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
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^2^C 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!)
- 0xFFFF800-0xFFFFEFF: I/O Registers
- 0xFFFFD080-0xFFFFEFF: RAM (Data Variables)
- 0xFFFFF00-0xFFFF7F: RAM (Used as Stack)
- 0xFFFFF80-0xFFFFFFFF: 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)
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
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)
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, I²C)
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

<table>
<thead>
<tr>
<th>ROM:2030C g SmpTable:</th>
<th>smc key desc &quot;#KEY&quot;, 0x88, 4, 0, &quot;ui32&quot;, g SmcKeyCount&gt;; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;$Adr&quot;, 0x88, 4, 0, &quot;ui32&quot;, g LpcAddress&gt;; 1</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;$Num&quot;, 0xFD, 1, 0, &quot;ui8&quot; g SmcGetNum&gt;; 2</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;+Lks&quot;, 0x90, 1, 0, &quot;flag&quot; g SmcGetLockBits&gt;; 3</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;ACCL&quot;, 0x50, 1, 0, &quot;ui8&quot; g SmcGetAccelByKey&gt;; 4</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;ACFP&quot;, 0x80, 1, 0, &quot;flag&quot; g ACFP&gt;; 6</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;ACIC&quot;, 0x80, 2, 0, &quot;ui16&quot; g ACIC&gt;; 7</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;ACID&quot;, 0x90, 8, 0, &quot;ch8&quot; g SmcGetAdapterId&gt;; 8</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;ACIN&quot;, 0x80, 1, 0, &quot;flag&quot; g ACIN&gt;; 9</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;ACL&quot;, 0x00, 1, 0, &quot;ui8&quot; g SmcGetAcl&gt;; 10</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL1&quot;, 0x00, 2, 0, &quot;ui16&quot; g Al1Forced&gt;; 11</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL2&quot;, 0x00, 6, 0, &quot;ala&quot; g AlsAnalogLuxCalc0&gt;; 12</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL3&quot;, 0x00, 6, 0, &quot;ala&quot; g AlsAnalogLuxCalc1&gt;; 13</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL4&quot;, 0x00, 6, 0, &quot;ala&quot; g AlsAnalogLuxCalc2&gt;; 14</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL5&quot;, 0x00, 6, 0, &quot;ala&quot; g AlsAnalogLuxCalc3&gt;; 15</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL6&quot;, 0x00, 6, 0, &quot;ala&quot; g AlsAnalogLuxCalc4&gt;; 16</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL7&quot;, 0x00, 4, 0, &quot;alp&quot; g AlsAnalogLuxThreshold&gt;; 18</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL8&quot;, 0x00, 4, 0, &quot;alp&quot; g AlsAnalogLuxThreshold&gt;; 19</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL9&quot;, 0x88, 4, 0, &quot;alp&quot; g All0&gt;; 20</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL10&quot;, 0x88, 4, 0, &quot;alp&quot; g All1&gt;; 21</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL11&quot;, 0x00, 4, 0, &quot;alp&quot; g All1&gt;; 22</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL12&quot;, 0x00, 4, 0, &quot;alp&quot; g Alp0&gt;; 23</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL13&quot;, 0x00, 4, 0, &quot;alp&quot; g Alp1&gt;; 24</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL14&quot;, 0x88, 2, 0, &quot;ui16&quot; g SmcGetAcl&gt;; 25</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL15&quot;, 0x00, 0x10, 0, &quot;{alc}&quot;, g AlsConfiguration&gt;; 26</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL16&quot;, 0x00, 2, 0, &quot;fplic&quot; g AlsScaleFactor&gt;; 27</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL17&quot;, 0x00, 2, 0, &quot;ui16&quot; g AlsAverageAmbientLight&gt;; 28</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL18&quot;, 0x00, 2, 0, &quot;ui16&quot; g AlsTemperature&gt;; 29</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL19&quot;, 0x00, 2, 0, &quot;ui16&quot; g AlsTemperature1&gt;; 30</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL20&quot;, 0x00, 0xA, 0, &quot;alr&quot; g AlsThermalCoefficient&gt;; 31</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL21&quot;, 0x00, 0xA, 0, &quot;alv&quot; g AlsReadin0&gt;; 32</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AL22&quot;, 0x00, 0xA, 0, &quot;alv&quot; g AlsReadin1&gt;; 33</td>
</tr>
<tr>
<td>ROM:2030C</td>
<td>smc key desc &quot;AUPO&quot;, 0x00, 1, 0, &quot;ui8&quot;, g AutoPowerOn&gt;; 34</td>
</tr>
</tbody>
</table>
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

- 3\(^{rd}\) 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 succeeds
How to make KPPW Succeed?

Requires input buffer to be “SpecialisRevelio”
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.

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

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!

- 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
- `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!