The Exploration and Exploitation of an SD Memory Card

bunnie & xobs
30c3
Origin: Searching for Fakes
Card Teardowns
Solution: managed Flash

- Small embedded controller in every “managed Flash” device
  - 8051 or ARM7 CPU
  - 4-8mm^2 silicon = ~$0.15-$0.30 cost add-on
  - Compare to Flash die area = 100mm^2, $2.90 cost
  - Compare to test cost, wafer-scale tester = $1mm = ~$0.45 for a 30 second test (assuming 24 month lifetime and usage 24x7x365)
Faking Reliability

- Flash memory is “unreliable”
  - You are not storing data, you are storing probabilistic approximation of your data
  - Workaround: computational error correction (ECC)
Also, Bad Blocks

- TLC/MLC Flash price is < 0.1nano$/bit
  - Only achievable because every piece of silicon fabricated is sold, regardless of fabrication errors – nothing is thrown away
  - Work around: bad block remapping
    - In some cases, over 80% of blocks are bad (e.g. 16GiB chip sold as 2GiB)
    - Also, blocks go bad with P/E cycles

[xeltek]
[theregister.co.uk]
Why do it at this layer?

<table>
<thead>
<tr>
<th>Rainbow tables</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OS</td>
</tr>
<tr>
<td></td>
<td>Bus controller</td>
</tr>
<tr>
<td></td>
<td>Device controller</td>
</tr>
<tr>
<td></td>
<td>Raw Flash</td>
</tr>
</tbody>
</table>

- JFFS, YAFFS
- SSD, SD, USB mass storage

- Considering:
  - Flash geometry changes every 12-18 mos
    - New ECC rules
    - New page size, block mapping
    - Intensely cost-sensitive market
    - Lowest cost, highest performing Flash chips are proprietary
The Concern

- This is the setup for a MITM attack
- What runs on the microcontroller? Can it be hacked? Can I trust my Flash memory?
Fakes Turn In; New Quest: Hack an SD Card

• Find and hack an SD card
  – Control the micro to make an LED flash, at a bare minimum
  – Challenge: no public docs available on controllers

• Our story
  – Hardware tools developed to inspect, learn, and hack SD cards
  – Software tools and static code analysis to discover back doors and controller structure
Step 1: Acquire targets
SD Cards Ahoy
Card Survey
What's inside
Easy mode decap
Taps: gen 1 monolithic
Taps Gen2

- This header connects to the analysis computer.
- This region solders onto the T80P flash pads, allowing for ROM emulation and logging.
- This region solders onto the SD card connector pins, allowing for injection of command stimuli.
Taps: gen 2, monolithic and discrete
Tap in-system
• Capabilities:
  – Flash ROM emulation
    • DDR3 as Flash
    • Dual-port implementation, mod and read on the fly
  – Interface logging
    • Trace capture of SD and Flash interface transactions
ROM reader
Identifying a target

- Discrete implementation – more hacking options than monolithic
- SLC memory (unscrambled, trivially readable)
  - Easy to check for strings:
    “China Buildwin SD Controller, Anti-Japig, Author: Y/G/S/P/X Date: 2008-7”
  - Cross-check against google → Appotech controller, likely 8051
    - AX211
Factory Firmware

• Initial code had to get there somehow
  – Try to get ahold of the factory's flashing tool
Obtaining software
Obtaining software
Programming tool
Strange filenames
About the 8051
About the 8051

```
dd if=/dev/urandom of=firmware.bin bs=2048 count=1
```
## About the 8051

### Instructions by opcode

<table>
<thead>
<tr>
<th>Opcode (Hex)</th>
<th>NOP</th>
<th>AJMP</th>
<th>LIMP</th>
<th>RR</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
<th>INC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>JBC</td>
<td>ACALL</td>
<td>LCALL</td>
<td>RRC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
</tr>
<tr>
<td>0x20</td>
<td>JB</td>
<td>AJMP</td>
<td>RET</td>
<td>RL</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
</tr>
<tr>
<td>0x30</td>
<td>JNB</td>
<td>ACALL</td>
<td>RETI</td>
<td>RLC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
</tr>
<tr>
<td>0x40</td>
<td>JC</td>
<td>AJMP</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
</tr>
<tr>
<td>0x50</td>
<td>INC</td>
<td>ACALL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
</tr>
<tr>
<td>0x60</td>
<td>JZ</td>
<td>AJMP</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
</tr>
<tr>
<td>0x70</td>
<td>INZ</td>
<td>ACALL</td>
<td>ORL</td>
<td>INP</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>0x80</td>
<td>JMP</td>
<td>AJMP</td>
<td>ANL</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>0x90</td>
<td>MOV</td>
<td>ACALL</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>0xa0</td>
<td>ORL</td>
<td>AJMP</td>
<td>MOV</td>
<td>INC</td>
<td>MUL</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>0xb0</td>
<td>ANL</td>
<td>ACALL</td>
<td>CPL</td>
<td>CPL</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
<td>CINE</td>
</tr>
<tr>
<td>0xc0</td>
<td>PUSH</td>
<td>AJMP</td>
<td>CLR</td>
<td>CLR</td>
<td>SWAP</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
</tr>
<tr>
<td>0xd0</td>
<td>POP</td>
<td>ACALL</td>
<td>SETD</td>
<td>SETD</td>
<td>DA</td>
<td>Dinz</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
</tr>
<tr>
<td>0xe0</td>
<td>MOVX</td>
<td>AJMP</td>
<td>MOVX</td>
<td>MOVX</td>
<td>CLR</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>0xff</td>
<td>MOVX</td>
<td>ACALL</td>
<td>MOVX</td>
<td>MOVX</td>
<td>CPL</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
</tbody>
</table>

About the 8051

About the AX211

AX211 阿方索 32GB SSD存储器

CPU 特性
- 采用ARM926 800MHz CPU，优化支持SD、NAND接口
- 最大256MB的容量，批量IC 成品率99%

SD接口特性
- 完全符合SD 1.2A功能
- 功耗控制水平
- 支持SPI接口
- 兼容工业级温度范围-40°C～85°C
- 工作电压3.3V/5V

NAND Flash 接口特性
- 支持B0及大容量NAND flash
- 支持256MB/512MB四种规格

CPU Features
- 1T/256-/512-MHz CPU, supported for SD, NAND Flash systems
- 32-bit 86 MHz performance with on-chip RC oscillator and PLL

SD Interface Features
- Fully supports SD standard 1.0A/1.2A
- Supports SD high-capacity standard
- Supports SPI mode
- Supports Command mode A1/4/5/6/7/9/10
- Supports CRC8
- Supports data size up to 8MB
- Supports fast clock up to 5MHz
- Supports Max Width X3/X6
- Supports 500 ms host function
- Enhanced CID protection

NAND Flash Interface Features
- Supports single-chip NAND flash
- Supports BL0 or WLC NAND flash

www.appotech.com
www.buildwin.com.cn
About the AX211

CPU Features
- 1T 32-Bit RISC CPU, optimized for SD, NAND Flash applications
- MAX 50 MIPS performance with on chip RC oscillator and PLL

SD Interface Features
- Fully Supports SD card standard 1.0/1.1/2.0
- Supports SD high capacity standard
- Supports SPI mode
- Supports Command class 0/2/4/5/6/7/8/10
- Supports CPRM
- Speed class up to 6
- Supports host clock up to 50MHz
- Supports bus width X1/X4
- Supports SD host function
- Enhanced ESD protection

NAND Flash Interface Features
- Supports small or large page NAND flash
- Supports SLC or MLC NAND flash

- Supports Two-Plane or Interleave NAND flash
- Supports X8/X16 NAND flash
- Supports parallel mode
- Supports up to 8 CE
- Built in 6-54bit/page(528 bytes) on-the-fly ECC
- Data protection during data transfer if unplugged /power off

Low Power Consumption
- Operating current is about 10mA during data transfer
- Supports Sleep Mode, current is less than 200uA during Sleep Mode
- Fast wake up during Sleep Mode

Package
- 48-pin TQFP or QFN package
- Die form
# About the AX211

## CPU Features
- 1T 32-Bit RISC CPU, optimized for SD, NAND Flash applications
- Low 50 MIPS performance with on-chip RC oscillator and PLL

## SD Interface Features
- Fully supports SD card standard 1.0/1.1/2.0
- Supports SD high capacity standard
- Supports SPI mode
- Supports Command class 0/2/4/5/6/7/8/10
- Supports CPRM
- Speed class up to 6
- Supports host clock up to 50MHz
- Supports bus width X1/X4
- Supports SD host function
- Enhanced ESD protection

## NAND Flash Interface Features
- Supports small or large page NAND flash
- Supports SLC or MLC NAND flash

## Low Power Consumption
- Operating current is about 10mA during data transfer
- Supports Sleep Mode, current is less than 200uA during Sleep Mode
- Fast wake up during Sleep Mode

## Package
- 48-pin TQFP or QFN package
- Die form
Programming process

- Open programmer
- Start burn
- Check flash size
- Set up programming
- Program firmware
- Wait for next card

Windows programmer
x86

AX2005 programming jig
8051

AX211 SD card
8051

2005FM.BIN
Load TestBoot.BIN
Load FLASH_SCAN.BIN
Load FLASH_PRO.BIN
Load correct BIN file

Boot
Ready

Passthru SD commands

Results
Okay
Done

Load SD interpreter
Run flash scan, send Result back to host
Load code to RAM, Return Okay
Write firmware to Flash
SD Protocol: Hardware

- **Signals:**
  - CMD
  - DAT0 – DAT3
  - CLK

- **Signal integrity**
  - Commands use CRC7
  - Data uses CRC16

- Also supports SPI mode
SD Protocol: Software

• 64 Possible Commands
  – CMD0: Reset / Go Idle
  – CMD10: Get CID
  – CMD41: ACMD “escape”
  – CMD60 – CMD63: Reserved for mfgr

• 32 bits of “argument” data

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>bit 5...bit 0</th>
<th>bit 31...bit 0</th>
<th>bit 6...bit 0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>start bit</td>
<td>host</td>
<td>command</td>
<td>argument</td>
<td>CRC7¹</td>
<td>end bit</td>
</tr>
</tbody>
</table>

[SanDisk Product Manual V1.9]
### SD Protocol: Response

<table>
<thead>
<tr>
<th>Bit position</th>
<th>47</th>
<th>46</th>
<th>[45:40]</th>
<th>[39:8]</th>
<th>[7:1]</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width (bits)</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>32</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>'0'</td>
<td>'0'</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>'1'</td>
</tr>
<tr>
<td>Description</td>
<td>start bit</td>
<td>transmission bit</td>
<td>command index</td>
<td>card status</td>
<td>CRC7</td>
<td>end bit</td>
</tr>
</tbody>
</table>

#### Table 4-29: Response R1

<table>
<thead>
<tr>
<th>C</th>
<th>CARD_IS_LOCKED</th>
<th>E</th>
<th>R</th>
<th>X</th>
<th>0 = card unlocked</th>
<th>1 = card locked</th>
<th>Set when a sequence or password error has been detected in lock/unlock card command.</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>LOCK_UNLOCK_FAILED</td>
<td>E</td>
<td>R</td>
<td>X</td>
<td>0 = no error</td>
<td>1 = error</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>23</td>
<td>COM_CRC_ERROR</td>
<td>E</td>
<td>R</td>
<td></td>
<td>0 = no error</td>
<td>1 = error</td>
<td>The CRC check of the previous command failed.</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ILLEGAL_COMMAND</td>
<td>E</td>
<td>R</td>
<td>X</td>
<td>0 = no error</td>
<td>1 = error</td>
<td>Command not legal for the card state.</td>
<td>C</td>
</tr>
<tr>
<td>21</td>
<td>CARD_ECC_FAILED</td>
<td>E</td>
<td>R</td>
<td>X</td>
<td>0 = success</td>
<td>1 = failure</td>
<td>Card internal ECC was applied but failed to correct the data.</td>
<td>C</td>
</tr>
<tr>
<td>20</td>
<td>CC_ERROR</td>
<td>E</td>
<td>R</td>
<td>X</td>
<td>0 = no error</td>
<td>1 = error</td>
<td>Internal card controller error</td>
<td>C</td>
</tr>
<tr>
<td>19</td>
<td>ERROR</td>
<td>E</td>
<td>R</td>
<td>X</td>
<td>0 = no error</td>
<td>1 = error</td>
<td>A general or an unknown error</td>
<td>C</td>
</tr>
</tbody>
</table>

[SD Simplified Layer Spec]
Fuzzing knock sequence

- 64 possible commands
  - Only 4 “manufacturer” commands
  - $2^{32}$ possible arguments

- Fuzz sequence:
  - Reset card
  - Send random command/argument
  - Check for a response
  - No response means it may have crashed
Still works!
No success

• Huge number of possibilities
• Fuzzer can run non-interactively
• Try a different approach
  – Look at the firmware burner
Programming jig

- AX2005
- Bit-banged SD
Running code

- Noticed 'APPO' in AX2005 firmware
- Preceeded by #63
- Maybe the knock is "CMD63 APPO"
- Card seems to respond
  - Doesn't say "invalid command"
  - Doesn't respond at all for 130 cycles
  - If CRC16 is valid, card stops responding at all
Writing a debugger

• We can run code. Great!
• We don't know what to run! Darn.
• Debugger can go over SD
• We have example code
TestBoot.bin

- 512 bytes
- Easy to analyze
- Tells us entry point
- Contains SD state machine
Also, Original Card Firmware Dump

[SD Simplified Layer Spec]
Writing a debugger

- Borrow TestBoot.bin
  - Code doesn't work out of the box
- No debugger whatsoever
  - Maybe we can wiggle a pin?
GPIO hunting

• Probably 1 – 3 registers
  – Set/Clear register value
  – Set/Clear pullup
  – Set pin function

• Toggle them with some frequency
Fuzzer

• Generate an 8051 program that:
  – Pokes value to a random SFR
  – Delays a while
  – Changes SFR value
  – Delays again
  – Repeat

• Read GPIO input values on host
  – Watch for toggling pins
“Hello, World” that finally worked!

```assembly
fuzz:
    mov   0xef, #0x00
    acall sleep
    mov   0xef, #0xff
    acall sleep
    sjmp  fuzz

sleep:
    mov   R5, #0xff
    mov   R6, #0x20

    top_of_pause:
    djnz  R5, top_of_pause
    djnz  R6, top_of_pause
    ret
```
"Hello, World"

Observed 65 changes:

```
00000000  57 57 57 57 57 57 57 47  47 47 47 47 47 57 57 57  WWWWWWWWGGGGGWW
00000010  57 57 57 57 57 57 47 47  47 47 47 47 47 57 57 57  WWWWWWWWGGGGGWW
00000020  57 57 57 57 57 57 47 47  47 47 47 47 47 57 57 57  WWWWWWWWGGGGGWW
00000030  57 57 57 57 57 57 47 47  47 47 47 47 47 57 57 57  WWWWWWWWGGGGGWW
00000040  57 57 57 57 57 57 47 47  47 47 47 47 57 57 57 57  WWWWWWWGWWWWWWW
00000050  57 57 57 57 57 57 47 47  47 47 47 47 57 57 57 57  WWWWWWWGWWWWWWW
00000060  57 57 57 57 57 57 47 47  47 47 47 47 57 57 57 57  WWWWWWWGWWWWWWW
00000070  57 57 57 57 57 57 47 47  47 47 47 57 47 57 57 57  WWWWWWWGWWWWWWW
00000080  57 57 57 57 57 57 47 47  47 47 57 57 57 57 57 57  WWWWWWWGWWWWWWW
00000090  57 57 57 57 57 57 47 47  47 47 57 57 57 57 57 57  WWWWWWWGWWWWWWW
000000a0  57 57 47 47 47 47 47 47  47 57 57 57 57 57 57 57  WWWGGGGGGWWWWWWW
000000b0  57 57 47 47 47 47 47 47  47 57 57 57 57 57 57 57  WWWGGGGGGWWWWWWW
000000c0  57 57 47 47 47 47 47 47  47 57 57 57 57 57 57 57  WWWGGGGGGWWWWWWW
000000d0  57 57 47 47 47 47 47 47  47 57 57 57 57 57 57 57  WWWGGGGGGWWWWWWW
000000e0  57 57 47 47 47 47 47 47  47 57 57 57 57 57 57 57  WWWGGGGGGWWWWWWW
000000f0  57 57 47 47 47 47 47 47  47 57 57 57 57 57 57 57  WWWGGGGGGWWWWWWW
00000100  57 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  WGGGGGGGWWWWWWW
00000110  57 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  WGGGGGGGWWWWWWW
00000120  57 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  WGGGGGGGWWWWWWW
00000130  57 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  WGGGGGGGWWWWWWW
00000140  57 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  WGGGGGGGWWWWWWW
00000150  57 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  WGGGGGGGWWWWWWW
00000160  47 47 47 47 47 47 47 47  47 57 57 57 57 57 57 57  GGGGGGGWWWWWW
00000170  47 47 47 47 47 47 47 47  47 57 57 57 57 57 57 57  GGGGGGGWWWWWW
00000180  47 47 47 47 47 47 47 47  47 57 57 57 57 57 57 57  GGGGGGGWWWWWW
00000190  47 47 47 47 47 47 47 47  47 57 57 57 57 57 57 57  GGGGGGGWWWWWW
000001a0  47 47 47 47 47 47 47 47  47 57 57 57 57 57 57 57  GGGGGGGWWWWWW
000001b0  47 47 47 47 47 47 47 47  57 57 57 57 57 57 57 57  GGGGGGGWWWWWW
000001c0  47 47 47 47 47 47 47 47  57 57 57 57 57 57 57 47  GGGGGGGGWWWGB
000001d0  47 47 47 47 47 47 47 47  57 57 57 57 57 57 57 47  GGGGGGGGWWWGB
000001e0  47 47 47 47 47 47 47 47  57 57 57 57 57 57 57 47  GGGGGGGGWWWGB
000001f0  47 47 47 47 47 47 47 47  57 57 57 57 57 57 57 47  GGGGGGGGWWWGB
```
Writing a Debugger

- Bidirectional SD communications
  - Send CMD with four 8-byte arguments
  - Get CMD back with four 8-byte responses
- Basic commands
  - peek/poke
  - GPIO control
  - IRQ status
  - NAND emulator
  - 32-bit opcodes?
- https://github.com/xobs/ax2xx-code
0xa5 “Escape” opcode

- Undefined in standard 8051
- All over the place in AX211 code
- 0xa5 0xXY
- 0xa5 0x7Y 0xWZ
8 bit or 32 bit?

- Four 32-bit registers
- "extop" debugger command
- Discovered 32-bit clr, not, inc, dec
- Many undiscovered opcodes
AX215

- Similar to AX211
- Faster, more GPIOs, different SFR map
Time for Tin Foil Hats

• Attack scenarios:
  – Eavesdropping
    • Report smaller than actual capacity
    • Data is sequestered to hidden sectors that are unerasable
  – ToC/ToU
    • Present one version of file for verification, another for execution
    • Bootloader manipulation, etc.
  – Selective-modify
    • Scan for assets of interest, e.g. security keys, binaries, and replace with insecure versions
Other Direction: Samsung MMC

- Samsung pushed firmware patch to eMMC cards in Android
- Contains ARM7 code
  - Uses “class 8” instructions reserved for manufacturer

“By inspecting some code, it seems that we know how to dump the eMMC RAM:
Look at the function mmc_set_wearlevel_page in line 206. It patches the RAM (using the
method mentioned before), then it validates what it has written (in lines 255-290). Seems that
the procedure to read the RAM is as following:
1. CMD62(0xEFAC62EC) CMD62(0x10210002) to enter RAM reading mode
2. MMC_ERASE_GROUP_START(Address to read) MMC_ERASE_GROUP_END(Length
to read) MMC_ERASE(0)
3. MMC_READ_SINGLE_BLOCK to read the data
4. CMD62(0xEFAC62EC) CMD62(0xDECCEE) to exit RAM reading mode”
Other Direction: TLC

- TLC Flash has scrambling applied to avoid “read-disturb” and “program-disturb” issues
  - Scrambling is a proprietary algorithm, as of yet unknown
  - Highly structured
Wrap-up

- SD cards contain fully programmable microcontrollers
- Controller program modifiable via special host commands
  - Potential for MITM attack scenarios 😞
  - Potential for extremely cheap microcontroller for fun projects 😊
Special Thanks

• Shout out to .mudge for creating CFT which enabled this research, and many other good things (some yet to come!)
Q&A

- Demo (time allowing)
- Thanks for your attention!
About the 8051

Internal RAM

RAM: 0x00 - 0x7f

Registers: 0x80 - 0xff

mov 0x40, #30

External RAM

0x0000 - 0xffff

mov DPTR, #0x4700
mov A, #30
movx @DPTR, A