SECURITY VULNERABILITIES OF DIGITAL VIDEO BROADCAST CHIPSETS

Adam Gowdiak
Security Explorations
INTRODUCTION

About Security Explorations

- Security start-up company from Poland
- Provides various services in the area of security and vulnerability research
- Commercial and Pro Bono research projects
- Came to life in a result of a true passion of its founder for breaking security of things and analyzing software for security defects
- Our ambition is to conduct quality, unbiased, vendor-free and independent security and vulnerability research
INTRODUCTION

Presentation Goal

- Continuation of our research in a digital satellite TV area
- Educate about security risks associated with less known technologies and platforms such as those used in a digital satellite TV ecosystem
- Warn about security risks associated with
  - closed ecosystems such as digital satellite TV
  - insecurely implemented proprietary hardware components
  - 3rd party security evaluation processes
INTRODUCTION

DISCLAIMER

- Information provided in this presentation is for **educational purposes only**
- Security Explorations neither promotes, nor encourages the acts of a digital satellite TV piracy
- Any use of the information provided in this presentation for illegal purposes is strictly prohibited
- In case of legal actions taken against Security Explorations, the following web pages will be updated
  
DIGITAL SATELLITE TV

Why bother about content security?

- Pay TV piracy remains a major concern for channels and operators
  - it leads to financial losses for the European pay TV industry
  - it substantially damages the image of transmitters and content rights holders
  - it reduces the allure and payback of investing in the industry
  - it hurts the industry and its innovation capabilities

- Signal theft estimated to be more than $2.1 billion at the end of 2011 for Asia region alone (CASBAA)
Digital Satellite TV

Security of a premium content

- Paid, premium content broadcasted in encrypted form
  - Scrambling at the TS or PES level
    - `transport_scrambling_control` bit of MPEG TS packet
  - Common Scrambling Algorithm (CSA) and its derivatives
    - Shared 64-bit secret key (Control Word)
  - Dedicated security chipsets for decryption

- Key components in the security system
  - Subscriber’s smartcard
    - Holds information about subscriber’s access rights to programming
    - Releases decryption keys to the set-top-box if access to a given service is granted
  - Set-top-box
    - Conducts decryption of a scrambled content with the use of a received decryption key
DIGITAL SATELLITE TV

Security of a premium content (2)

- Satellite dish
  - Source of scrambled TV signal

- SET-TOP-BOX
  - Transport stream demux
  - Audio and video decryption with the use of keys obtained from a subscriber’s smartcard
  - MPEG parsing and decoding

- TV screen
  - Displaying of decrypted and decoded Audio and Video data

- SUBSCRIBER'S SMARTCARD
  - Releases decryption keys to set-top-box if access to the content granted

- ENCRYPTED KEY
- KEY DATA
DIGITAL SATELLITE TV

Control Words (CW)

- 64-bit secret keys used to descramble encrypted MPEG streams
  - Audio, video and data
- Unique to each programming
- Generated automatically by the content provider
  - Changed every ~10s
  - Odd and even keys for uninterrupted programming reception
    - Current and next key
- Broadcasted in encrypted form to client devices (set-top-boxes)
  - carried in entitlement control messages (ECM)
  - encrypted with the use of asymmetric crypto (i.e. RSA)
DIGITAL SATELLITE TV

Entitlement Control Messages (ECM)

- ECM messages contain private conditional access information such as Control Words
  - Broadcasted by the means of a dedicated MPEG stream
  - Message format specific to CAS vendor
- PID of MPEG stream carrying ECM messages denoted by CA_descriptor
  - If elementary stream is scrambled, a CA descriptor shall be present for the program containing that elementary stream
  - Usually present in TS_program_map_section
    - MPEG table_id = 0x02
DIGITAL SATELLITE TV

Conditional Access System (CAS)

- It protects the content by requiring certain criteria to be met before granting access to the content.

- Subscriber’s smartcard holds information about subscriber’s access rights to a given programming:
  - what programming / program packages a subscriber is entitled to watch.

- Only authorized client devices (paying subscribers) can decrypt MPEG streams for premium content:
  - Set-top-box device asks the smartcard to decrypt encrypted Control Word (ECM message).
  - The smartcard makes sure that access to the content can be granted and releases the plaintext value of a Control Word.
DIGITAL SATELLITE TV

CAS architecture (set-top-box side)

- Scrambled MPEG stream (audio, video, data)
- Encrypted ECM message
- Descrambling keys (Control Words)
- Descrambled MPEG stream
- Access rights:
  - programming
  - VOD movies

SET-TOP-BOX

DESCRAMBLER

CAS - smartcard

Plaintext values of descrambling keys available to set-top-box

HITBSecConf, May 24-25, 2012, Amsterdam, The Netherlands
DIGITAL SATELLITE TV

Threats to the model

- Premium content is encrypted and broadcasted to all subscribers with the use of same crypto key (Control Word)
- One rogue subscriber with access to all premium content can share Control Word keys with others over the Internet
  - illegal reception / distribution of premium programming aka signal theft
  - Control Words sharing
DIGITAL SATELLITE TV

CAS with chipset pairing

- Control Words unique for each pair of a subscriber (smartcard) / client device (set-top-box)
  - smart cards can be used only with secure devices
  - the link between the smart card and the client device is secured
  - illegal content redistribution is prevented (no more CW sharing)
- Chipset pairing has a form of a cryptographic function
- It is usually implemented in a silicon chip (DVB chipset)
DIGITAL SATELLITE TV

CAS with chipset pairing (set-top-box side)

Set-top-box

Plaintext values of descrambling keys NOT available to set-top-box

Descriptor

Encrypted ECM message

Descrambling keys (Encrypted Control Words)

Descrambled MPEG stream

CAS - smartcard

Access rights
- programming
- VOD movies

Scrambled MPEG stream
(audio, video, data)

PAIRING FUNCTION

HITBSecConf, May 24-25, 2012, Amsterdam, The Netherlands
DIGITAL SATELLITE TV

Pairing function

- A function that cryptographically ties a set-top-box device and a subscriber’s smartcard

\[ \text{encCW} = \text{PAIRING}_{\text{enc}}(CW, \text{CWPK}) \]

\[ CW = \text{PAIRING}_{\text{dec}}(\text{encCW}, \text{CWPK}) \]

- Control Words pairing key (CWPK)
  - Unique to each subscriber
  - Assigned to it at the time of activating a given user’s digital satellite TV subscription
  - Usually, a function of a unique DVB chipset’s key
Conax AS is one of the major CAS providers for the Pay TV industry

- More than 350 installations in 80 countries world-wide

CAS implemented in software and hardware

- Partnership with set-top-box vendors
- Partnership with many DVB chipset vendors to implement chipset pairing functionality
  - STMicroelectronics, Broadcom, Renesas Electronics, …
Mandatory, comprehensive evaluation of all qualified chipsets run at independent, world-leading security laboratories

- All set-top-boxes and DVB chipsets implementing Conax CAS with chipset pairing undergo rigorous security evaluation process
- Official scoring assigned to set-top-boxes and DVB chipsets and certified in writing by Conax CSO
  - „0 represents no security and 9 corresponds to the security level of Conax smart cards”

Conax Security Department (09-Jan-2012)
DIGITAL SATELLITE TV

Conax security certification

Source: Neotion company website
DVB CHIPSETS

Introduction

- DVB chipsets implement the core functionality related to the handling of MPEG transport streams and A/V data such as:
  - MPEG transport filtering and descrambling (incl. chipset pairing function)
  - audio and video decoding
  - graphics display
  - communication interfaces
  - memory interfaces
- For security and efficiency reasons, they are usually implemented as a single chip (system-on-chip or SoC)
  - Multiple processor cores for various functions
DVB CHIPSETS

STMicroelectronics implementation

- STi7100 single-chip, high-definition STB decoder
STMicroelectronics implementation (2)

- STi7111 single-chip, high-definition STB decoder
DVB CHIPSETS

STMicroelectronics STB H.264 generations

**GENERATION-1**

- **STi7100**
- **STi7103/Douglas**
- 7109, 5202
- 90 / 80nm
- Mass Production: 2007

**GENERATION-2**

- **STi7105**
- **STi7104/Sequoia**
- 7111, 7141, 7200
- 5211, 5206
- 65 / 55 nm
- Production Start: 2009

**Source:** *Multimedia Convergence & ACCI Sector Overview*, Philippe Lambinet, STMicroelectronics
Implementation of a chipset pairing function in a proprietary silicon chip makes it far more difficult to reverse engineer and break

- no target software for the static analysis / reverse engineering or runtime interception
- undocumented interfaces
- unknown implementation of the pairing function
- unknown crypto algorithm and keys (their sizes, byte order, etc.)
TEDIOUS ANALYTICAL AND REVERSE-ENGINEERING WORK

By gathering and gluing together many pieces of information (clues), it was possible not only to discover the operation and implementation of investigated DVB chips, but also find security weaknesses in them.

THE TOOLS

- Without custom reverse engineering tools we would not be able to successfully complete most of our projects
- This is especially valid for SE-2011-01 project
BREAKING DVB CHIPSETS

Common approach (chips documentation)

- Data briefs available from st.com (STi710x, STi7111, STM7710, etc.)
  - Generic chip architectures
  - Processor cores
    - ST40 32-bit superscalar RISC CPU
    - Dual ST231 CPU cores for audio and video decoding
  - Transport subsystem
    - Programmable Transport Interface (PTI)
      - PID filtering, Demultiplexing, Descrambling
    - Transport Stream Merger (TSM) and router
  - FDMA controller
    - PES parsing and start code detection
    - Routing elementary streams to A/V buffers
BREAKING DVB CHIPSETS

Common approach (discovering core device drivers)

- Device drivers implementing Control Words operations
- Static / binary analysis
  - Inspecting libraries and device driver code / symbols
  - Figuring out code dependencies
    - Call and link graphs
BREAKING DVB CHIPSETS

Common approach (pinning down CW API calls)

Runtime analysis
- SmartCard APDU watch
- IOCTL watches

STTKDMA device driver

DecryptKey IOCTL C0305A09
45 e5 6c 28 b4 02 32 5a
00 00 00 00 00 00 00 00
4c db 73 29 14 4d d1 2a
14 01 75 29 28 4d d1 2a

GSECHAL device driver

DecryptSCK IOCTL c020ff06
45 e5 6c 28 b4 02 32 5a
00 00 00 00 00 00 00 00
06 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

Encrypted CW1 as input
Non-HD prepaid satellite TV service (TNK) available along the main Platform ‘N’
- Different set-top-box decoders
  - Technisat, ...
- Conax CAS smartcards

A few services available to both SAT TV platforms
- Shared audio / video streams (same broadcast)
- Separate conditional access information
  - separate ECM streams
BREAKING DVB CHIPSETS

Helpful CAS system implementation (2)

Program Stream Information

```
[0] "DIGITAL TV" "TVN" dvb://13e.3e8.10d7
ServiceDetails: com.adb.dvb.si.AdbSIService, tune_id=4, type=4, version=1, onid=13e, tsid=3e8,
svid=10d7, name=TVN, type=1
PMTService: com.adb.dvb.si.AdbPMTService, tune_id=4, type=5, version=3, onid=318, tsid=1000,
svid=4311, PCR=512
PMTElementaryStreams:
  tag 00 type 02 dvb://13e.3e8.10d7.0 PID 0200
  tag 8a type 04 dvb://13e.3e8.10d7.8a PID 028a
  tag 8b type 06 dvb://13e.3e8.10d7.8b PID 028b
  tag 40 type 06 dvb://13e.3e8.10d7.40 PID 0240
  tag fe type c0 dvb://13e.
  tag fe type c1 dvb://13e.
  tag 01 type 06 dvb://13e.
Descriptors:
  ...
  tag 0009 UNKNOWN
    len 000f 05 00 a6 4c 10 01 00 13 01 20 14 03 03 2a 00
  ...
  tag 0009 UNKNOWN
    len 0004 0b 00 a5 08
  ...
  tag 0009 UNKNOWN
    len 0004 0b 01 a5 94
  ...
```

- CA ID 0x0b00 (CONAX CAS)
- PID for ECM stream 0x0508
- CA ID 0x0b01 (CONAX CAS)
- PID for ECM stream 0x0594
BREAKING DVB CHIPSETS

Helpful CAS system implementation (3)

- Parallel Conax CAS without chipset pairing
- The plaintext values of encrypted Control Words
BREAKING STi7100 CHIPSET

Device drivers’ functionality and operation

- Detailed analysis of GSECHAL’s DecryptSCK function
  - The meaning of configuration data
    - SecureMode = 1
    - UsingAES = 0
  - Memory mapped I/O registers
    - CONFIG
    - STATUS
    - COMMAND
    - DATA
  - Implementation of direct chip programming commands
    - gSecWaitForComplete, gSecDataRead, gSecDataWrite,
BREAKING STi7100 CHIPSET

Reconstruction of a chip’s programming sequences

DecryptionSCK IOCTL (pseudocode)

If (SecureMode=1, UsingAES=1) {
    gSecDataWrite buf 4
    gSecInstWrite 3|arg<<8
}

If (SecureMode=0, UsingAES=1) {
    d1 -> BASE_ADDR +0x6100+arg<<4
    d2 -> BASE_ADDR +0x6104+arg<<4
    d3 -> BASE_ADDR +0x6108+arg<<4
    d4 -> BASE_ADDR +0x6110c+arg<<4
}

If (SecureMode=1, UsingAES=0) {
    gSecDataWrite buf 2
    gSecInstWrite 3|arg<<8
    gSecInstWrite 3|arg<<8
    gSecSetKeyPtr
}

If (SecureMode=0, UsingAES=0) {
    d1 -> BASE_ADDR +0x6100+arg<<4
    d2 -> BASE_ADDR +0x6104+arg<<4
    d3 -> BASE_ADDR +0x6108+arg<<4
    d4 -> BASE_ADDR +0x6110c+arg<<4
    gSecSetKeyPtr

- Default GSEC chip operation
- SecureMode value defined by chip fuses
Secure storage area for Control Word keys
- 0x6100 offset from the chip’s base addr
- Unavailable for reading / writing
  - Read operation always returns ZEROS
  - Write operation does not disrupt the descrambling process

The arguments to the **DecryptSCK command** include the index of the key slot to load with 0x10 bytes (key data)
- Device driver code makes sure that this index is within the 0x00-0x31 bounds
The ability to extract plaintext values of Control Words

- the chip needs to be programmed manually by issuing commands directly to its I/O mapped registers
- index of the `DecryptSCK` command needs to be greater than 0x31

\[
\text{plaintext CW mem} = 0x6100 + \text{key_idx} \times 0x10
\]

where \text{key_idx}>0x31
**BREAKING STi7100 CHIPSET**

Security vulnerability (formula)

\[
\text{Key}_{32h} \leftarrow TDES_{\text{dec}}(\text{encCW}, \text{plainCWPK})
\]

\[
\text{Key}_{32h} \equiv \text{plainCW}
\]

Where:

- \( \text{Key}_{32h} = \text{key slot} \#32h \) (accessible chip location)
- \( \text{encCW} = \text{encrypted Control Word value} \) (known value)
- \( \text{plainCWPK} = \text{plaintext Pairing Key value} \) (unknown value)
BREAKING STi7100 CHIPSET

Security vulnerability (SoC location)
BREAKING STi7111 CHIPSET

Introduction

- Different design brings more challenges
  - STTKDMA chip component
  - One DecryptKey IOCTL for key related operations
    - Control Words
    - Pairing Key (CWPK)
    - AES keys

- Easier reverse engineering
  - Modular architecture of ITI2850ST/ ITI2849ST set-top-boxes’ OS distribution
    - Many dedicated user level libraries
      - Conax CA, Crypto operations, NAND encryption, STB configuration, …
    - Text XML configuration files
      - Conax CA client settings
  - Support for crypto DMA operations
  - Kernel symbols via /proc/kallsyms (680KB+)
BREAKING STi7111 CHIPSET

Dedicated key memories

- Separate memory mapped chip regions for AES and Control Word keys
  - Deduced with the help of kernel and user level library symbols
  - Code / data symbols associated with accesses to chip’s I/O memory

DecryptKEY FUNCTIONALITY
- cpcw_keys, descrambling keys (set_cleartext_descramblingkey)
  - 3100 offset, indices 0x00-0x31, key size 0x10

CRYPTO DMA FUNCTIONALITY
- sttkdma_keys, cdma_dev_keys, cpcwKeys (STDRMCRYPTO_AES_LoadKeySlot)
  - 3420 offset, indices 0x00-0x07, key size 0x10
BREAKING STi7111 CHIPSET

CCORE library

- The library used for crypto DMA operations
- Reverse engineering the meaning of CCORE library/chips configuration bits and input/output arguments
  - Manual analysis of data propagation
    - libstd_drv_ccore.so API -> STTKDMA device driver API -> chip’s configuration registers
      - DMA CONFIG
      - TKD CONFIG
  - Analysis of STTKDMA code writing to configuration registers
    - resetAES_NOT_TDES
- Custom AES/TDES Java subroutines
  - Verification of CCORE results
BREAKING STi7111 CHIPSET

The existence of a chip specific SCK key

- Initial hints in STi7100 GSECHAL device driver
  - DecryptSCK command

- Confirmed at a time of the analysis of a set-top-box boot loader code (SH4 emulator)
  - The use of SCK key to decrypt the boot loader code
    - No initialization of the usual key registers
    - Different chip configuration bits

- Used by a device driver from a software upgrade OS distribution
  - The use of SCK key for NAND encryption
    - parm=nand_crypt_use_sck_key:Use SCK key instead of default ADB key for NAND encryption
Hints in `sttkdma_core_user.ko` module

- Code symbols
  - `st_tkdma_loader`
  - `st_tkdma_loader_checksum`
- Data symbols
  - `tkdma_fw_address_1`
  - `tkdma_fw_address_2`
- Writing data / code to STi7111 chip’s I/O space
  - 5944 code bytes
  - 1156 data bytes

Firmware code implementing unknown processor instructions
IST FP6 PROSYD EU project ([http://www.prosyd.org](http://www.prosyd.org))

Paragraph 2.3 of Deliverable D1.4/1 gives some information about the SLIM Core Processor

- a collaboration between ST UK and OneSpin after the spin-off from Infineon
- lightweight processor with 27 instructions and a 4-stage pipeline
- processor special features: a coprocessor interface; circular buffer operation; a STOP instruction

BREAKING STi7111 CHIPSET

SLIM Core processor
BREAKING STi7111 CHIPSET

JMP instruction format

- OpenDuckbox project (http://gitorious.org/open-duckbox-project-sh4)
  - GNU source code for SLIM Core Generic driver
  - `slim_boot_core` function leaks information about the format of one Slim Core instruction (JMP)
    - memory addressing (by word number)
    - instruction opcode width

  // Init imem so every instruction is a jump to itself
  for (n = 0; n < core->imem_size / 4; n++)
    SLIM_IMEM(core, n) = 0x00d00010 | (n & 0xf) | ((n & 0xffff0) << 4);
**BREAKING STi7111 CHIPSET**

Finding patterns in SLIM Core code

<table>
<thead>
<tr>
<th>0x4020</th>
<th>0x4024</th>
<th>0x4028</th>
<th>0x402c</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIP ID</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**Result of GetPublicID command**

**SLIM Core firmware sequence**

<table>
<thead>
<tr>
<th>01a1</th>
<th>0x00a5008b</th>
<th>ld r5,[r0,008b] // 0x5e2c</th>
</tr>
</thead>
<tbody>
<tr>
<td>01a2</td>
<td>0x00b9001f</td>
<td>ld r9,[r0,001f] // 0x407c</td>
</tr>
<tr>
<td>01a3</td>
<td>0x00b05008</td>
<td>st r5,[r0,0008] // 0x4020</td>
</tr>
<tr>
<td>01a4</td>
<td>0x00b0009</td>
<td>st r0,[r0,0009] // 0x4024</td>
</tr>
<tr>
<td>01a5</td>
<td>0x00b0000a</td>
<td>st r0,[r0,000a] // 0x4028</td>
</tr>
<tr>
<td>01a6</td>
<td>0x00b0000b</td>
<td>st r0,[r0,000b] // 0x402c</td>
</tr>
<tr>
<td>01a7</td>
<td>0x00d01cl</td>
<td>jmp _01ca</td>
</tr>
</tbody>
</table>

**Discovery of STORE instruction format**

- Firmware pattern matching format of GetPublicID result
- Confirmed by changing ST R5 with ST R0 instruction
Discovery of SLIM Core instruction opcodes

- Exploited the ability to change the operation of SLIM Core firmware in runtime
  - Overwriting chip’s memory loaded with firmware code
  - should secure crypto chip allow for it?
- Replacement of an arbitrary instruction from the code path of GetPublicID function
- Analysis of the instruction’s execution effect to memory and registers
  - Discovery of load / store instructions format
BREAKING STi7111 CHIPSET

Reverse engineering SLIM Core instructions (2)

 Injecting user code into GetPublicID codepath
  - user code in unused space
  - input and output arguments

Execution return to firmware code

User code

jmp b_label

Execution transfer to user code

SLIM Core firmware

GetPublicID codepath

jmp user_code
st r0, [r0, 0009]
st r0, [r0, 000a]
st r0, [r0, 000b]
jmp l_01ca

HITBSecConf, May 24-25, 2012, Amsterdam, The Netherlands
BREAKING STi7111 CHIPSET

Reverse engineering SLIM Core instructions (3)

- JMP and LOAD/STORE instructions sufficient to discover the meaning of all other instructions
  - JMP from firmware to user’s code path
    - STORE the contents of registers (firmware context)
      - LOAD user’s environment (contents of registers)
      - EXECUTE unknown SLIM Core instruction opcode
      - STORE user’s environment (contents of registers)
    - LOAD the contents of registers (firmware context)
  - JMP back to firmware code path
- The need to properly handle conditional jumps
One instruction opcode at a time

- LOAD / STORE instructions
- MOV instructions
- CMP instructions
- Conditional branching instructions
- Computational instructions
- Other instructions (bit extraction, manipulation)
- RPT instruction

Scope limited to unknown opcodes from firmware code
BREAKING STi7111 CHIPSET
Reverse engineering SLIM Core instructions (5)

Visible code patterns

MOV instruction patterns

<table>
<thead>
<tr>
<th>MOV instruction confirmed by the result of STORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0003  0x00e30374 mov r3, #0374</td>
</tr>
<tr>
<td>0004  ....</td>
</tr>
<tr>
<td>0005  0x00e4ffff mov r4, #ffff</td>
</tr>
<tr>
<td>0006  0x00e3ffff mov r3, #ffff</td>
</tr>
<tr>
<td>0007  ....</td>
</tr>
<tr>
<td>0008  0x00e30001 mov r3, #0001</td>
</tr>
</tbody>
</table>

CMP / conditional jump instruction patterns

<table>
<thead>
<tr>
<th>CMP values correspond to device driver commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>0014  0x00981026 je 1_0026</td>
</tr>
<tr>
<td>0015  0x00c3002 cmp r3, #02</td>
</tr>
<tr>
<td>0016  0x00981028 je 1_0028</td>
</tr>
<tr>
<td>0017  0x00c3006 cmp r3, #06</td>
</tr>
<tr>
<td>0018  0x0098102a je 1_002a</td>
</tr>
<tr>
<td>0019  0x00c300b cmp r3, #0b</td>
</tr>
<tr>
<td>001a  0x0098102c je 1_002c</td>
</tr>
<tr>
<td>001b  0x00c300f cmp r3, #0f</td>
</tr>
<tr>
<td>001c  0x0098102e je 1_002e</td>
</tr>
<tr>
<td>001d  0x00c3003 cmp r3, #03</td>
</tr>
<tr>
<td>001e  0x00981030 je 1_0030</td>
</tr>
<tr>
<td>001f  0x00c3007 cmp r3, #07</td>
</tr>
</tbody>
</table>
BREAKING STi7111 CHIPSET

SLIM Core disassembler

- Final disassembly dump of Slim Core firmware code
  - 1400+ instructions disassembled
  - ~11 instruction opcodes not recognized
    - Not relevant from the analysis point of view
- Sufficient data for firmware analysis
  - Discovery of separate dispatching for DMA and all TKD operations
    - Semi-threads (context-switching)
  - Discovery of a key initialization subroutine
The goal was to locate SLIM Core instruction sequences implementing DecryptKey functionality.

Tracer implementation

- SLIM core part
  - Custom code on the GetPublicID function path
    - Binary instrumented instruction copied from a currently traced code location
    - SLIM Core instruction executed in the original registers context
    - Heavy use of the SLIM core disassembler

- Java part
  - Logging
  - SLIM Core syncing and control code
BREAKING STi7111 CHIPSET

Tracing SLIM Core firmware (output log)

```
starting logger at 80
break at: 0x00000086
r0 00000000 *r1 00000001 *r2 00000100 r3 00000000
*r4 00000011 *r5 00000031 *r6 00001103 *r7 00000005
*r8 00000006 *r9 31ff0001 r10 00000000 *r11 00000001
r12 00000000 *r13 0000024e *r14 000000d0 IP 00000086
0086 0x00e10001 mov r1,#0001

SCDC stopped
0086 0x00e10001 mov r1,#0001
break at: 0x00000087
r0 00000000 r1 00000001 r2 00000100 r3 00000000
r4 00000011 r5 00000031 r6 00001103 r7 00000005
r8 00000006 r9 31ff0001 r10 00000000 r11 00000001
r12 00000000 r13 0000024e r14 000000d0 IP 00000087

0087 0x00a20048 ld r2,[r0,0048] // 0x4120
break at: 0x00000088
r0 00000000 r1 00000001 *r2 00000001 r3 00000000
r4 00000011 r5 00000031 r6 00001103 r7 00000005
r8 00000006 r9 31ff0001 r10 00000000 r11 00000001
r12 00000000 r13 0000024e r14 000000d0 IP 00000088

0088 0x00722c21 bitval r2,r2,#0002
0089 0x00881091 jz l_0091
break at: 0x00000091
r0 00000000 r1 00000001 *r2 00000000 r3 00000000
r4 00000011 r5 00000031 r6 00001103 r7 00000005
r8 00000006 r9 31ff0001 r10 00000000 r11 00000001
r12 00000000 r13 0000024e r14 000000d0 IP 00000091
```
BREAKING STi7111 CHIPSET

Tracing SLIM Core firmware (DecryptKey code)

DecryptKey implementation

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0206</td>
<td>0x00fa4000</td>
<td>copTDES</td>
</tr>
<tr>
<td>0207</td>
<td>0x000f093c</td>
<td>mov r15, r9</td>
</tr>
<tr>
<td>0208</td>
<td>0x008e1208</td>
<td>wait1</td>
</tr>
<tr>
<td>0209</td>
<td>0x00af0008</td>
<td>ld r15, [r0, 0008]</td>
</tr>
<tr>
<td>020a</td>
<td>0x00af0009</td>
<td>ld r15, [r0, 0009]</td>
</tr>
<tr>
<td>020b</td>
<td>0x00af000a</td>
<td>ld r15, [r0, 000a]</td>
</tr>
<tr>
<td>020c</td>
<td>0x00af000b</td>
<td>ld r15, [r0, 000b]</td>
</tr>
<tr>
<td>020d</td>
<td>0x008e120d</td>
<td>wait1</td>
</tr>
<tr>
<td>020e</td>
<td>0x00500a00</td>
<td>tst r10, r10</td>
</tr>
<tr>
<td>020f</td>
<td>0x00881215</td>
<td>jz l_0215</td>
</tr>
<tr>
<td>0210</td>
<td>0x00b0f008</td>
<td>st r15, [r0, 0008]</td>
</tr>
<tr>
<td>0211</td>
<td>0x00b0f009</td>
<td>st r15, [r0, 0009]</td>
</tr>
<tr>
<td>0212</td>
<td>0x00b0f00a</td>
<td>st r15, [r0, 000a]</td>
</tr>
<tr>
<td>0213</td>
<td>0x00b0f00b</td>
<td>st r15, [r0, 000b]</td>
</tr>
<tr>
<td>0214</td>
<td>0x00d02117</td>
<td>jmp l_0217</td>
</tr>
<tr>
<td>0215</td>
<td>0x00d00004</td>
<td>rpt 4</td>
</tr>
<tr>
<td>0216</td>
<td>0x000000f3c</td>
<td>mov r0, r15</td>
</tr>
<tr>
<td>0217</td>
<td>0x00d0211a</td>
<td>jmp l_021a</td>
</tr>
</tbody>
</table>

put chip into TDES mode
load TKD command
wait
load encrypted CW
wait
optionally store result
BREAKING STi7111 CHIPSET

Internal crypto core (TKD) commands

- Static analysis of SLIM Core firmware disassembly
- Discovery of internal chip commands
Discovery of the meaning of TKD commands by the means of executing special SLIM Core instruction sequence

```java
public static final int tkd_code[] = {
    0x00e61234,  //mov r6,#1234 TKD_CMD_HI
    0x00e55678,  //mov r5,#5678 TKD_CMD_LO
    0x00e00000,  //mov r0,#0000
    0x00756210,  //mov r5,(r6<<16)|r5
    0x00e00000,  //mov r0,#0000
    0x00fa4000,  //COPINS
    0x00e00000,  //mov r0,#0000
    0x000f053c,  //mov r15,r5
    0x00e00000,  //mov r0,#0000
    0x008e1abc,  //WAITINS
    0x00e00000,  //mov r0,#0000
    0x00af0050,  //ld r15,[r0,0050]
    0x00af0051,  //ld r15,[r0,0051]
    0x00af0052,  //ld r15,[r0,0052]
    0x00af0053,  //ld r15,[r0,0053]
    0x00e00000,  //mov r0,#0000
    0x008e1abc,  //WAITINS
    0x00e00000,  //mov r0,#0000
    0x00bf054,   //st r15,[r0,0054]
    0x00bf055,   //st r15,[r0,0055]
    0x00bf056,   //st r15,[r0,0056]
    0x00bf057,   //st r15,[r0,0057]
    0x00e00000,  //mov r0,#0000

};
```

TKD command to test

Chip mode instruction (AES/TDES)

Wait ins (different for AES/TDES mode)

Input data (key)

Wait ins (different for AES/TDES mode)

Output data (key)
BREAKING STi7111 CHIPSET

TKD inspector (playing with the commands)

test> input "b6 04 78 c3 0f 26 a3 06 d5 20 10 0f c0 93 4f f3"
test> ed 0x01ff0000 0x00fa4000 0x008e1abc
tkcmd 01ff0000

[running SLIM code]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

test> input "e0 c9 cd 4e 2f bd 52 a0 e0 c9 cd 4e 2f bd 52 a0"
test> ed 0x15ff0101 0x00fa4000 0x008e1abc
tkcmd 15ff0101

[running SLIM code]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

test> keys
[00] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[01] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[02] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[03] f8 25 41 20 00 00 00 00 00 00 00 00 00 00 00 00 43 11 71
[04] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[05] 00 85 55 26 86 09 29 54 00 85 55 26 86 09 29 54
[06] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[07] d4 c8 94 af 84 84 5c de 17 82 f7 73 1e c3 2f e7
DMA CONFIG: 20 08 00 00
TKD CONFIG: 00 00 00 00
INPUT: e0 c9 cd 4e 2f bd 52 a0 e0 c9 cd 4e 2f bd 52 a0
## BREAKING STi7111 CHIPSET

### TKD commands format

<table>
<thead>
<tr>
<th>Target:</th>
<th>Source:</th>
<th>Key:</th>
<th>Config:</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

**Target:**
- Target, where the result of the operation should be stored
- A key slot number or 0xff for chip registers

**Source:**
- Source, from which data for the operation should be fetched
- A key slot number or 0xff for chip registers

**Key:**
- Key slot number, which holds the key used for the crypto operation
- Value 0x00 usually identifies SCK key (unique key for each chip)

**Config:**
- Configuration bits
- Bit 0 Usually denotes encryption (0) or decryption (1) operation
BREAKING STi7111 CHIPSET

TKD commands format (key operations explained)

- TKD CMD 0x00ff0000
  - Setting encrypted Control Word Pairing Key (CWPK)
  - Interpreted as decryption (always) of register input (0xff) with SCK key (0x00) and storing the result at a key slot 0x00

- TKD CMD 0x20ff0001
  - Setting encrypted Control Word
  - Interpreted as decryption (0x01) of register input (0xff) with SCK key (0x00) and storing the result at a key slot 0x20
BREAKING STi7111 CHIPSET

Security vulnerability #1 (step 2)

Decipher CW into key slot #15 using key slot #1 (TKD 0x15ff0101)

TKD keys

INACCESSIBLE KEYS

00
01 plaintext CWPK
02
10
11
12
15 plaintext Control Word

ACCESSIBLE KEYS

10
11
12
15 plaintext Control Word

SCK

TDES DECRYPTION

encrypted Control Word

TKD CMD 0x15ff0101

Plaintext Control Word available for reading in key slot #15
BREAKING STi7111 CHIPSET

Security vulnerability #1 (formula)

\[ Key_1 \leftarrow TDES_{dec}(encCWPK, SCK) \]
\[ Key_{15h} \leftarrow TDES_{dec}(encCW, Key_1) \]
\[ Key_1 \equiv plainCWPK \]
\[ Key_{15h} \equiv plainCW \]

Where:

- \( Key_1 \) = key slot #01h (inaccessible chip location)
- \( Key_{15h} \) = key slot #15h (accessible chip location)
- \( encCW \) = encrypted Control Word value (known value)
- \( encCWPK \) = encrypted Control Word Pairing Key value (known value)
- \( SCK \) = chip specific key (unknown value)
BREAKING STi7111 CHIPSET

Security vulnerability #2 (step 1)

Decipher CWPK into key slot #15 utilizing SCK key (TKD 0x15000001)
BREAKING STi7111 CHIPSET

Security vulnerability #2 (step 2)

Encrypt RESULT VALUE into registers using SCK key (TKD 0xffff0000)

Plaintext Control Word Pairing Key available for reading
\[ Key_{15h} \leftarrow TDES_{\text{dec}}(\text{plainCWPk}, SCK) \]
\[ \text{plainCWPk} \leftarrow TDES_{\text{enc}}(Key_{15h}, SCK) \]

Where:

- \( Key_{15h} \) = key slot #15h (accessible chip location)
- \( SCK \) = chip specific key (unknown value)
- \( \text{plainCWPk} \) = plaintext Pairing Key value (unknown value)
BREAKING STi7111 CHIPSET

STi7111 security vulnerabilities (SoC location)
Issue 18 makes use of the encrypted value of the chipset pairing key (CWPK)

- CWPK key sent by the operator at the time of activating user’s subscription
  - Encrypted CWPK key bytes returned by the Conax card in response to EMM message

- For ITI2850ST and ITI2849ST set-top-boxes encrypted CWPK key encrypted again and cached in a file
  - `/mnt/flash/secure/7/0`
  - `libstd_cai_client_conax7.so` API for decryption
PROOF OF CONCEPT CODE

Implemented functionality

- Access to information about cryptographic keys
  - Plaintext Control Words (STi7100)
  - Plaintext Control Words and plaintext CWPK (STi7111)
- Control Words sharing via network between arbitrary decoders protected with Conax conditional access method and chipset pairing
- Video on Demand ECM decryption and sharing of programming protected with Conax conditional access method with chipset pairing
PROOF OF CONCEPT CODE

Control Words sharing (aka. signal theft)

- Sharing of the crypto keys used to descramble digital satellite TV programming

ECM SERVER
- Forwarding of plaintext Control Word values to subscribed client devices

SET-TOP-BOX A
- Extraction of plaintext Control Word values from STI71xx chip

SET-TOP-BOX B
- Use of plaintext Control Word Values for unauthorized program reception
- MPEG capture of descrambled HDTV signal
PROOF OF CONCEPT CODE

Push VOD sharing

- Sharing of the crypto keys used to descramble VOD movies
  - Obtaining Control Word for arbitrary movie during the rental period
    - VOD movies rented for 48 hours period
    - Encrypted MPEG data pushed into set-top-boxes
    - ECM messages accompanying movies files
  - Sharing Control Words after the rental period
  - Using plaintext CW values to descramble the movie
    - The use of key memory beyond index 0x32 on STi7100
    - The use of CWPK to reencrypt Control Word on STi7111
Summmary

STi7100 / STi7111 Security vulnerabilities

- How come the issues were not discovered before the market release?
  - STMicroelectronics a major silicon vendor
    - #1 in Europe, #7 in the world (source: Wikipedia)
  - Conax security evaluation of STB / CAM / DVB chipset solutions
    - Final scoring of STi7100/STI7111 not disclosed to Security Explorations
SUMMARY
Vulnerabilities Impact

- No information from STMicroelectronics (DVB chipsets vendor) in response to the impact inquiry questions
  - All your inquiries, as listed below, are pointing towards confidential information and as such can not be disclosed by ST to you or to others.
    - Jan-17-2011, STMicroelectronics in an email to Security Explorations
- Impact estimation upon publicly available data
SUMMARY

Vulnerabilities Impact (2)

- Cumulative MPEG-2 & MPEG-4 Shipments in 2008
  - 541 millions of units
    - Set-top-boxes, digital television sets, DVD / Bluray players
- STMicroelectronics #1 in H.264 market (68% of market share in 2008)
- Customers from Europe, Middle East and Africa, Asia-Pacific and the Americas
  - DishTV (India)
  - DirectTV (USA)
  - Platforma N, Cyfrowy Polsat (Poland)
  - BSkyB (UK)
  - ...
First successful attack against the implementation of a Conax conditional access system with chipset pairing
- Pay TV piracy possible in the environment of hacked digital satellite TV set-top-boxes
- Security of dedicated DVB chipsets broken

Security based on a complex, secret functionality embedded in a silicon is a dangerous concept
- Security through Obscurity?

The need to improve security and evaluation processes by silicon and CAS vendors

The need to tighten set-top-boxes security relying on vulnerable DVB chipsets
Thank You

contact@security-explorations.com