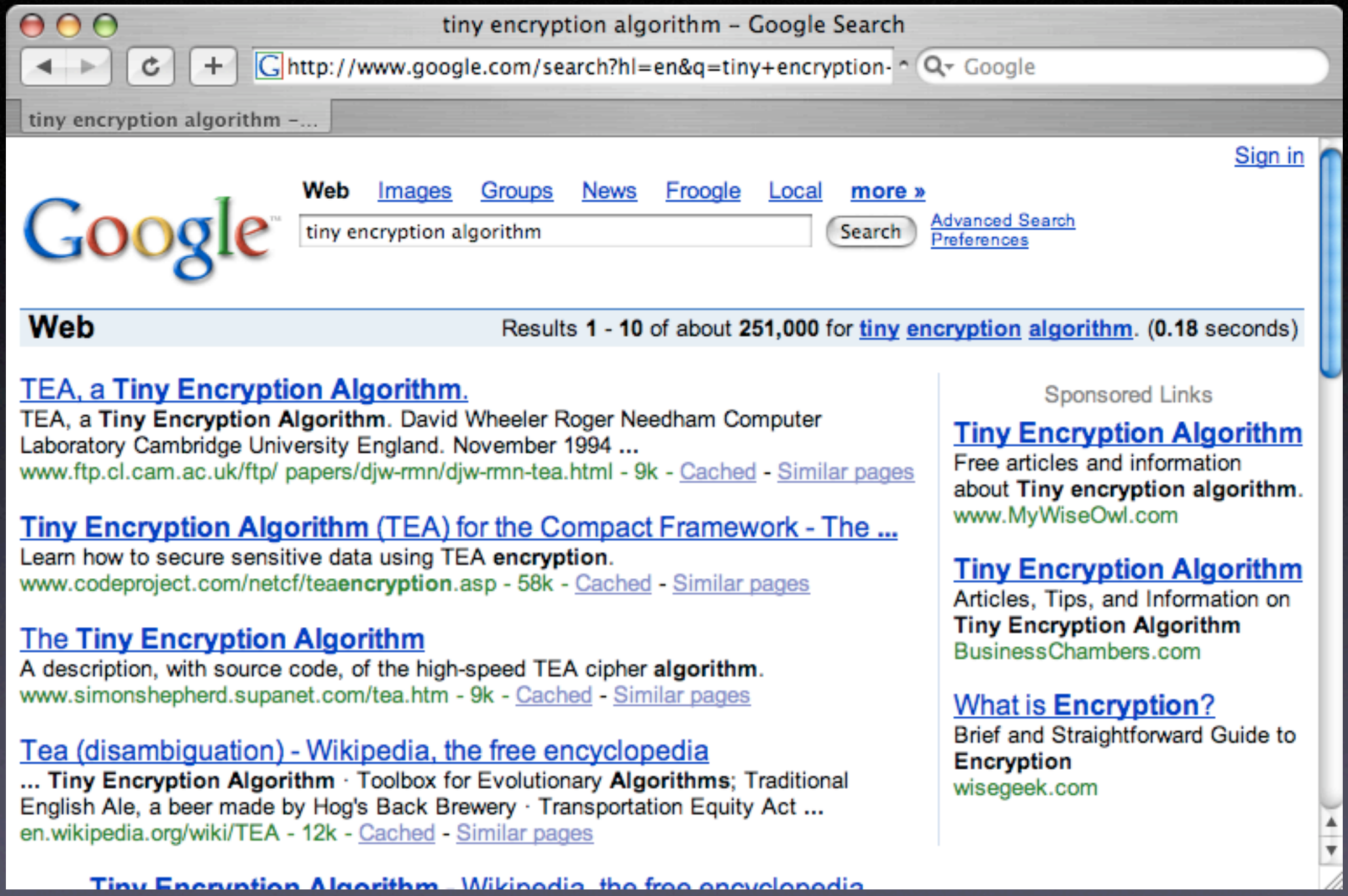

How did Microsoft react to finding out that RC4 can not be used as a hash?
A: Code audit
B: Fix the hash
C: Wait for more attacks
D: Do nothing


How did Microsoft react to finding out that RC4 can not be used as a hash?
A: Code audit
B: Fix the hash
C: Wait for more attacks
D: Do nothing

# Reaction

- RC4 is no hash
- RC5 is no option
- add a tiny hash
- many encryption algorithms can be used as hashes


Google
http://www.google.com/
Google
Google
Personalized Home | Sign in
Google™
Web Images Groups News Froogle Local more »
tiny encryption algorithm
Advanced Search
Preferences
Language Tools
Google Search
I'm Feeling Lucky
Advertising Programs - Business Solutions - About Google - Go to Google Sverige
©2005 Google


tiny encryption algorithm - Google Search
http://www.google.com/search?hl=en&q=tiny+encryption-
Google
tiny encryption algorithm -...
Sign in
Web Images Groups News Froogle Local more »
Google
tiny encryption algorithm
Search
Advanced Search
Preferences
Web
Results 1 - 10 of about 251,000 for tiny encryption algorithm. (0.18 seconds)
TEA, a Tiny Encryption Algorithm.
TEA, a Tiny Encryption Algorithm. David Wheeler Roger Needham Computer Laboratory Cambridge University England. November 1994 ...
www.ftp.cl.cam.ac.uk/ftp/ papers/djw-rmn/djw-rmn-tea.html - 9k - Cached - Similar pages
Tiny Encryption Algorithm (TEA) for the Compact Framework - The ...
Learn how to secure sensitive data using TEA encryption.
www.codeproject.com/netcf/teaencryption.asp - 58k - Cached - Similar pages
The Tiny Encryption Algorithm
A description, with source code, of the high-speed TEA cipher algorithm.
www.simonshepherd.supanet.com/tea.htm - 9k - Cached - Similar pages
Tea (disambiguation) - Wikipedia, the free encyclopedia
... Tiny Encryption Algorithm · Toolbox for Evolutionary Algorithms; Traditional English Ale, a beer made by Hog's Back Brewery · Transportation Equity Act ...
en.wikipedia.org/wiki/TEA - 12k - Cached - Similar pages
Sponsored Links
Tiny Encryption Algorithm
Free articles and information about Tiny encryption algorithm.
www.MyWiseOwl.com
Tiny Encryption Algorithm
Articles, Tips, and Information on Tiny Encryption Algorithm
BusinessChambers.com
What is Encryption?
Brief and Straightforward Guide to Encryption
wisegeek.com

# Fix the Secret ROM

- keep RC4
- add a TEA hash (really tiny)
- update the RC4 key
- trash thousands of Southbridge chips

And that we will be taking an inventory write off in Q2 related to the amount of Xbox MCPs that were made obsolete when MSFT transitioned to a new security code (by way of the MIT hacker) [...].

# Hacking it again

- Extracting the new secret ROM should be trivial
- Just let bunnie dump it again

But there is an easier way...

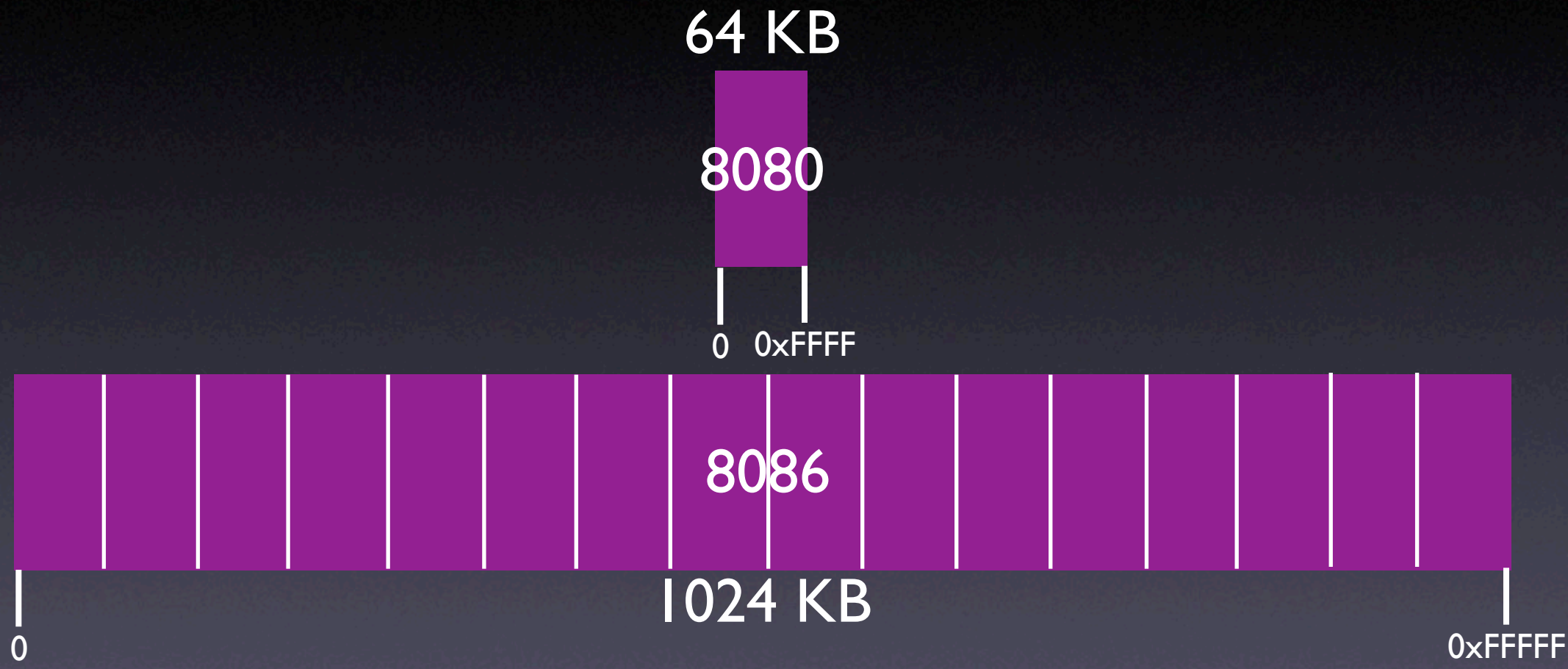
# Hacking it again

- Extracting the new secret ROP should be trivial
- Just let Bunnie jump it again

A20#

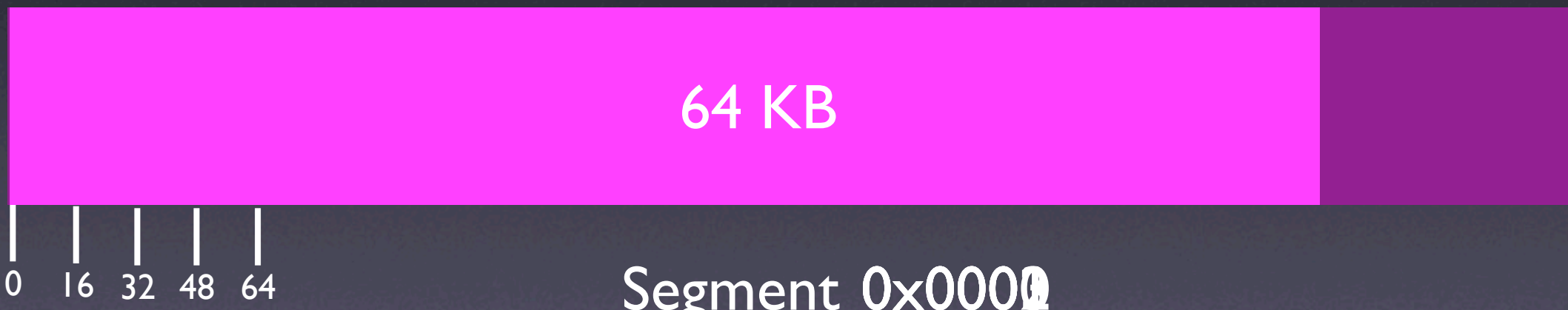
But there is an easier way...

# A History Lesson II



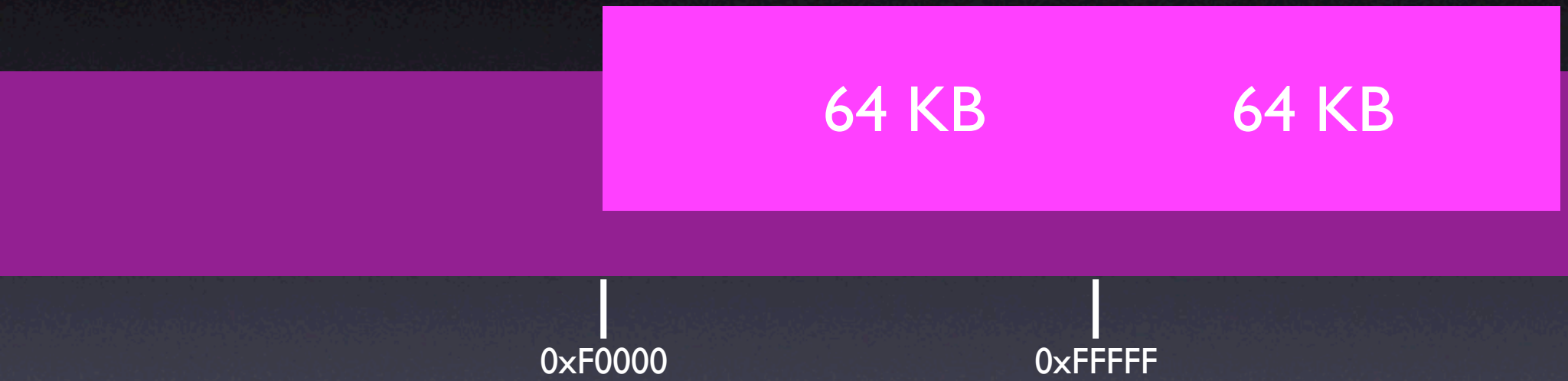

16 bit registers - addresses 0 up to 0xFFFF (64 KB)

solution: segments of 64 KB each




Segment 0x0000

Effective Address = Segment * 16 + Offset




Segment 0xF000

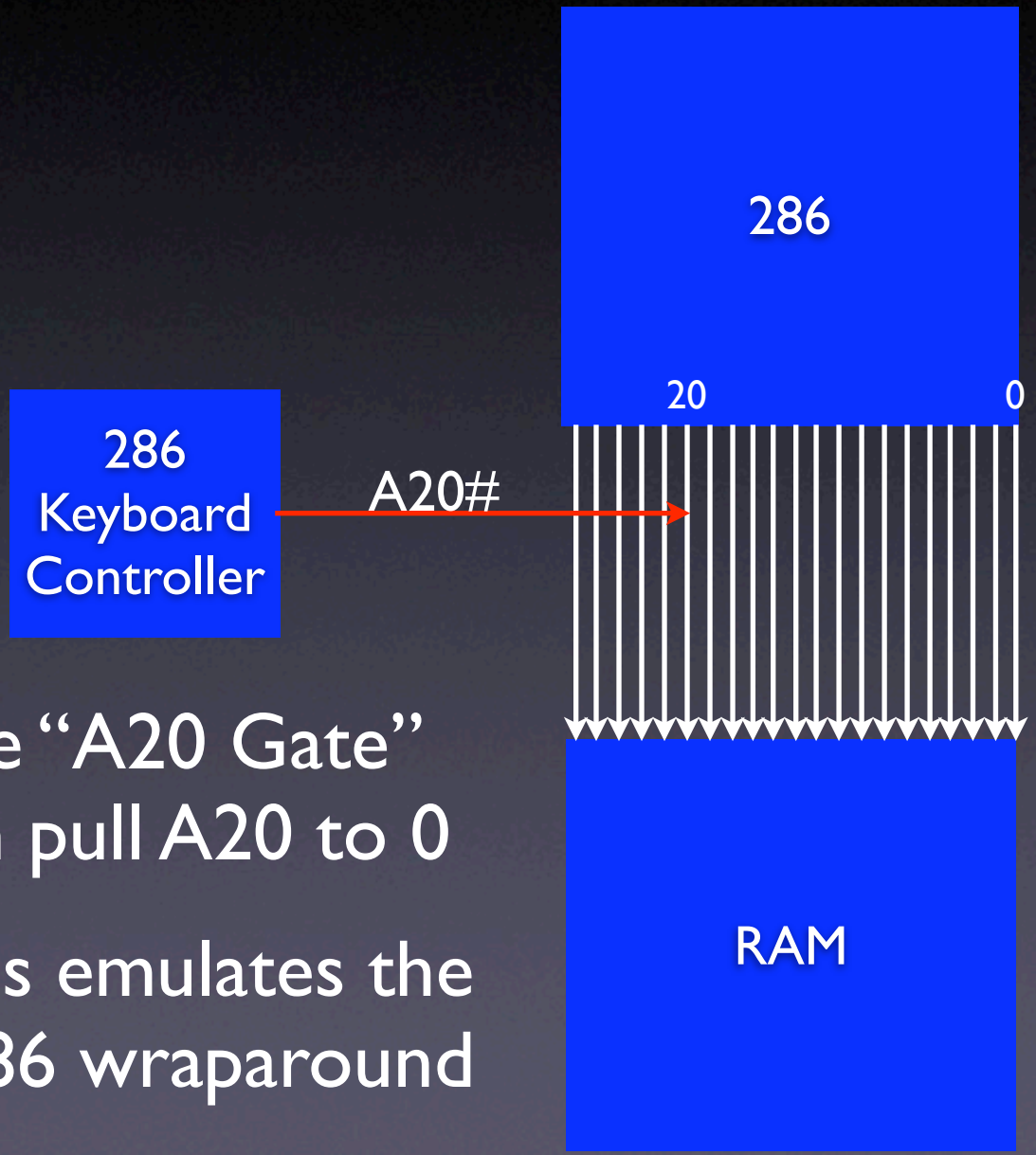
FFFF:0010 = 100000 = 00000 FFFF:FFFF = 10FFF0 = 0FFF0

the 286 is incompatible!

# IBM's Hack

- The 8086 has address lines A0 to A19
- The 286 has an address line A20
- Addresses >= 1 MB won't wrap around
- IBM had to hack around this
- Option: A20 is always 0

# The A20 Gate




- The "A20 Gate" can pull A20 to 0
- This emulates the 8086 wraparound

# The A20 Gate

- Open A20# - you can use memory > 1 MB using Segment:Offset
- MS-DOS calls this “High Memory”
- Close A20# - simulate wraparound
- all addresses are “AND 0xFFEFFFFF”

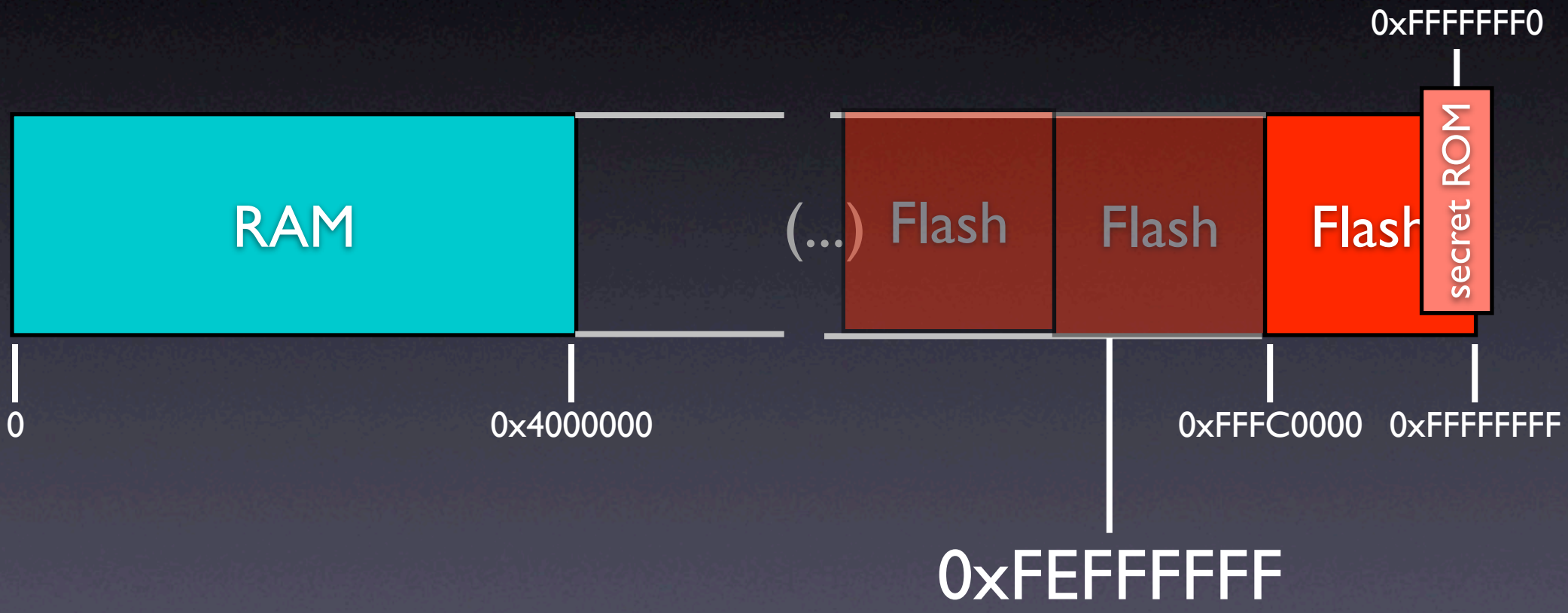
# Legacy Functionality

- Current x86 CPUs still have this functionality
- It is now a pin of the CPU

# A20 Hack Howto

- Connect A20# to GND
- All addresses are “AND 0xFEFFFFFF”
- The CPU starts at...

0xFEFFFFF0




so we connected a modchip and put code at 0xFEFFFFFF to dump the secret ROM to the I2C

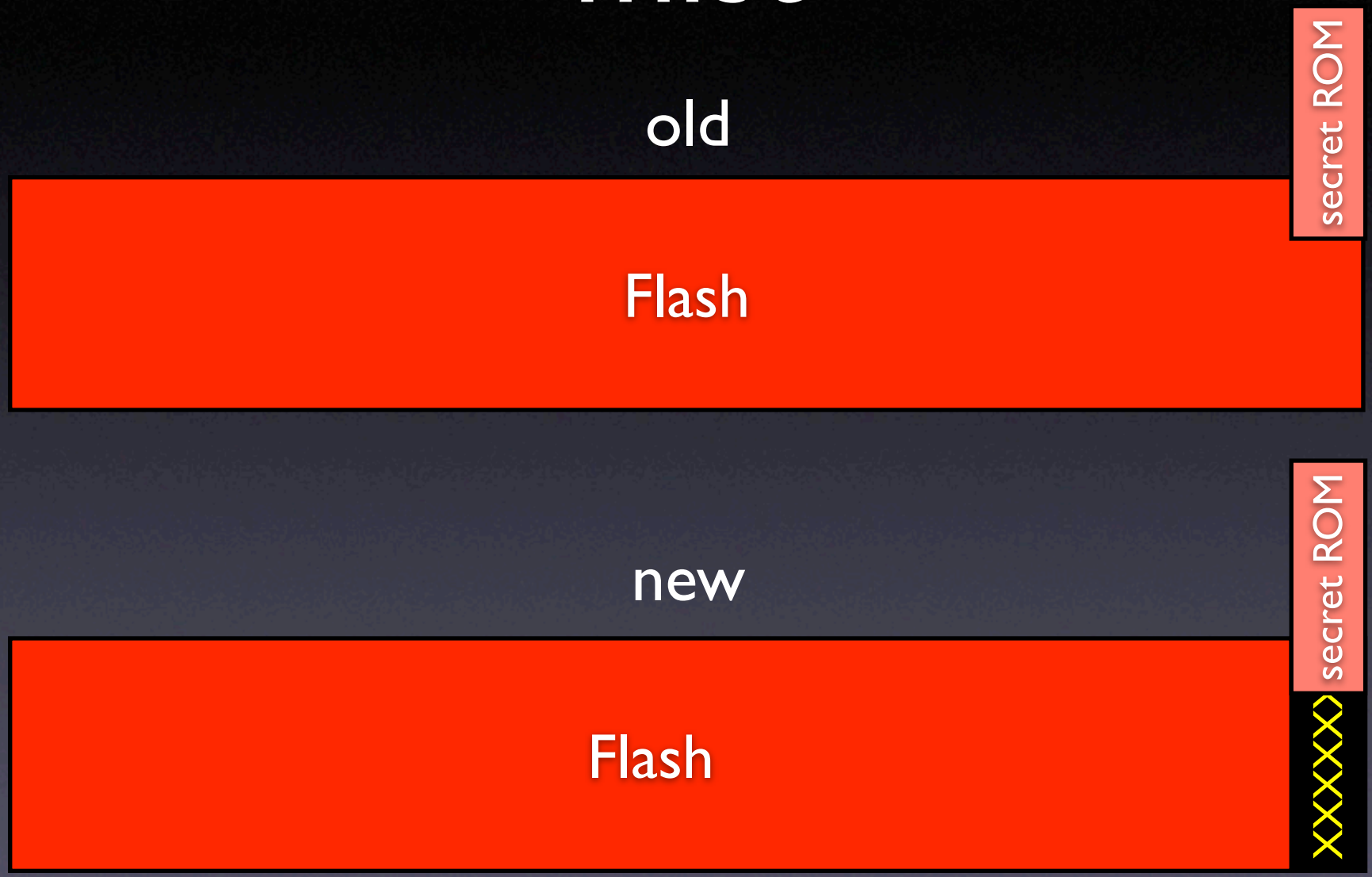
# TEA

Consider complementing the most significant bits of $K_0$ and $K_1$. Note that flipping the most significant bit propagates through both the addition and XOR operations, and flipping it twice cancels the modification. Therefore, modifying the 128-bit master key in this way does not effect the encryption process. We can also complement the most significant bits of $K_2, K_3$ without any effect. This means that each TEA key has 3 other equivalent keys. In particular, it is easy to construct collisions for TEA when used in a Davies-Meyer hashing mode [Win84].

*John Kelsey, Bruce Schneier, and David Wagner*. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. Lecture Notes in Computer Science, 1109: 237–25[illegible], 1996.

We could easily change a JMP to jump to our code.

# “mist”



# “visor”

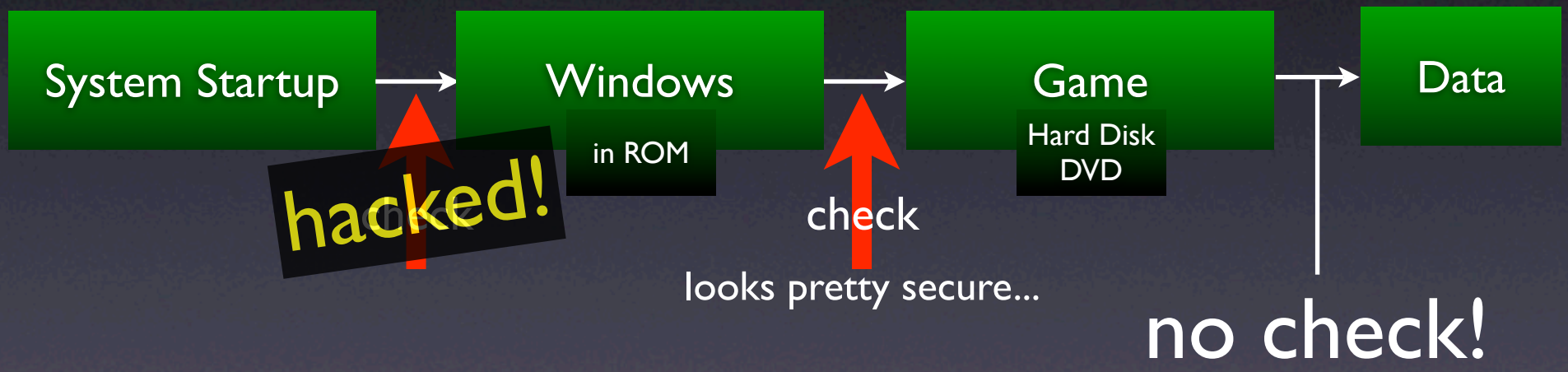
- The FFFFFFFF/00000000 wraparound trick still works.
- Microsoft acted too quickly.
- They should have
  - waited 2 months
  - done an effective code audit
- So there is no need for the attack against TEA...

# Today

- Microsoft did not trash Southbridge chips again.
- latest revision
  - real ROM
  - integrated into PAL/NTSC encoder

LPC override still possible - same Southbridge!

# Non-hardware Attacks




we can try standard buffer exploit methods

# Game exploits

- What data do games load?
- Graphics, audio, video
  - cannot be altered, games don't run from DVD-R
- Savegames
  - stored on hard disk or USB storage (!)

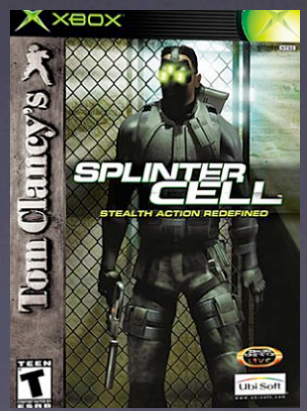
# Savegame Exploits

- David Jilli tried games alphabetically
- What's the first game on the alphabet?

# Savegame Exploit Howto

- “dd” a hacked savegame on a USB stick
- load the savegame







```
Uniform Multi-Platform E-IDE driver Revision: 6.31
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
PCI_IDE: unknown IDE controller on PCI bus 00 device 48, VID=10de, DID=01bc
Keyboard timed out[1]
keyboard: Timeout - AT keyboard not present?(ed)
Keyboard timed out[1]
keyboard: Timeout - AT keyboard not present?(f4)
PCI_IDE: chipset revision 177
PCI_IDE: not 100% native mode: will probe irqs later
    ide0: BM-DMA at 0xff60-0xff67, BIOS settings: hda:pio, hdb:pio
PCI_IDE: simplex device:  DMA disabled
ide1: PCI_IDE Bus-Master DMA disabled (BIOS)
hda: C/H/S=49934/1/232 from BIOS ignored
hda: ST310211A, ATA DISK drive
hdb: THOMSON-DVD, ATAPI CD/DVD-ROM drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
hda: 19541088 sectors (10005 MB) w/512KiB Cache, CHS=19386/16/63, UDMA(33)
Uniform CD-ROM driver Revision: 3.12
Partition check:
 /dev/ide/host0/bus0/target0/lun0: unknown partition table
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 4096 bind 4096)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
RAMDISK: Compressed image found at block 0
```

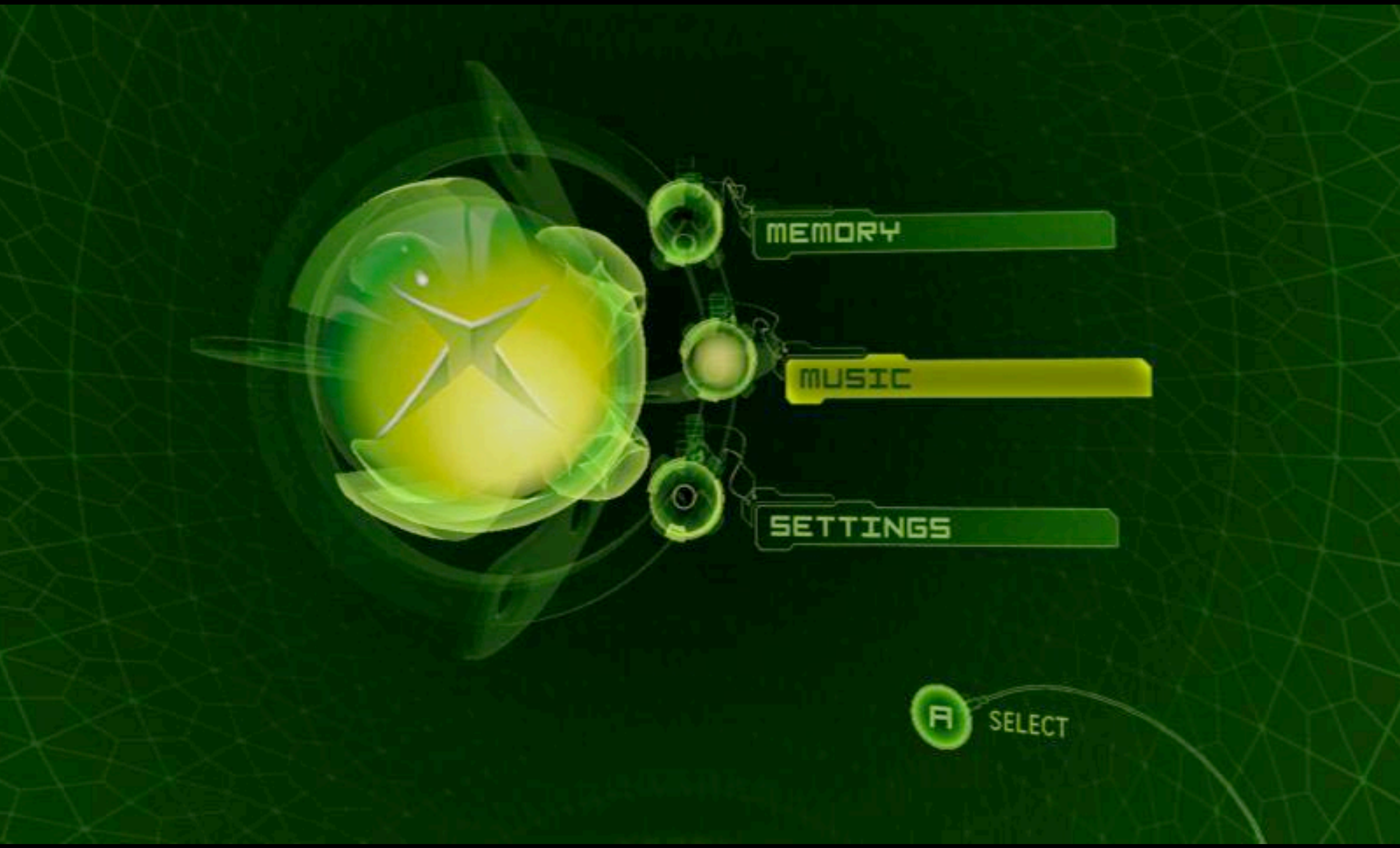
# How is that possible??

- After a game exploit - aren't we in user mode?

no, all games run in kernel mode - we have full control

# Problem

- We must run the game every time to boot Linux
- But there is an application on the hard disk...


MEMORY
MUSIC
SETTINGS
A
SELECT

# Xbox Dashboard

- loads
  - audio
  - 3D meshes
  - fonts

# Xbox Dashboard

- checksums
  - audio
  - 3D meshes
  -

there is a vulnerability in the font handler


MEMORY
MUSIC
LINUX
SETTINGS
A
SELECT

# Complete HOWTO

- “dd” a savegame to a USB stick
- load the savegame
- a script installs the hacked fonts
- every time you turn on the Xbox, you get the hacked Dashboard
- you can run Linux from the menu - or games

# Chain of Mistakes

- USB storage as memory cards
- games in kernel mode
- game exploit
- no font checksum
- font exploit

and there's another integer exploit in the music playlist handler...

# The Fixing Odyssey

1. Microsoft ships fixed version of Dashboard
2. Hackers downgrade to old dashboard (“dd”)
3. Microsoft blacklists old dashboard
4. “xonlinedash.xbe”: same bug, not blacklisted
5. Microsoft blacklists “xonlinedash.xbe”
6. “dashupdate.xbe” on every Xbox Live game DVD: same bug, not blacklisted
7. Microsoft can’t blacklist this one

old games must run on every Xbox

# Today

- You can
  - permanently mod an Xbox
  - to run anything (and games)
  - without opening it

# The 17 Mistakes

- 8 design mistakes
- 6 implementation mistakes
- 3 policy mistakes

these are mistake “classes”

several of them have been made more than once

# I. Design

# #1 Security vs. Money

- There is no such thing as “more secure” or “less secure”.
- Either security is effective or it isn’t.
- There is no sensible compromise.
- Do spend money on security to avoid losses on an ineffective security system!

| | | | |
|---|---|---|---|
| in-system programming of Flash | cheap & faulty RAM chips | secret ROM in Southbridge | no second Southbridge update |

# #2 Security vs. Speed

- Don't trade security for speed.
- Don't be "10%" faster, but "less secure".
- "10%" is nothing - "200%" would be!

all games run in kernel mode

# #3 Combination of Weaknesses

- Be aware that a combination of vulnerabilities can lead to a successful attack.
- Don't add another barrier in front of a potentially vulnerable component.
- Instead, fix the component.

007 + Dashboard = Linux

# #4 Hackers' Resources

- Don't underestimate hackers' resources:
  - access to hardware from work
  - access to hardware from university
  - commercial hackers
- "This would be too expensive/too much work to hack" is a very misleading idea.

MIT's "bunnie" sniffs HyperTransport

# #5 Barriers and Obstacles

- Don't make something "harder for hackers".
- Instead, make it "impossible for hackers".
- Obstacles never slow down hacking significantly.
- You might be mislead into thinking your security system is better.
- Direct these resources into real "barriers" instead.

| savegames must be signed | hard disk ATA-password-protected | hide the secret ROM | games not readable in PC-DVD |
|---|---|---|---|

# #6 Hacker Groups

- Don't use one security system against all attackers.
- Otherwise groups with different goals will unite.
- Instead, find out who your enemies are and what they want, and handle them accordingly.

run Linux
run homebrew
run copies

# #7 Security by Obscurity

- ...does not work.
- But well-proven algorithms do work (SHA-1, RSA, ...)
- ...if used correctly.

| hide secret ROM | encrypt Flash contents | hide DVD contents | hide hard disk Contents |
|---|---|---|---|

# #8 Fixes

- Don't release "quick" fixes:
  - Fixes may be flawed.
  - More holes tend to be found soon after that.
- Instead, audit the complete security system again, making use of that new knowledge
- Follow hackers' progess for some more time

| secret ROM hash function | Dashboard font bug odyssey |
| --- | --- |

# II. Implementation

# #9 Data Sheets

- Read data sheets.
- Don't just hack around, don't assume anything.
- Be very careful with components that have legacy functionality.

| A20# vulnerability | Intel's "visor" wraparound |
| --- | --- |

# #10 Literature

- Read standard literature.
- For crypto, that's at least Schneier.
- (Hint for post-2004: Read all external links of the Wikipedia article on the topic)

RC4 as a hash

TEA as a hash

# #II Pros

- Get experienced professionals.
- Your engineers must have a background on security systems.
- Don’t get students on internships.

implementation of secret ROM

# #12 Completeness

- Check whether your code catches all cases.
- Instead, your work has the very opposite effect: It gives hints to attackers.

| secret ROM turnoff check | hash everything but fonts |
| --- | --- |

# #13 Leftovers

- Look at the *final* product from the perspective of a hacker.
- Hexdump and disassemble your final builds.

old version of
secret ROM
in Flash

# #14 Final Test

- Test your security system (again) when you have all the *final* components in place.
- Even small changes can break everything else - especially security.

Switch from AMD to Intel

Switch from RC5 to RC4

# III. Policies

# #15 Source

- Keep your source safe.
- Find engineers you can trust.

leaked Xbox source code

# #16 Many People

- Have *many* people look at your design and your implementation.
- Find engineers you can *trust* instead of preventing them of seeing the source.

obviously weak QA on many parts

# #17 Talk

- Know your “enemy” - and talk to them!
- “Not talking to terrorists” is stupid.
- They are not your enemy by definition - they just want to reach their goals.
- Compromises are a good thing.

| don’t talk about 007 exploit | don’t talk about font exploit |
| --- | --- |

# Summary of the Xbox "Security System"

- broken by design
- broken by implementation
- broken by policies
- broken.

## 17 Mistakes Microsoft Made in the Xbox Security System

Michael Steil <mist@c64.org>
Xbox Linux Project http://www.xbox-linux.org/

### Introduction

The Xbox is a gaming console, which has been introduced by Microsoft Corporation in late 2001 and competed with the Sony Playstation 2 and the Nintendo GameCube. Microsoft wanted to prevent the Xbox from being used with copied games, unofficial applications and alternative operating systems, and therefore designed and implemented a security system for this purpose.

This article is about the security system of the Xbox and the mistakes Microsoft made. It will not explain basic concepts like buffer exploits, and it will not explain how to construct an effective security system, but it will explain how *not* to do it: This article is about how easy it is to make terrible mistakes and how easily people seem to overestimate their skills. So this article is also about how to avoid the most common mistakes.

For every security concept, this article will first explain the design from Microsoft's perspective, and then describe the hackers' efforts to break the security. If the reader finds the mistakes in the design, this proves that Microsoft has weak developers. If, on the other hand, the reader doesn't find the mistakes, this proves that constructing a security system is indeed hard.

### The Xbox Hardware

Because Microsoft had a very tight time frame for the development of the Xbox, they used off-the-shelf PC hardware and their Windows and DirectX technologies as the basis of the console. The Xbox consists of a Pentium III Celeron mobile 733 MHz CPU, 64 MB of RAM, a GeForce 3 MX with TV out, a 10 GB IDE hard disk, an IDE DVD drive, Fast Ethernet, as well as USB for the gamepads. It runs a simplified Windows 2000 kernel, and the games include adapted versions of Win32, libc and DirectX statically linked to them.

Although this sounds a lot more like a PC than, for example, a GameCube with its PowerPC processor, custom optical drive and custom gamepad connectors, it is important to point out that, from a hardware point of view, the Xbox shares *all* properties of a PC: It has LPC, PCI and AGP busses, it has IDE drives, it has a Northbridge and a Southbridge, and it includes all the legacy PC features such as the "PIC" interrupt controller, the "PIT" timer and the A20 gate. nVidia sold a slightly modified Southbridge and a Northbridge with a another graphics core embedded for the PC market as the "nForce" chipset between 2001 and 2002.

### Motivation for the Security System

The Xbox being a PC, it should be trivial to install Linux on it in order to have a cheap and, for that time, powerful PC. Even today, a small and silent 733 MHz PC with TV connectivity for 149 USD/EUR is still attractive. But this is not the only thing Microsoft wanted to prevent. There are three uses that should not have been possible:

- **Linux**: The hardware is subsidized and money is gained with the games, therefore people should not be able to buy an Xbox without the intent to buy any games. Microsoft apparently feels that allowing the Xbox to be used as a (Linux) computer would be too expensive for them.
- **Homebrew/Unlicensed**: Microsoft wants the software monopoly on the Xbox platform. Nobody should be able to publish unlicensed software, because Microsoft wants to gain money with the games to amortize the hardware losses, and because they do not want anyone to release non-Internet Explorer browsers and non-Windows Media Player multimedia software.
- **Copies**: Obviously it is important to Microsoft that it is not possible to run copied games on the Xbox.

Microsoft decided to design a single security system that was supposed to make Linux, homebrew/unlicensed software and copies impossible. The idea to accomplish this was by simply locking out all software that is either not on the intended (original) medium or not by Microsoft.