CROWDSTRIKE

Ninjas and Harry Potter

“Spell”unking in Apple SMC Land

Alex Ionescu, Chief Architect
NoSuchCon 2013

@aionescu
alex@crowdstrike.com

Bio

- Reverse engineered Windows kernel since 1999
  - Lead kernel developer for ReactOS Project
- Co-author of Windows Internals 5th and 6th Edition
- Founded Winsider Seminars & Solutions Inc., to provide services and Windows Internals training for enterprise/government
- Interned at Apple for a few years (Core Platform Team)
- Now Chief Architect at CrowdStrike

Introduction

Your Mac has a chip…

…that **anyone** can **update**…

…but you **can’t read it.**

It **manages** your light sensor…

…**protects** your disk…

…**stores** your FileVault key…

…has a “**Ninja timer**”…

…and has a **backdoor**…

…using a **Harry Potter spell**…

…all while **regulating** current and voltage

What is the SMC?

The System Management Controller I/O Chip

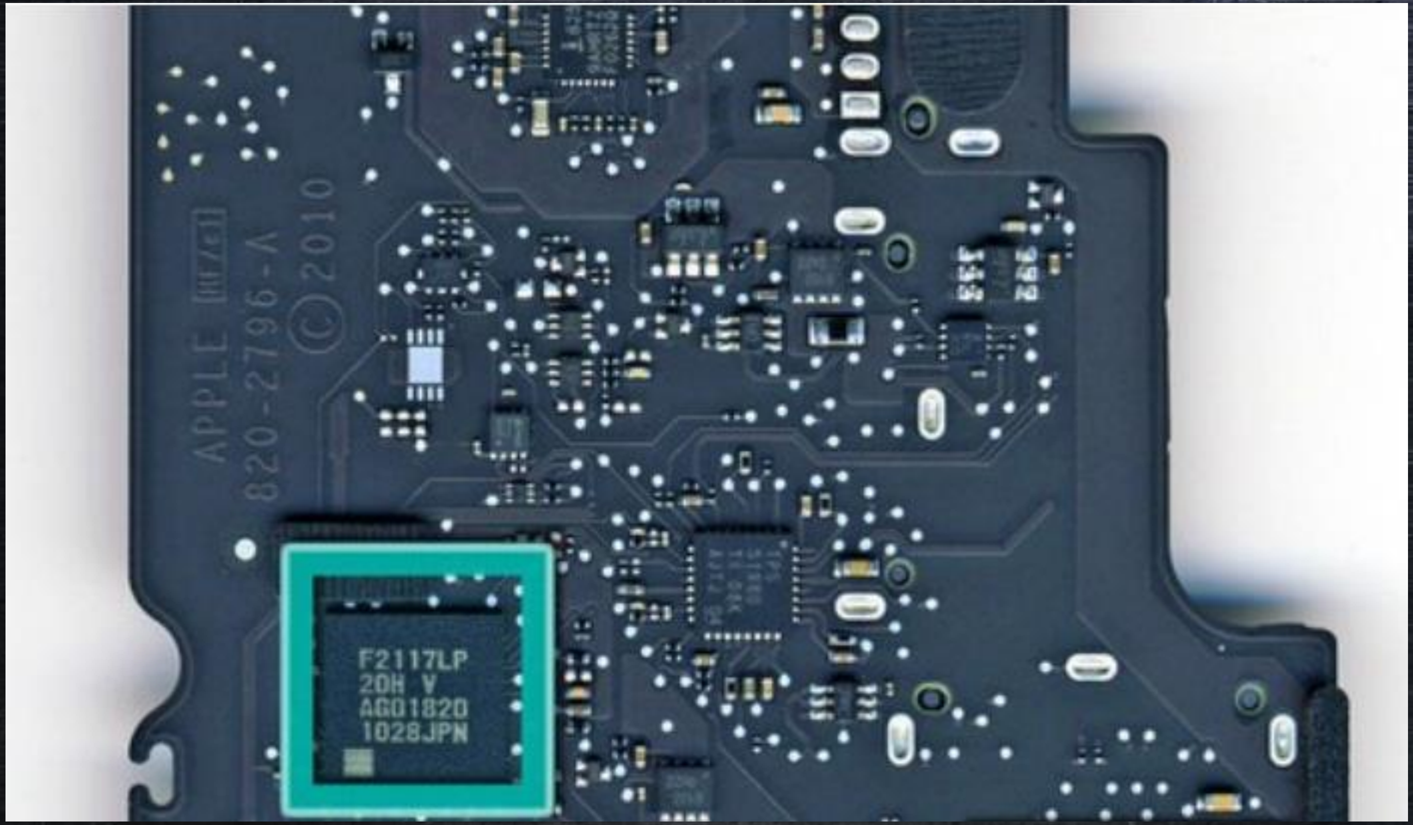
20MHz 16-bit Processor

8 32-bit General Purpose Registers

24-bit (16MB) Address Space

160K Flash ROM

8K RAM



Multiple Timers + Watchdog

I^2C Bus Access

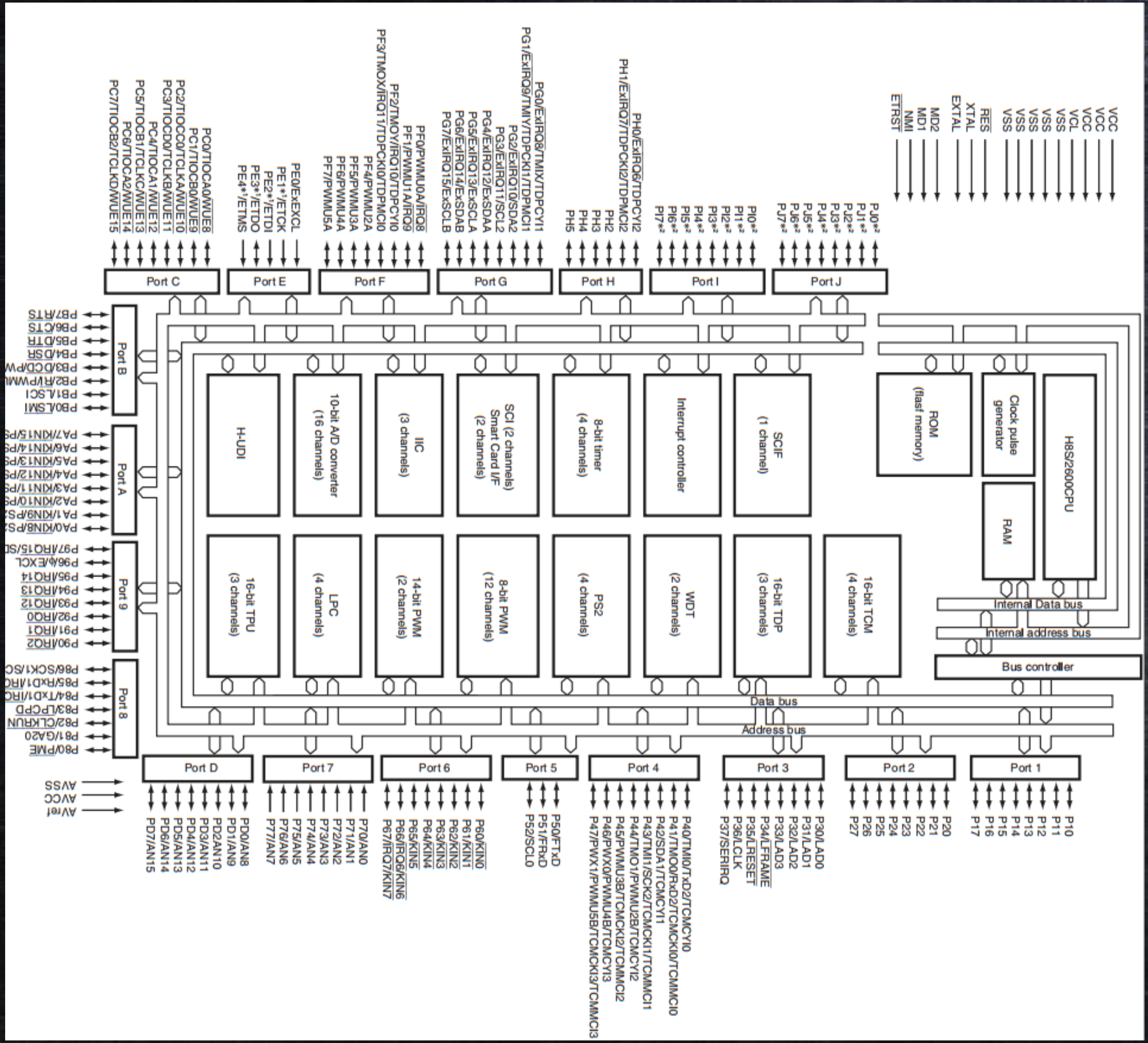
12-line Interrupt Controller

Analog/Digital Converter

LPC Bus Access, UART, USB, ACPI

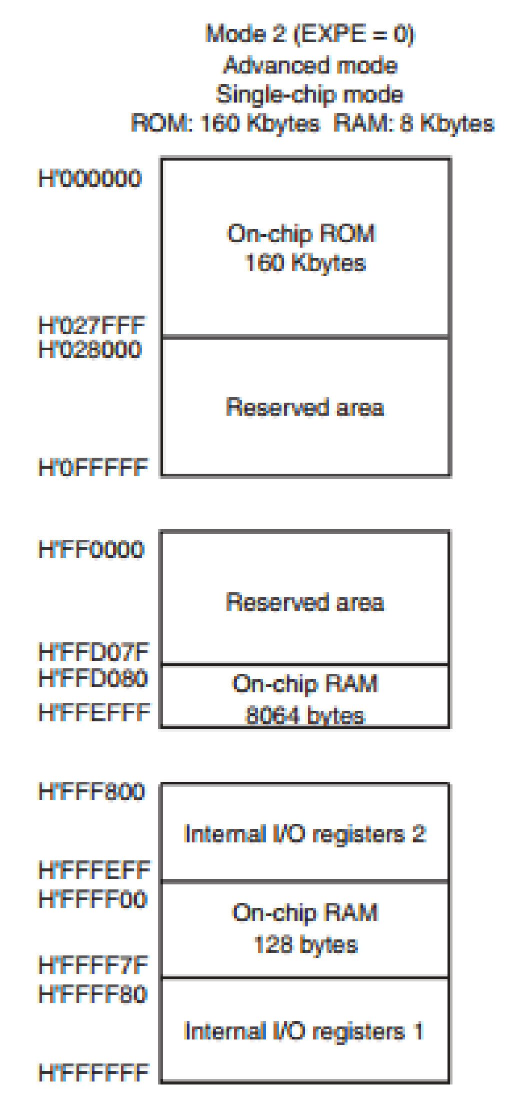
Various I/O Ports

The System Management Controller I/O Chip



SMC Address Map

- 0x000000-0x000FFF: Exception Vectors
- 0x001000-0x005FFF: Unknown/Unused
- 0x006000-0x006FFF: EPM UV Area
- 0x007000-0x007FFF: EPM CV Area
- 0x008000-0x022FFF: ROM Code + Data Variables
- 0x027FE0-0x027FFF: Code Markers (TBD)
- 0xFF2000-0xFF2FFF: Reserved (but used!)
- 0xFFF800-0xFFFEFF: I/O Registers
- 0xFFD080-0xFFEFFF: RAM (Data Variables)
- 0xFFFF00-0xFFFF7F: RAM (Used as Stack)
- 0xFFFF80-0xFFFFFF: I/O Registers



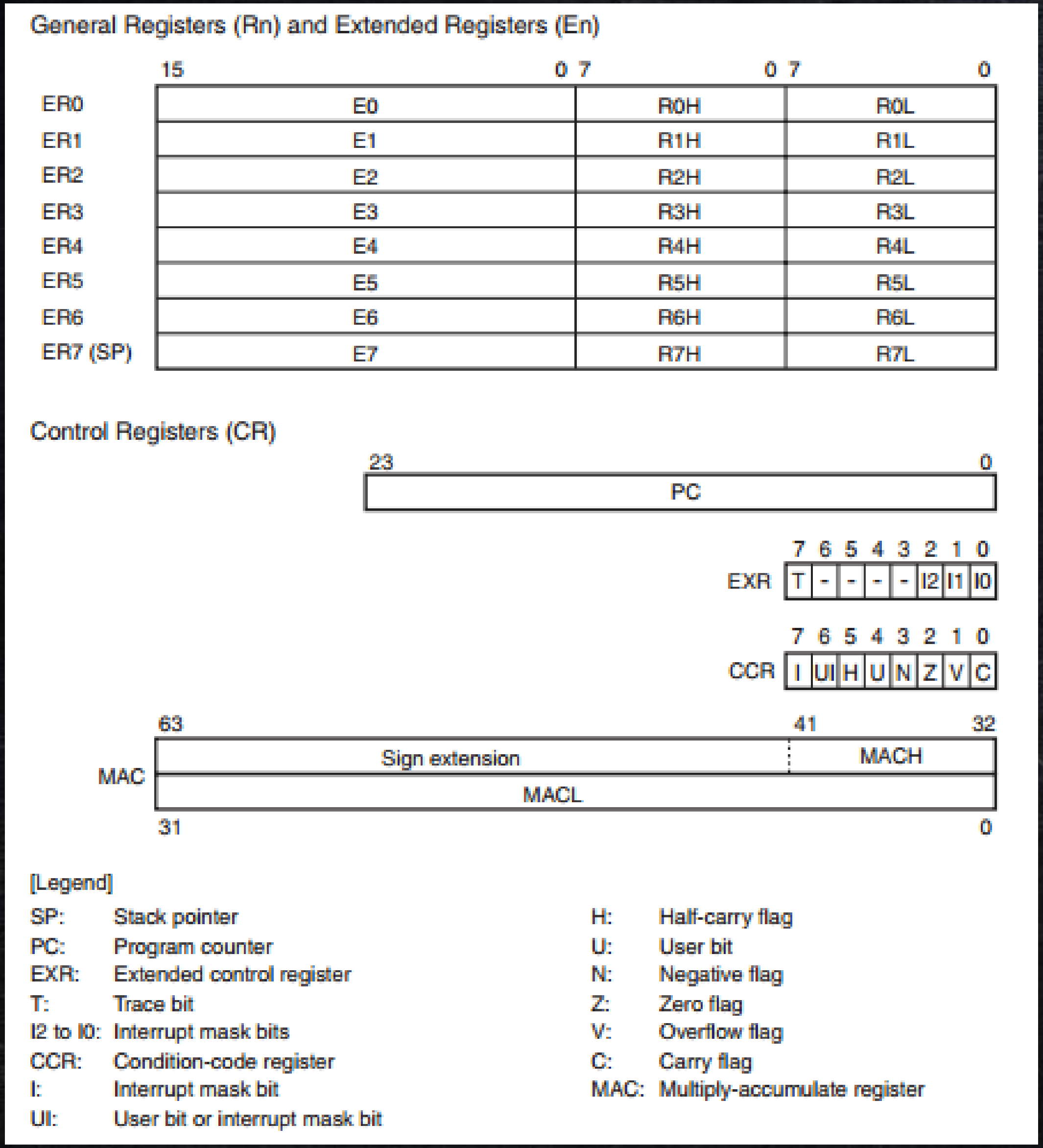
Renesas H8S/2117

- Full compiler support through GCC
  - Renesas also has development kit and free SDK available
- Used by many Intel Reference Platforms
  - Not just Apple – although this talk is only covering the Apple SMC
- Full 32-bit registers (er0-er7)
  - Access model similar to x86 (er0 -> e0 + r0h, r0l)

Renesas H8S/2117

- Different kinds of addressing modes
  - Absolute and relative, with various shifts and offsets
- Fully supported by IDA processor module
  - But IDA sometimes has trouble with references
- 69 instructions total
  - Complex data patterns hard to follow, but bit-instructions make I/O register access a breeze to understand

H8S/2117 Registers & Instructions



General Registers (Rn) and Extended Registers (En)

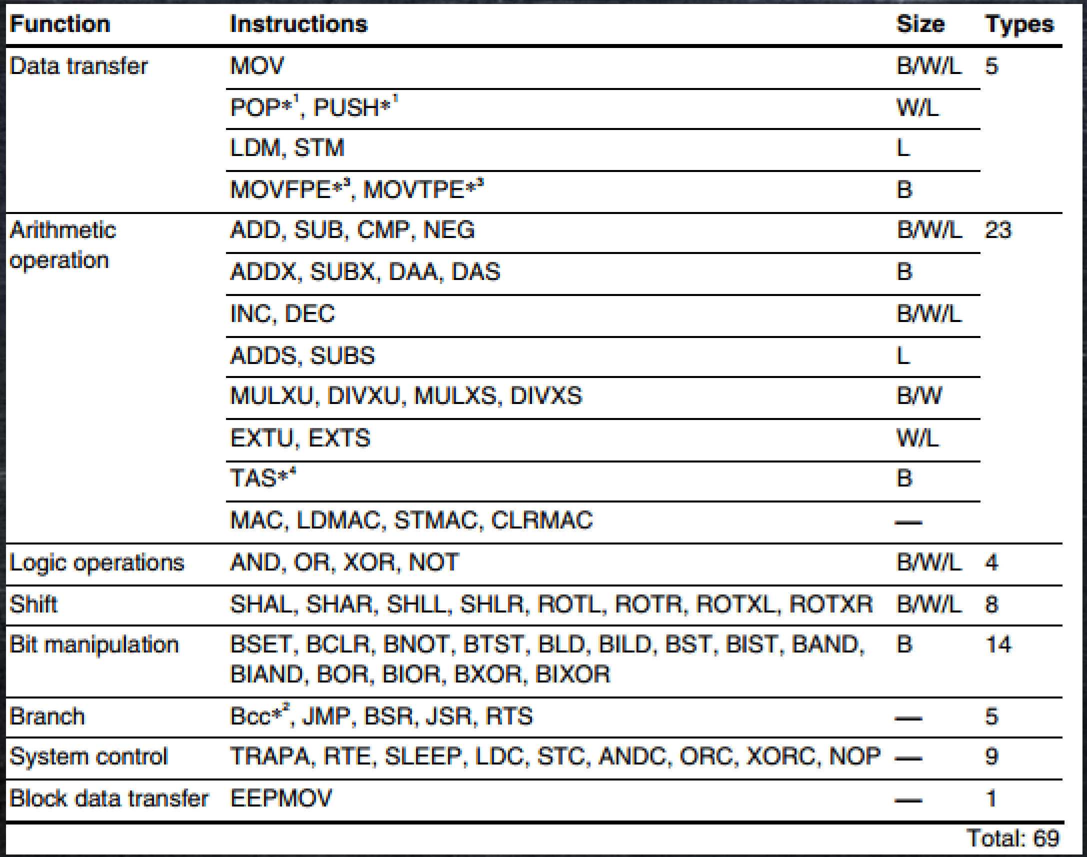
	15 … 0	7 … 0	7 … 0
ER0	E0	R0H	R0L
ER1	E1	R1H	R1L
ER2	E2	R2H	R2L
ER3	E3	R3H	R3L
ER4	E4	R4H	R4L
ER5	E5	R5H	R5L
ER6	E6	R6H	R6L
ER7 (SP)	E7	R7H	R7L

Control Registers (CR)

	7	6	5	4	3	2	1	0
EXR	T	-	-	-	-	I2	I1	I0
CCR	I	UI	H	U	N	Z	V	C

[Legend]

SP:	Stack pointer
PC:	Program counter
EXR:	Extended control register
T:	Trace bit
I2 to I0:	Interrupt mask bits
CCR:	Condition-code register
I:	Interrupt mask bit
UI:	User bit or interrupt mask bit
H:	Half-carry flag
U:	User bit
N:	Negative flag
Z:	Zero flag
V:	Overflow flag
C:	Carry flag
MAC:	Multiply-accumulate register



Function	Instructions	Size	Types
Data transfer	MOV	B/W/L	5
	POP*[1], PUSH*[1]	W/L	
	LDM, STM	L	
	MOVFPE*[3], MOVTPE*[3]	B	
Arithmetic operation	ADD, SUB, CMP, NEG	B/W/L	23
	ADDX, SUBX, DAA, DAS	B	
	INC, DEC	B/W/L	
	ADDS, SUBS	L	
	MULXU, DIVXU, MULXS, DIVXS	B/W	
	EXTU, EXTS	W/L	
	TAS*[4]	B	
	MAC, LDMAC, STMAC, CLRMAC	—	
Logic operations	AND, OR, XOR, NOT	B/W/L	4
Shift	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR	B/W/L	8
Bit manipulation	BSET, BCLR, BNOT, BTST, BLD, BILD, BST, BIST, BAND, BIAND, BOR, BIOR, BXOR, BIXOR	B	14
Branch	Bcc*[2], JMP, BSR, JSR, RTS	—	5
System control	TRAPA, RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP	—	9
Block data transfer	EEPMOV	—	1
			Total: 69

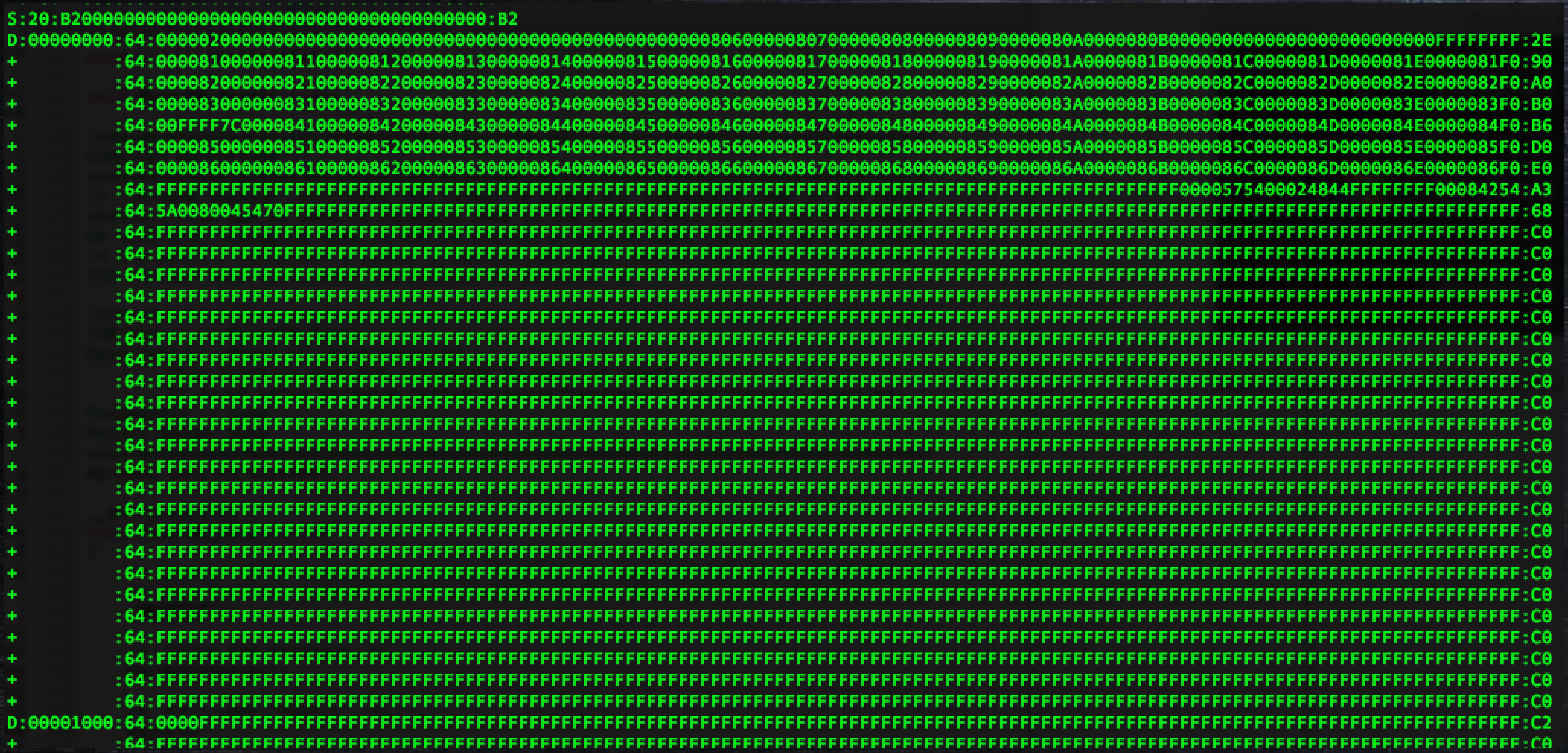
What’s in an SMC Update?

- Today’s SMC Updates are done through SMCFlasher.efi
  - Leverages AppleSMC.efi, which exposes the AppleSMCProtocol
  - SMCFlasher.efi is nothing but a renamed SMCUtil!
- SMCUtil is a long sought-after “Internal Apple Tool”
  - Can dump all sorts of SMC information
  - Change SMC Modes
  - Flash various portions of the SMC

SMC Update Payload

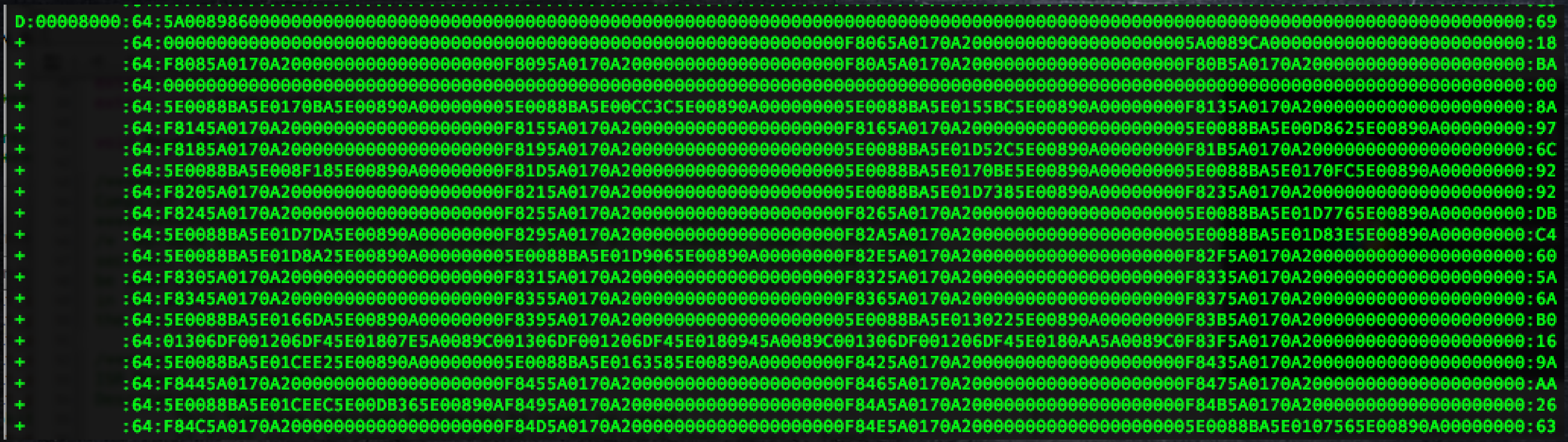
- SMCFlasher.efi takes a compressed payload as input
- Unusual S-REC-lookalike format, but no standard tools for it
  - Contains typical checksum byte for each 64-byte block
  - But also contains checksum vectors for the checksums themselves
- Wrote own tool to convert to binary image
  - Turns out, could’ve done it with grep (see presentation by Inverse Path)

Apple SMC Update Payload



```
S:20:B200000000000000000000000000000000000000:B2
D:00000000:64:0000020000000000000000000000000000000000000000000000080600000807000008080000080900000 80A0000080B000000000000000000000000FFFFFFFF:2E
+          :64:00008100000081100000812000008130000081400000815000008160000081700000818000008190000081A0000081B0000081C0000081D0000081E0000081F0:90
+          :64:00008200000082100000822000008230000082400000825000008260000082700000828000008290000082A0000082B0000082C0000082D0000082E0000082F0:A0
+          :64:00008300000083100000832000008330000083400000835000008360000083700000838000008390000083A0000083B0000083C0000083D0000083E0000083F0:B0
+          :64:00FFFF7C000084100000842000008430000084400000845000008460000084700000848000008490000084A0000084B0000084C0000084D0000084E0000084F0:B6
+          :64:00008500000085100000852000008530000085400000855000008560000085700000858000008590000085A0000085B0000085C0000085D0000085E0000085F0:D0
+          :64:00008600000086100000862000008630000086400000865000008660000086700000868000008690000086A0000086B0000086C0000086D0000086E0000086F0:E0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000575400024844FFFFFFFF00084254:A3
+          :64:5A0080045470FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:68
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
D:00001000:64:0000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C2
+          :64:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:C0
```

Apple SMC Update Payload



```
D:00008000:64:5A00898600000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000:69
+         :64:0000000000000000000000000000000000000000000000000000000000000000F8065A0170A2000000000000000000005A0089CA000000000000000000000000:18
+         :64:F8085A0170A200000000000000000000F8095A0170A200000000000000000000F80A5A0170A200000000000000000000F80B5A0170A200000000000000000000:BA
+         :64:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000:00
+         :64:5E0088BA5E0170BA5E00890A000000005E0088BA5E00CC3C5E00890A000000005E0088BA5E0155BC5E00890A00000000F8135A0170A200000000000000000000:8A
+         :64:F8145A0170A200000000000000000000F8155A0170A200000000000000000000F8165A0170A2000000000000000000005E0088BA5E00D8625E00890A00000000:97
+         :64:F8185A0170A200000000000000000000F8195A0170A2000000000000000000005E0088BA5E01D52C5E00890A00000000F81B5A0170A200000000000000000000:6C
+         :64:5E0088BA5E008F185E00890A00000000F81D5A0170A2000000000000000000005E0088BA5E0170BE5E00890A000000005E0088BA5E0170FC5E00890A00000000:92
+         :64:F8205A0170A200000000000000000000F8215A0170A2000000000000000000005E0088BA5E01D7385E00890A00000000F8235A0170A200000000000000000000:92
+         :64:F8245A0170A200000000000000000000F8255A0170A200000000000000000000F8265A0170A2000000000000000000005E0088BA5E01D7765E00890A00000000:DB
+         :64:5E0088BA5E01D7DA5E00890A00000000F8295A0170A200000000000000000000F82A5A0170A2000000000000000000005E0088BA5E01D83E5E00890A00000000:C4
+         :64:5E0088BA5E01D8A25E00890A000000005E0088BA5E01D9065E00890A00000000F82E5A0170A200000000000000000000F82F5A0170A200000000000000000000:60
+         :64:F8305A0170A200000000000000000000F8315A0170A200000000000000000000F8325A0170A200000000000000000000F8335A0170A200000000000000000000:5A
+         :64:F8345A0170A200000000000000000000F8355A0170A200000000000000000000F8365A0170A200000000000000000000F8375A0170A200000000000000000000:6A
+         :64:5E0088BA5E0166DA5E00890A00000000F8395A0170A2000000000000000000005E0088BA5E0130225E00890A00000000F83B5A0170A200000000000000000000:B0
+         :64:01306DF001206DF45E01807E5A0089C001306DF001206DF45E0180945A0089C001306DF001206DF45E0180AA5A0089C0F83F5A0170A200000000000000000000:16
+         :64:5E0088BA5E01CEE25E00890A000000005E0088BA5E0163585E00890A00000000F8425A0170A200000000000000000000F8435A0170A200000000000000000000:9A
+         :64:F8445A0170A200000000000000000000F8455A0170A200000000000000000000F8465A0170A200000000000000000000F8475A0170A200000000000000000000:AA
+         :64:5E0088BA5E01CEEC5E00DB365E00890AF8495A0170A200000000000000000000F84A5A0170A200000000000000000000F84B5A0170A200000000000000000000:26
+         :64:F84C5A0170A200000000000000000000F84D5A0170A200000000000000000000F84E5A0170A2000000000000000000005E0088BA5E0107565E00890A00000000:63
```

SMC ROM (0x00000-0x27FFF – 160KB)

- The SMC ROM code is called the User MAT by Renesas
  - It is considered the SMC “Application”, with a main()
  - It begins execution through the Reset Vector (0x0)
- The first ~KB is filled with the various Interrupt Vectors
  - Renesas Datasheet has all the internal/external interrupt nubmers
- Part of the chip’s responsibility is reacting to such interrupts
  - Timers, Watchdog, and ACPI + I/O Port (Accelerometer, I^{2}C)

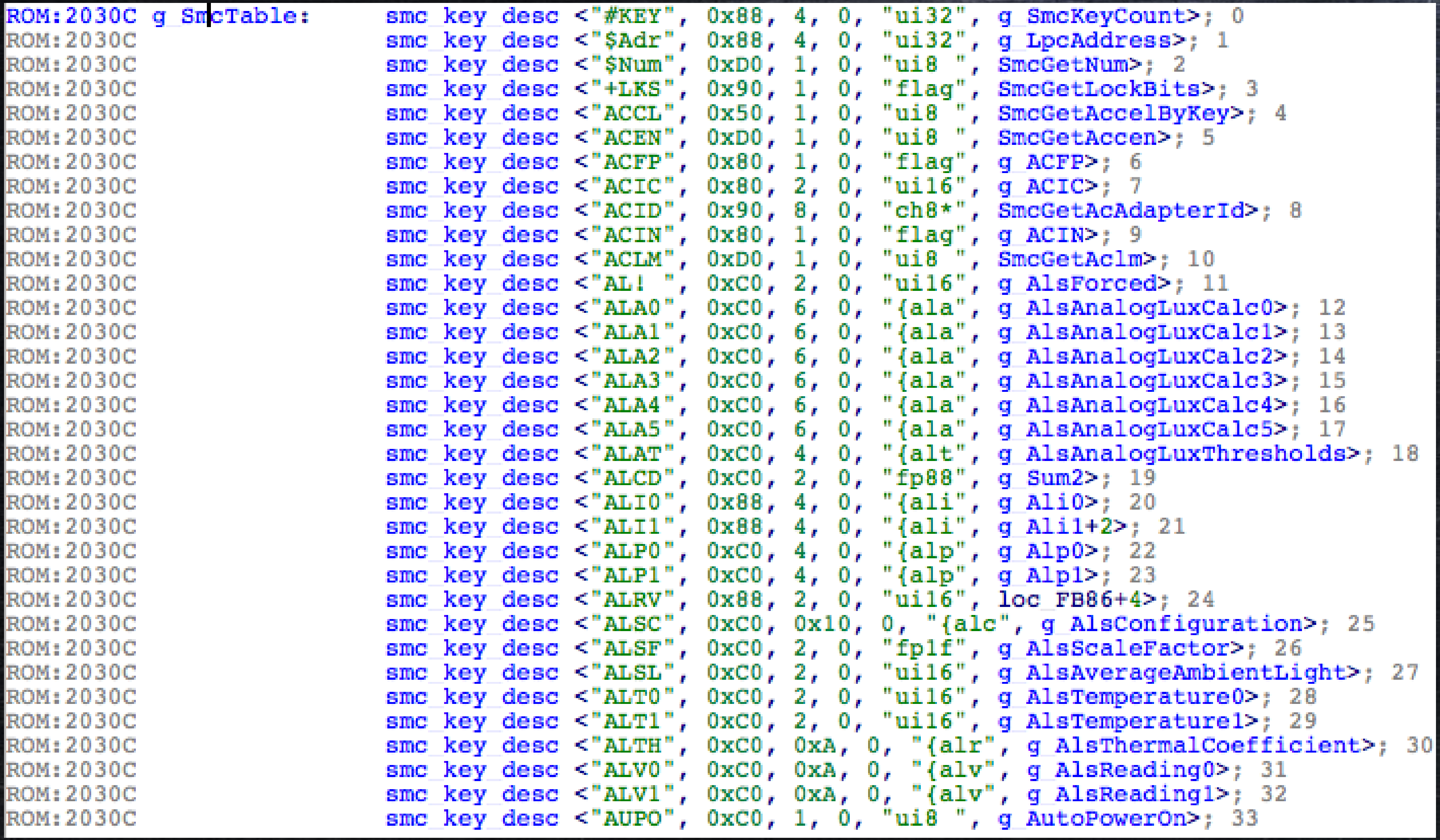
SMC ROM Code

- As external events cause interrupts, the SMC code updates state
  - Some of this state is internal, used in further interrupts for chained state
  - Some of this state is exposed back to the system through SMC “Keys”
- Likewise, interrupts can be generated by the SMC
  - Either on a regular basis, sending some piece of state to other hardware
  - Or on request (such as for UART or ACPI IF Notify Bytes)
  - The data can also be internal, or externalized through an SMC “Key”

SMC Key Mechanism

- Much of SMC functionality is done by read/write access to “keys”
  - 4-byte character tags describing some functionality
  - SMC Firmware has handlers for each key
  - Total keys = #SMCs * #Keys
    - Both of these are exposed through defined keys (TBD)
  - Key names can be enumerated
    - But all is not what it seems..

SMC Firmware Key Descriptors



```
ROM:2030C g_SmcTable:      smc_key_desc <"#KEY", 0x88, 4, 0, "ui32", g_SmcKeyCount>; 0
ROM:2030C                  smc_key_desc <"$Adr", 0x88, 4, 0, "ui32", g_LpcAddress>; 1
ROM:2030C                  smc_key_desc <"$Num", 0xD0, 1, 0, "ui8 ", SmcGetNum>; 2
ROM:2030C                  smc_key_desc <"+LKS", 0x90, 1, 0, "flag", SmcGetLockBits>; 3
ROM:2030C                  smc_key_desc <"ACCL", 0x50, 1, 0, "ui8 ", SmcGetAccelByKey>; 4
ROM:2030C                  smc_key_desc <"ACEN", 0xD0, 1, 0, "ui8 ", SmcGetAccen>; 5
ROM:2030C                  smc_key_desc <"ACFP", 0x80, 1, 0, "flag", g_ACFP>; 6
ROM:2030C                  smc_key_desc <"ACIC", 0x80, 2, 0, "ui16", g_ACIC>; 7
ROM:2030C                  smc_key_desc <"ACID", 0x90, 8, 0, "ch8*", SmcGetAcAdapterId>; 8
ROM:2030C                  smc_key_desc <"ACIN", 0x80, 1, 0, "flag", g_ACIN>; 9
ROM:2030C                  smc_key_desc <"ACLM", 0xD0, 1, 0, "ui8 ", SmcGetAclm>; 10
ROM:2030C                  smc_key_desc <"AL! ", 0xC0, 2, 0, "ui16", g_AlsForced>; 11
ROM:2030C                  smc_key_desc <"ALA0", 0xC0, 6, 0, "{ala", g_AlsAnalogLuxCalc0>; 12
ROM:2030C                  smc_key_desc <"ALA1", 0xC0, 6, 0, "{ala", g_AlsAnalogLuxCalc1>; 13
ROM:2030C                  smc_key_desc <"ALA2", 0xC0, 6, 0, "{ala", g_AlsAnalogLuxCalc2>; 14
ROM:2030C                  smc_key_desc <"ALA3", 0xC0, 6, 0, "{ala", g_AlsAnalogLuxCalc3>; 15
ROM:2030C                  smc_key_desc <"ALA4", 0xC0, 6, 0, "{ala", g_AlsAnalogLuxCalc4>; 16
ROM:2030C                  smc_key_desc <"ALA5", 0xC0, 6, 0, "{ala", g_AlsAnalogLuxCalc5>; 17
ROM:2030C                  smc_key_desc <"ALAT", 0xC0, 4, 0, "{alt", g_AlsAnalogLuxThresholds>; 18
ROM:2030C                  smc_key_desc <"ALCD", 0xC0, 2, 0, "fp88", g_Sum2>; 19
ROM:2030C                  smc_key_desc <"ALI0", 0x88, 4, 0, "{ali", g_Ali0>; 20
ROM:2030C                  smc_key_desc <"ALI1", 0x88, 4, 0, "{ali", g_Ali1+2>; 21
ROM:2030C                  smc_key_desc <"ALP0", 0xC0, 4, 0, "{alp", g_Alp0>; 22
ROM:2030C                  smc_key_desc <"ALP1", 0xC0, 4, 0, "{alp", g_Alp1>; 23
ROM:2030C                  smc_key_desc <"ALRV", 0x88, 2, 0, "ui16", loc_FB86+4>; 24
ROM:2030C                  smc_key_desc <"ALSC", 0xC0, 0x10, 0, "{alc", g_AlsConfiguration>; 25
ROM:2030C                  smc_key_desc <"ALSF", 0xC0, 2, 0, "fp1f", g_AlsScaleFactor>; 26
ROM:2030C                  smc_key_desc <"ALSL", 0xC0, 2, 0, "ui16", g_AlsAverageAmbientLight>; 27
ROM:2030C                  smc_key_desc <"ALT0", 0xC0, 2, 0, "ui16", g_AlsTemperature0>; 28
ROM:2030C                  smc_key_desc <"ALT1", 0xC0, 2, 0, "ui16", g_AlsTemperature1>; 29
ROM:2030C                  smc_key_desc <"ALTH", 0xC0, 0xA, 0, "{alr", g_AlsThermalCoefficient>; 30
ROM:2030C                  smc_key_desc <"ALV0", 0xC0, 0xA, 0, "{alv", g_AlsReading0>; 31
ROM:2030C                  smc_key_desc <"ALV1", 0xC0, 0xA, 0, "{alv", g_AlsReading1>; 32
ROM:2030C                  smc_key_desc <"AUPO", 0xC0, 1, 0, "ui8 ", g_AutoPowerOn>; 33
```

SMC Key Attributes

- SMC Keys have attributes, which are a combination of:
  - Read (0x80)
  - Write (0x40)
  - Function (0x10)
  - Const (0x8)
  - Private (0x1)
  - Atomic? (0x2)

SMC Key Example

- We can run functions in the SMC which return a result
  - SMC Functions receive a parameter in er0 which is 0x10 (R) or 0x11 (W)
  - Input and/or output buffers are in er1
- DEMO: As an example, take CRCB vs CRCU
  - CRCU causes a checksum to be taken of the entire UserMAT area
  - Useful to write this down somewhere and periodically check on it ;-)
  - Attacker could “fake” it however

Interesting SMC Keys

- 3rd party Apple Service Technician leaked old Apple SMC Key List
  - Outdated, and focused on desktop device, but contains many useful keys
- Reveals existence of a Ninja Action Timer
  - Can be programmed to fire at a certain time and take an action (i.e.: reboot)
- Reveals many keys related to power management & regulation, thermals, battery and adaptor data
  - DEMO: Controlling the fans manually

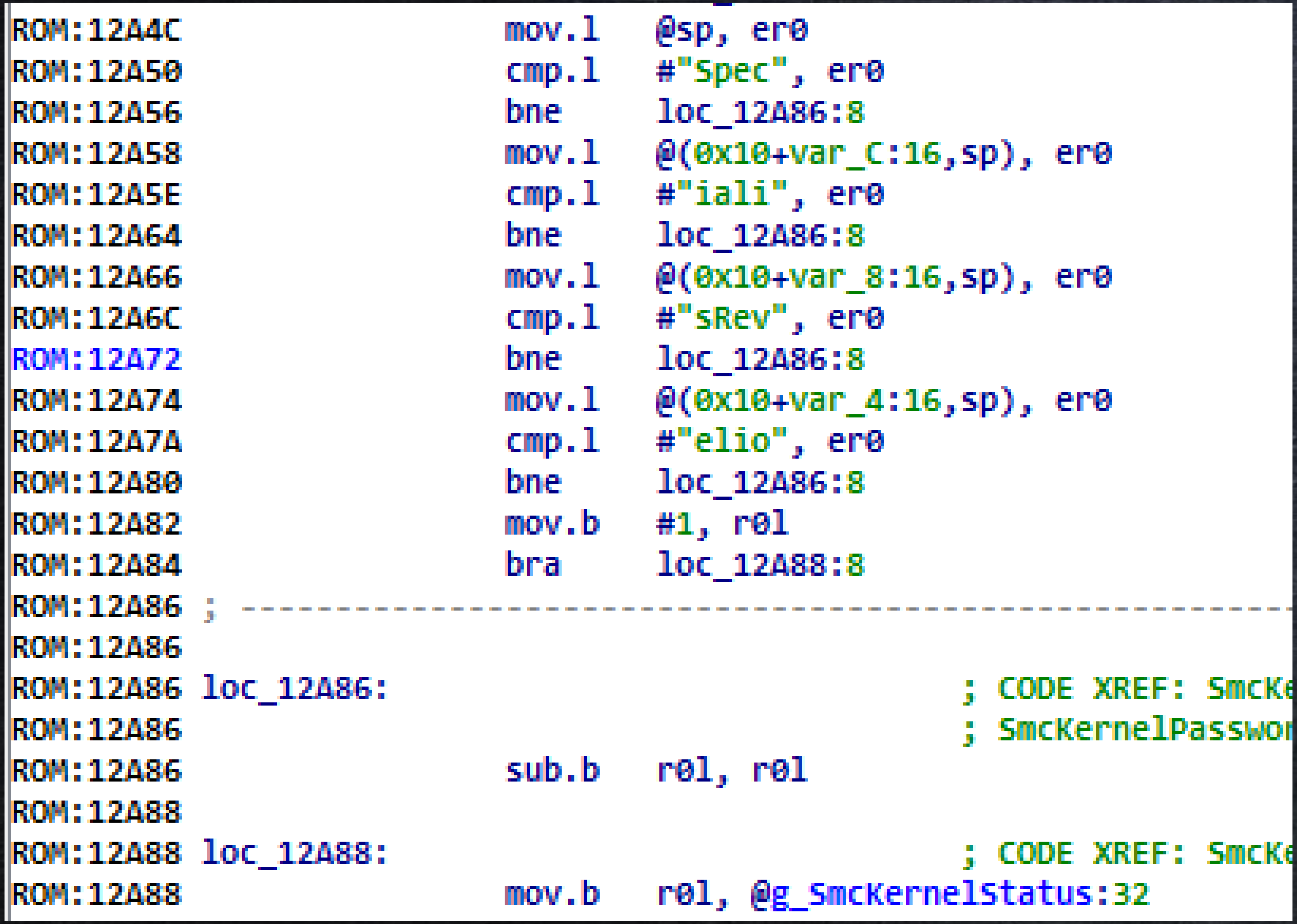
More Interesting SMC Keys…

- The last two keys enumerated by the SMC are OSK0 and OSK1
  - Names suggest “Operating System Key 0, 1”
  - Large data blobs (32-characters), suggestive indeed of cryptographic keys
- DEMO: Let’s dump the keys
- There’s actually a very good reason for having keys as English
  - Any lawyers in the room? ☺

Really Interesting SMC Keys…

- By using IDA to dump the list of keys, a discrepancy is noted!
  - There are two more keys that are *not* officially listed
    - In fact a function (*smcManageBackdoor* in my IDB) is responsible for patching the table
- The two mystery keys are KPPW and KPST
  - Kernel Protection Password, Kernel Protection Status?
- KPST returns the variable (g_KernelProtectionStatus)
  - Set to 1 if KPPW suceeds

How to make KPPW Succeed?



```
ROM:12A4C                     mov.l   @sp, er0
ROM:12A50                     cmp.l   #"Spec", er0
ROM:12A56                     bne     loc_12A86:8
ROM:12A58                     mov.l   @(0x10+var_C:16,sp), er0
ROM:12A5E                     cmp.l   #"iali", er0
ROM:12A64                     bne     loc_12A86:8
ROM:12A66                     mov.l   @(0x10+var_8:16,sp), er0
ROM:12A6C                     cmp.l   #"sRev", er0
ROM:12A72                     bne     loc_12A86:8
ROM:12A74                     mov.l   @(0x10+var_4:16,sp), er0
ROM:12A7A                     cmp.l   #"elio", er0
ROM:12A80                     bne     loc_12A86:8
ROM:12A82                     mov.b   #1, r0l
ROM:12A84                     bra     loc_12A88:8
ROM:12A86 ; ---------------------------------------------------------------
ROM:12A86
ROM:12A86 loc_12A86:                                      ; CODE XREF: SmcKe
ROM:12A86                                                 ; SmcKernelPasswor
ROM:12A86                     sub.b   r0l, r0l
ROM:12A88
ROM:12A88 loc_12A88:                                      ; CODE XREF: SmcKe
ROM:12A88                     mov.b   r0l, @g_SmcKernelStatus:32
```

Requires input buffer to be “SpecialisRevelio”

Wait… seriously?

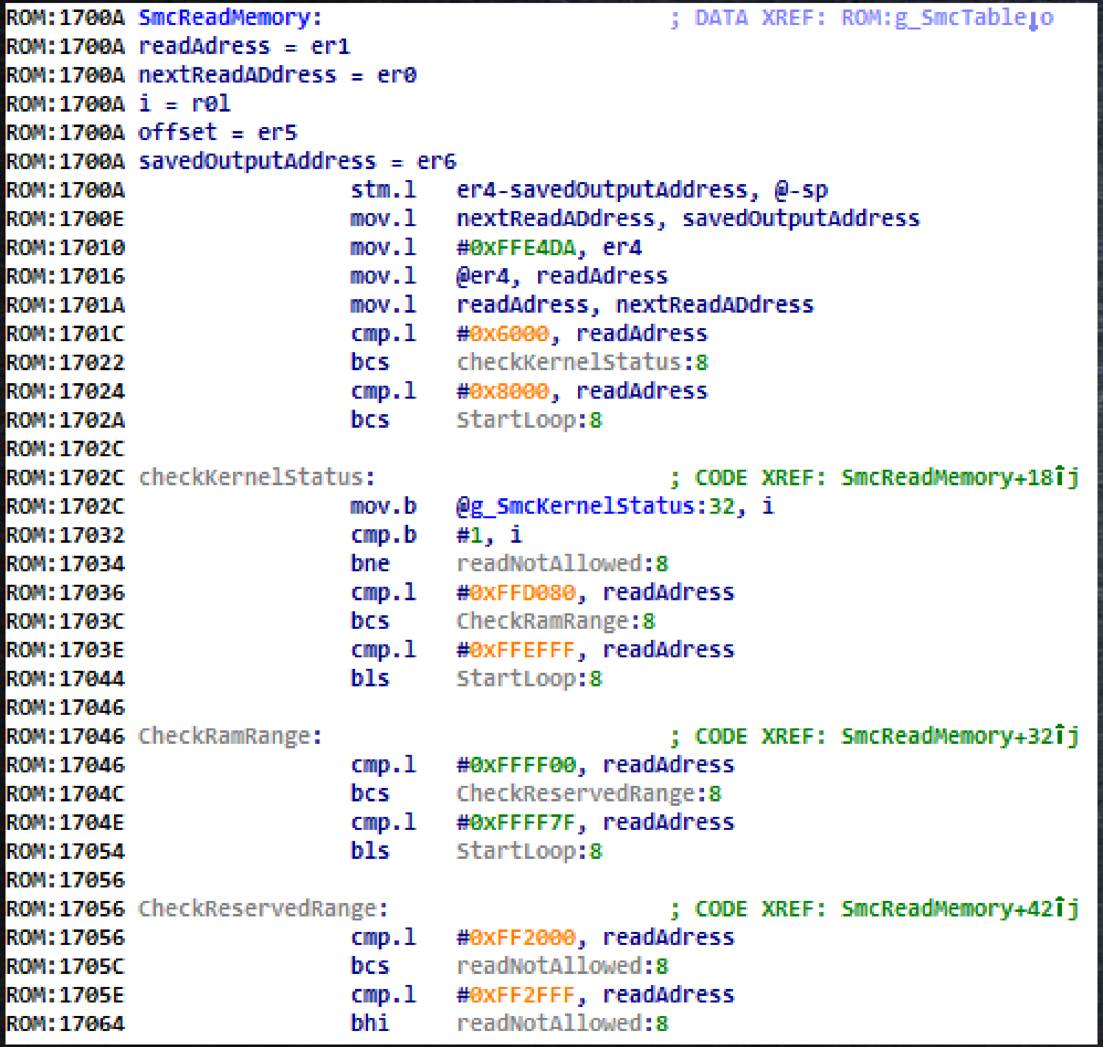
- http://harrypotter.wikia.com/wiki/Scarpin's_Revelaspell
- **Scarpin's Revelaspell** (*Specialis Revelio*) is a charm that is used to reveal charms and hexes that have been cast onto a target[1]. It can also, however, be used to reveal the ingredients of a potion.

- http://en.wikipedia.org/wiki/List_of_spells_in_Harry_Potter#Specialis_Revelio_.28Scarpin.27s_Revelaspell.29
- **Description:** Causes an object to show its hidden secrets or magical properties.
- **Seen/mentioned:** Used by Hermione to find out more of Harry's Advanced Potion-Making book in *Half-Blood Prince*. Used by Ernie Macmillan to find out the ingredients of a potion.

Memory Address Cycle (MAC)

- Three keys allow reading the SMC!
  - MACA: Sets the address in the SMC to read
  - MACM: Auto-incrementing addressing or manual-MACA addressing
  - MACR: Returns 32-bits from MACA, increments if MACM set
- But “restricted to EPM range”
  - This is where the mystery “Kernel Status” comes in

Effect of SmcKernelStatus == 1



```
ROM:1700A SmcReadMemory:                         ; DATA XREF: ROM:g_SmcTable↓o
ROM:1700A readAdress = er1
ROM:1700A nextReadADdress = er0
ROM:1700A i = r0l
ROM:1700A offset = er5
ROM:1700A savedOutputAddress = er6
ROM:1700A                 stm.l   er4-savedOutputAddress, @-sp
ROM:1700E                 mov.l   nextReadADdress, savedOutputAddress
ROM:17010                 mov.l   #0xFFE4DA, er4
ROM:17016                 mov.l   @er4, readAdress
ROM:1701A                 mov.l   readAdress, nextReadADdress
ROM:1701C                 cmp.l   #0x6000, readAdress
ROM:17022                 bcs     checkKernelStatus:8
ROM:17024                 cmp.l   #0x8000, readAdress
ROM:1702A                 bcs     StartLoop:8
ROM:1702C
ROM:1702C checkKernelStatus:                     ; CODE XREF: SmcReadMemory+18↑j
ROM:1702C                 mov.b   @g_SmcKernelStatus:32, i
ROM:17032                 cmp.b   #1, i
ROM:17034                 bne     readNotAllowed:8
ROM:17036                 cmp.l   #0xFFD080, readAdress
ROM:1703C                 bcs     CheckRamRange:8
ROM:1703E                 cmp.l   #0xFFEFFF, readAdress
ROM:17044                 bls     StartLoop:8
ROM:17046
ROM:17046 CheckRamRange:                         ; CODE XREF: SmcReadMemory+32↑j
ROM:17046                 cmp.l   #0xFFFF00, readAdress
ROM:1704C                 bcs     CheckReservedRange:8
ROM:1704E                 cmp.l   #0xFFFF7F, readAdress
ROM:17054                 bls     StartLoop:8
ROM:17056
ROM:17056 CheckReservedRange:                    ; CODE XREF: SmcReadMemory+42↑j
ROM:17056                 cmp.l   #0xFF2000, readAdress
ROM:1705C                 bcs     readNotAllowed:8
ROM:1705E                 cmp.l   #0xFF2FFF, readAdress
ROM:17064                 bhi     readNotAllowed:8
```

Allows reading RAM, Stack, and FF2000 “Reserved” Region
ROM Reads still not allowed ☹

SMC Kernel Extension (AppleSMC.kext)

Kernel Extension

- Manages SMC Runtime Support
  - Interrupts from SMC
  - Notifications to SMC
- Implements IOUserClient
  - Allows read (non-privileged) and write (privileged) to SMC Keys
  - Allows other special commands (ACPI Notify, more…)

SMC Interrupts

- Five interrupts are configured in the SMC
  - sms-shock-int (Detection of sudden disk shock, causes Disk Head Park)
  - sms-drop-int (Same as above)
  - sms-orientation-int (Change in orientation)
  - als-change-int (Change in ambient lighting)
  - EmergencyHeadPark (Again, related to disk head parking)

SMC Notifications

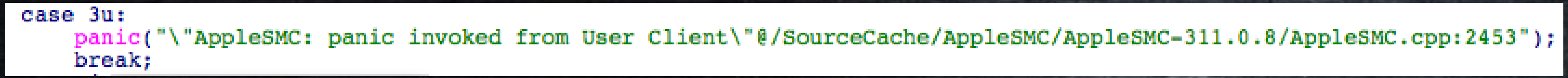
- SMC can also be notified with *IoRegistryEntrySetCFProperty*
  - “TheTimesAreAChangin”
    - Sets SMC ‘CLKT’ and ‘CLKH’
- Also supports Mach Message Notification (0xE0078000)
  - Sets SMC ‘RAID’ value to 1
- Power State Change Callback (0xE000031)
  - Sets SMC ‘MSDW’ key to zero

SMC KEXT User-Mode Client Access

- *IOServiceGetMatchingService*(“AppleSMC”)
- *IoConnectCallMethod*(kSMCUserClientOpen/kSMCUserClientClose)
- *IoConnectCallMethod*(kSMCHandleYPCEvent)
  - kSMCReadKey, kSMCWriteKey
  - kSMCGetKeyCount, kSMCGetKeyFromIndex, kSMCGetKeyInfo
  - kSMCReadStatus, kSMCReadResult
  - kSMCGetPLimits, kSMCFireInterrupt, kSMCGetVers

SMC KEXT “Variable Commands”

- kSMCVariableCommand provides interesting access
  - 1: Writes LAtN with user input (ACPI Proprietary IF Notify)
  - 2: Sets SMC System Type
  - 3: Panics the machine!



```
case 3u:
    panic("\"AppleSMC: panic invoked from User Client\"@/SourceCache/AppleSMC/AppleSMC-311.0.8/AppleSMC.cpp:2453");
    break;
```

  - 4: Sets Watchdog Timer
  - 5: Dumps Notifications
  - 6: Sets SMC Sleep State

SMC Errors (Shared in Firmware + KEXT)

- kSMCCommCollision = -80
- kSMCSpuriousData = -7F
- kSMCBadCommand = -7E
- kSMCBadParameter = -7D
- kSMCKeyNotFound = -7C
- kSMCKeyNotReadable = -7B
- kSMCKeyNotWritable = -7A
- kSMCKeySizeMismatch = -79
- kSMCFramingError = -78
- kSMCBadArgumentError = -77
- kSMCTimeoutError = -49
- kSMCKeyIndexRangeError = -48
- kSMCBadFuncParameter = -40
- kSMCDeviceAccessError = -39
- kSMCUnsupportedFeature = -35
- kSMCSMBAccessError = -34

Conclusion

Key Takeaways

- The Apple SMC is a treasure trove of undocumented mechanisms
  - Probably partly responsible for power & thermal efficiency
- The AppleSMC KEXT opens up interesting non-admin possibilities for SMC access
  - But most holes plugged in Mountain Lion
- The OS, EFI, and ACPI, all contain code to work with the SMC
- Anyone can flash the SMC, but nobody can (easily) read it

Future Work

- Reverse engineered 100% of the AppleSMC KEXT for Lion
  - Working on updating it for Mountain Lion Support
  - There are also other KEXTs, such as the SMC Platform Plugin
  - Would *like* to release it, but most interest around SMC is related to piracy/cloning of OS X, and do not want to condone that
- Reverse engineered 30% of the Apple SMC firmware
  - Still don’t understand what EPM UV/CV areas are
  - Lots of behaviors still misunderstood / not yet understood

QA

- Greetz/shouts to: msuiche, Andrea Barisani, Daniele Bianco
- See you at Recon!

CROWDSTRIKE