Analýza AMD x86 SMU firmwaru

Ruik
r . marek @ assembler.cz

Osnova přednášky

- Historie platform procesorů
- Platformové procesory dnes
- Analýza:
  - Hardwaru
  - Firmwaru

Pomocníci x86

- x86 s námi už od roku 1978
- IBM PC XT
  - Intel 8048 – Od roku 1983
- IBM PC AT
  - Intel 8042 – Od roku 1984, klávesnice, A20, reset systému
- Notebooky
  - Všelijaké menší porcesory - embedded controllers
  - Obyčejně 8051 kompatibilní nebo H8

Kde všude je firmware?

- V periferních zařízeních
  - GPU, bezdrátové sítě, Ethernet, úložiště, USB apod...
- V systémových řadičích (chipset) případně v procesoru
  - Intel
    - Management Engine/AMT atd .. (ME, ARC, SPARC)
  - AMD
    - System Management Unit (SMU, ?)
    - Integrated Microcontroller (IMC, 8051, southbridge)
    - USB3.0 controllers (USB, domácí úkol!!!!)
    - Platform Security Processor (PSP, ARM Cortex A5 s TrustZone)
  - ARM, PowerPC too (DMA nebo kanálové řadiče s firmware)

Chyby a Malware

Kde?	Chyby?	Malware?
Aplikace	X	X
Aplikační knihovny	X	X
Operační systémy	X	X
Periferní procesory	X	?

Pozor na bezpečnost!

- Rostoucí bezpečnostní problém
  - Firmware bloby jsou všude x86, PowerPCs, ARMS atd...
- Vývojáři hardware != vývojáři software
- První opensource firmware na netypických procesorech:
  - Psychson (badUSB) – 8051, USB3.0 controller
  - Sprite_tm – ARM, MMU-less Linux na hardiskovém řadiči
- A closedsource:
  - NSA, BIS, SIS :) ? kdo ví ?

Začátek příběhu

- Někdo čte knihy....
- Někdo datasheety.... (taky)
  - AMD BIOS and Kernel Developers Guide (BKDG)
    - for fam15h & fam16h

  “The system management unit (SMU) is a subcomponent of the northbridge that is responsible for a variety of system and power management tasks during boot and runtime. The SMU contains a microcontroller to assist”

  - Northbridge je dnes část procesoru

Jaký procesor???

- Vygooglíme... "AMD" "system management unit"
- Dostaneme linky na nějaký datasheety
  - S trochou informací navíc, ale k nim později
- A taky linkedin.com :)

  “Developed a NLMS Adaptive Filter in firmware C for an embedded **Lattice LM32 microprocessor** in the **SMU IP**, to dynamically compute the coefficients used for calculating the GPU dynamic power consumption. This also includes creating a Matlab model of the adaptive filter and running simulations with real silicon data to verify the algorithm functionality.”

LatticeMico lm32

- LatticeMico32 Open, Free 32-Bit Soft Processor
- Info v “LatticeMico32 Processor Reference Manual”
- SDK from LatticeMico System for Diamond 3.3
- Hodí se vlastní toolchain:
  - gcc, objdump, as, gdb...
- http://www.latticesemi.com/en/Products/DesignSoftwareAndIP/IntellectualProperty/IPCore/IPCores02/LatticeMico32.aspx

Architektura LM32

- 32-bit processor, fixní 32-bit instrukce
- 32 general-purpose registers (R0-R31)
  - R0 – zero
  - R26 – GP (global pointer)
  - R27 – FP (frame pointer)
  - R28 – SP (stack pointer)
  - R29 – RA (return address, like LR on PowerPC/ARM)
  - R30 – EA (exception address)
  - R31 – BA (breakpoint address)

To je supr ale kde je firmware?

- Hledej flash chip
- Zkus BIOS image
  - Jeho část je AMD AGESA (viz přednáška coreboot)
  - AMD Generic Encapsulated Software Architecture (AGESA)
  - Blob s inicializací v corebootu některé I jako opensource
- Hledej šmudlo v BIOSech pro soket FM2
- A pak... textové vyhledávání

Hledáme firmware

- Hledejme* “SMU”
- Najdeme ji pro všechny procesory fam15h
  - Trinity, Richland, Kaveri, Kabini etc

```
007acf30  00 00 00 00 5f 53 4d 55  5f 53 4d 55 80 dd 00 00  |...._SMU_SMU....|
007acf40  00 20 00 00 00 00 01 00  e7 62 54 d9 54 4e 00 00  |. .......bT.TN..|
007acf50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
007acf60  00 00 00 00 0a 00 0a 00  40 00 00 00 14 37 00 00  |........@....7..|
007acf70  00 01 01 00 10 98 c4 cd  97 53 2d 93 cc 9d 09 f7  |.........S-.....|
007acf80  c4 6e ea f7 1e a0 9c f8  c0 d9 01 00 d0 da 01 00  |.n..............|
007acf90  00 00 00 00 f1 da 01 00  00 db 01 00 14 da 01 00  |................|
007acfa0  38 dc 01 00 00 00 00 00  00 00 00 00 00 00 00 00  |8...............|
007acfb0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
007ad060  55 aa 55 aa 00 00 00 98  00 00 00 98 00 00 00 d0  |U.U.............|
```

* Pozn: Pro verzi binárního blobu AMD hledejme “AGESA”

Přístup z hlavního procesoru

- Zdokumentováno v AMD BKDG 15h, 16h)
- Přístup do adresního prostoru LM32
  - Přes PCI registers 0xB8 (adresa) and (0xBC data) PCI 0:0.0
  - Napišme si prográmek a uvidíme co dostaneme

SMU address space dump

```
00000000  55 55 aa aa 55 55 aa aa  55 55 aa aa 55 55 aa aa  |UU..UU..UU..UU..|
*
00010000  0a 00 0a 00 40 00 00 00  14 37 00 00 00 01 01 00  |....@....7......|
00010010  10 98 c4 cd 97 53 2d 93  cc 9d 09 f7 c4 6e ea f7  |.....S-......n..|
00010020  1e a0 9c f8 c0 d9 01 00  d0 da 01 00 60 f1 01 00  |............`...|
00010030  f1 da 01 00 00 db 01 00  14 da 01 00 38 dc 01 00  |............8...|
00010040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000100f0  00 00 00 00 00 00 00 00  00 00 00 00 55 aa 55 aa  |............U.U.|
00010100  55 55 aa aa 55 55 aa aa  55 55 aa aa 55 55 aa aa  |UU..UU..UU..UU..|
*
0001dc50  55 55 aa aa ac c3 01 00  40 2f 01 00 fc 72 01 00  |UU......@/...r..|
0001dc60  78 ac 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |x...............|
0001dc70  3c b1 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |<...............|
0001dc80  74 af 01 00 38 2f 01 00  88 07 01 00 88 07 01 00  |t...8/..........|
0001dc90  88 07 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |................|
0001dca0  88 07 01 00 88 07 01 00  88 07 01 00 78 d9 01 00  |............x...|
0001dcb0  7c d9 01 00 88 07 01 00  bc 21 01 00 88 07 01 00  ||........!......|
0001dcc0  88 07 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |................|
0001dcd0  88 07 01 00 88 07 01 00  44 33 01 00 a0 42 01 00  |........D3...B..|
0001dce0  88 07 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |................|
*
0001dd30  88 07 01 00 88 07 01 00  00 00 00 00 00 01 00 00  |................|
0001dd40  c1 46 0c 00 01 00 00 00  00 00 00 00 00 00 00 00  |.F..............|
0001dd50  00 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
0001dd60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0001de50  00 00 00 00 00 00 00 00  c4 da 01 00 01 01 00 00  |................|
```

Adresní prostor SMU

Start	Konec	Účel	Pozn
0x00000000	0x0000ffff	Neznámý	Jenom opakující se pattern of 0x55 0xaa 0x55 0xaa
0x00010000	0x000100ff	Viditelná hlavička firmwaru z BLOBu	
0x00010100	0x0001dc53	Asi schovaný firmware???	0x55 0xaa...
0x0001dc54	0x0001ffff	Náhodná? data	
0x00020000	0x000203ff	Težko říct	0x55 0xaa...
0x00020400	0x...?	Nemám páru....	
0xe0000000	?	Hardware registers	

Hlavička Firmware

Offset	Value	Usage
0x0000	0x0a 0x00 0x0a 0x00	Version (Minor, Major)
0x0004	0x40 0x00 0x00 0x00	Délka hlavičky (in 4 bytes)
0x0008	0x14 0x37 0x00 0x00	Délka těla (in 4 bytes, without header)
0x000C	0x00 0x01 0x01 0x00	0x10100 Entrypoint?
0x0010	Random data	Checksum?
0x00FC	0x55 0xaa 0x55 0xaa	Signature
0x0100	0x00 0x00 0x00 0x98	První instukce?

```
00000000  0a 00 0a 00 40 00 00 00  14 37 00 00 00 01 01 00  |....@....7......|
00000010  10 98 c4 cd 97 53 2d 93  cc 9d 09 f7 c4 6e ea f7  |.....S-......n..|
00000020  1e a0 9c f8 c0 d9 01 00  d0 da 01 00 60 f1 01 00  |............`...|
00000030  f1 da 01 00 00 db 01 00  14 da 01 00 38 dc 01 00  |............8...|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 55 aa 55 aa  |............U.U.|
```

Kde je první instrukce?

- Možná že je hned za hlavičkou (byte 256)
- Začíná takhle:
  - 0x00 0x00 0x00 0x98 0x00 0x00 0x00 0x98
  - srui r0,r0,152
  - srui r0,r0,152
  - Je to bordel...
- Ale zkusíme Big Endian...
  - xor r0, r0
  - xor r0, r0
- Nastavení R0 na 0 to vypadá dobře

Firmware blob co dál...

- Vyrobíme ELF
  - A překlopíme do Big Endian
  - ELF takhle:
    - lm32-elf-objcopy -S -I binary -O elf32-lm32 -B lm32 --rename-section .data=.text,alloc,load,contents,code,readonly --change-address 0x10000 --set-start=0x100 $1 fw.elf
- A pak objdump...

Zdisassemblujeme...

- Nastaví to R0, zakáže přerušení a nastaví exception vectory...

```
10100:      98 00 00 00     xor r0,r0,r0
10104:      98 00 00 00     xor r0,r0,r0
10108:      d0 00 00 00     wcsr IE,r0
1010c:      78 01 00 01     mvhi r1,0x1
10110:      38 21 01 00     ori r1,r1,0x100
10114:      d0 e1 00 00     wcsr EBA,r1
10118:      d1 21 00 00     wcsr DEBA,r1
1011c:      f8 00 00 39     calli 0x10200
```

Stack....

- A nastaví se stack

```
10200:      98 00 00 00     xor r0,r0,r0
10204:      78 1c 00 01     mvhi sp,0x1
10208:      3b 9c eb fc     ori sp,sp,0xebfc
1020c:      78 1a 00 02     mvhi gp,0x2
10210:      3b 5a 5d 40     ori gp,gp,0x5d40
```

Další analýza

- Napíšeme linker script a přidáme symboly do našeho ELFu
  - Nápověda: Exception handling najdeme v LM32 manuálu

```
.text :
  {
    . = ALIGN(4);
_start = 0x100;
RESET = 0x100;
BREAKPOINT = 0x120;
INSTRBUSERROR = 0x140;
WATCHPOINT = 0x160;
DATABUS_ERR = 0x180;
DIV = 0x1a0;
INT =  0x1c0;
SYSCALL = 0x1e0;
RESET_INIT = 0x200;
```

Analyzujeme dál...

- Typický Function prologue & epilogue

  Přerušovací rutiny

  - Saves/Restore všech registrů
  - Na stacku místo pro lokální proměnný

  – Normální funkce

  - Save/Restore jenom (r11-r26), viz ABI
  - Na stacku místo pro lokální proměnný

```
addi sp,sp,-16
sw (sp+16),r11
sw (sp+12),r12
sw (sp+8),r13
sw (sp+4),ra
…
lw r11,(sp+16)
lw r12,(sp+12)
lw r13,(sp+8)
lw ra,(sp+4)
ret
```

Bezva finta

- Říkal jsem že má Lattice SDK?
- Tak se mrknem
  - Crt0ram.S
  - Další C funkce pro registraci ISR & IRQs
- Supr AMD to použilo taky...
- Přidáme další funkce do našeho linkerskriptu

```
MicoISRHandler = 0x448;
MicoISRInitialize = 0x500;
MicoRegisterISR = 0x54c;
MicoDisableInterrupt = 0x604;
MicoEnableInterrupt = 0x660;
MicoEnableInterrupts = 0x6d8;
MicoDisableInterrupts = 0x728;
```

Komunikace s firmwarem

- BIOS může spustit SMU firmware service request
- Dokumentováno v BKDG, ale jen pro jeden
  - Číslo service requestu do registru
  - A pak zmáčknout bit v doorbell registeru

SMU Firmware requests detailně

- Většina se týká power managementu
  - Budget Aware Power Management (BAPM)
  - Často přes registry v oblasti 0x1f000-0x1ffff
- Zajímavé a jednoduché služby
  - Flush data cache
  - Flush instruction cache
  - “Ping” request
    - Zvyš byte o 1 v diagnostickém registru

Velké pokušení

- Teď toho víme dost...
- jestlipak pustíme I vlastní kód?
  - Pro legraci...
  - A jak...
  - Kde můžou být problémy
- Podívejme se na hlavičku firmware ještě jednou
  - Bajty 0x10 - 0x23 vypadají dost náhodně...
  - Dohromady 20 bajtů (8 trilobitů = jeden trilobajt)
  - 160 bitů...

Kontrolní součet

- Je to160 bitů celkem náhodných
- Mohlo by to být SHA1
- Zkousil jsem všechno možný
  - S hlavičkou
  - Bez
  - Little/big endian
  - Hlavička s nulama...
  - Ale nikdy to nevyšlo
- Navíc, pak bychom museli přepočítat I hlavičky jinde v BIOSu...

Runtime code injection?

- Code a data segment jsou skryty při pohledu z procesoru x86...
- Ale ne všechno, neboť celá oblast není zaplácnutá
  - Podívejme se ještě jednou do dumpu
  - 0x1dc54 → 0x1ffff binary garbage
  - Jsou to jen data?

Data z otevřených částí

- X86 vidí jen dvě části
  - Hlavičku – 256 bytes
  - A nějaký data nakonci 64KB
    - Offset data
    - Bordel?
    - BIOS communication area

```
<Header is here>
00000100  55 55 aa aa 55 55 aa aa  55 55 aa aa 55 55 aa aa  |UU..UU..UU..UU..|
*
0000dc50  55 55 aa aa ac c3 01 00  40 2f 01 00 fc 72 01 00  |UU......@/...r..|
0000dc60  78 ac 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |x...............|
0000dc70  3c b1 01 00 88 07 01 00  88 07 01 00 88 07 01 00  |<...............|
```

Offset data

- Offset data jsou skutečně adresy funkcí z firmwaru z BIOS blobu...
- Wtf ?

```
0001dc50: aaaa5555
0001dc54: 0001c3ac
0001dc58: 00012f40
0001dc5c: 000172fc
0001dc60: 0001ac78
0001dc64: 00010788
0001dc68: 00010788
0001dc6c: 00010788
0001dc70: 0001b13c
0001dc74: 00010788
0001dc78: 00010788
0001dc7c: 00010788
0001dc80: 0001af74
```

Něco se dost pokazilo....

- Vypadá to že 256 bajtů není zakryto
- Hlavička má taky 256 bytes
  - Délka firmwaru v hlavičce je taky bez délky hlavičky!
- Můžeme modifikovat ty offset data za běhu z x86?
  - Yep!
- Je tam funkce co se spustí z těch offset dat?
  - Yep! - ISR dispatcher pouští obsluhu přerušení ...
  - Navíc je tam taky handler pro obsluhu service requestů...

Jak spustit vlastní program?

- Nahrajeme ho do nepoužívané oblasti...
- Změníme pointer z těch offset dat....
- A pak zavoláme ten původní...

The SMU request handler

- SMU request handler provádí:
  - ACK IRQ (0xe0003004)
  - Nahraj Request number (0xe0003000)
  - Maskuj 0xfffe
  - Posun o 1 doprava
  - Nahraje pointer na pole pointrů obsluhy requestů
  - Posun request number o 2 doleva
  - Nahraj function address to r2
  - Call r2
  - Wtf???
  - Žádná kontrola přetečení pole!
  - Signalizuj Intdone and intack

```
mvhi r1,0xe000
mvi r3,1
ori r1,r1,0x3000
mvhi r12,0xe000
ori r12,r12,0x3004
sw (r12+0),r3
lw r3,(r1+0)
mvhi r1,0x1
ori r1,r1,0xfffe
and r3,r3,r1
mvhi r1,0x1
srui r3,r3,1
sli r3,r3,2
ori r1,r1,0xbad4
add r3,r3,r1
lw r2,(r3+0)
call r2
mvi r4,3
sw (r12+0),r4
lw r12,(sp+8)
lw ra,(sp+4)
addi sp,sp,8
ret
```

Něco se dost zeslonilo...

- My můžeme předat request value (15 bitů)
  - Offset začátku tabulky requestů je známý též
- Takže můžeme spustit libovolnou funkci...
  - I tu co tam nahrajeme my...
- Stačí spočítat číslo falešného requestu
  - Bude vynásobeno 4 a přičteno ke známému offsetu
  - Na cílové adrese uložíme jen pointer na funkci co chceme spustit...
  - A je to...

Adresní prostor SMU

Start	Konec	Užití	Pozn
0x00000000	0x0000ffff	**Tak teď copak máme tady?**	**Vidíme jen pattern of 0x55 0xaa 0x55 0xaa**
0x00010000	0x000101ff	Viditelná hlavička firmware	Stejná jako v BIOS image
0x00010100	0x0001dc53	Kód firmwaru...	0x55 0xaa...
0x0001dc54	0x0001ffff	Náhodná data...	
0x00020000	0x000203ff	Netuším	0x55 0xaa...
0x00020400	0x...?	Težko říct	
0xe0000000	?	Hardware registry	

Dump tajné části

- Napíšeme program
  - Nový SMU service request
  - Kopíruj paměť aka memcpy do oblasti kam vidíme
- Spustíme opakovaně...
- Teď už jen zanalyzovat co dostaneme...

Tajná ROM

- Je asi ROM
- Stejná struktura jako hlavní firmware
- Složitější inicializace
- Implementuje jenom SMU service request 0
  - Kontroluje image nahraný BIOSem
  - Takže se konečně dozvíme:
    - Jaký je to hash
    - A kde udělali soudruzi z Texasu chybu?

Authentikace firmwaru

- Divný konstanty
- Jako 0x98badcfe, 0x10325476
- Nebo I 0x36363636, 0x5c5c5c5c
- googlujme...

```
78 03 98 ba    mvhi r3,0x98ba
38 43 dc fe    ori r3,r3,0xdcfe
```

  - SHA1 a taky HMAC ...
- Jak se vyznat ve složitějších funkcích
  - Zkusíme QEMU + GDB

QEMU

- Má podporu pro LM32!
- Stačí poupravit pro SMU memory layout
- Nahrajeme firmware
  - ROM i RAM část (0-64KB, 64-128KB)
- Spustíme ROM SMU request function v QEMU
- Oddebugujeme v GDB...

Authentifikace firmwaru

- Nahraje data z hlavičky firmware
- Spláchne cache
- Spočte hash
- Spláchne cache
- Kontrola hashe s hashem v hlavicce v konstantním čase
- Nastaví ochranné registry
  - Problém: někdo k offsetu nepřipočetl délku hlavičky
- Nastaví reset vector na zauthentifikovaný firmware
- Pošle BIOSu výsledky

Hash z hlavičky

- Není checksum ale hash
- Debugování pomohlo
- Požitý HMAC/SHA1 s tajným klíčem!
  - Klíč je symetrický
- A kdo ho má...
  - Si může ... třeba podepsat vlastní FW.
  - Viz další slide...

Tajemství

42?

Tajemství

- Zůstane tajemstvím
- Tak jsem napsal do AMD ...
  - Ale co?

    “To whom it may concern,

    I have a discovered a security vulnerability in the recent AMD processors which allows arbitrary code execution on the System Management Unit (SMU).”
  - A komu?
    - Tak zase někoho najít přes linked in...
  - Jak podpořit tvrzení že jsou pwned...
    - Změnit diagnostickou funkci aby přičítala 0x42!
    - A přepočítat hash
    - Poslat do AMD

Časová osa

- Firmware byl analyzován o vánocích 2013
- Chyba nalezena v dubnu 2014
- 30.4.2014 – Email do AMD
- 15.5.2014 – Odpověď
- 16.5.2014 – PGP komunikace...
- 09.7.2014 – AMD potvrzuje problém
- Občasná komunikace
- 25.11.2014 – AMD posíla seznam opravených verzí

AMD odpovídalo na emaily a bylo nápomocno

Opravené problémy

- Oba problémy opravené
- Celý firmware je teď delší takže je celý schován
- The SMU request function kontroluje meze
- Podobný ale ne stejný problém opraven I v Kabini a Kaveri

Opravené verze

- Opravený SMU firmware je součást nových AMD AGESA
- **Nový BIOS obsahuje opravenou AGESAu a tudíž I firmware.**

Processor	AGESA version	SMU version *	CPU family
Trinity	1.1.0.7	10.14 (0x000a000e)	fam15h
Richland	1.1.0.7	12.18 (0x000c0012)	fam15h
Kabini	1.1.0.2	12.21 (0x000c0015)	fam16h
Kaveri	1.1.0.7	13.52 (0x000d0034)	fam15h

* Note: stored as LSB first in the BIOS image: 0x09 0x00 0x0a 0x00 means version 10.09

Otázky?

Díky, Ruik

r . marek @ assembler.cz

Domácí úkol

- USB 3 firmware v AMD chipsetech...
  - Dá se zanalyzovat podobně
- Buď zodpovědný, kontaktuj výrobce
  - Responsible disclosure