IDEAS REGARDING VULNERABILITIES IN ST DVB CHIPSETS

SE-2011-01

[Security weaknesses in a digital satellite TV platform]

DISCLAIMER

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER SECURITY EXPLORATIONS, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE INFORMATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY SECURITY EXPLORATIONS OR BY ANY OTHER PARTY THAT THE INFORMATION CONTAINED IN THE THIS DOCUMENT WILL MEET YOUR REQUIREMENTS OR THAT IT WILL BE ERROR-FREE.

YOU ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION AND USE OF THE INFORMATION TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SECURITY EXPLORATIONS, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF SECURITY EXPLORATIONS OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

INTRODUCTION

This document presents three ideas related to our security research of STMicroelectronics chipsets conducted as part of SE-2011-01 project [1]. The first one is an idea for a crypto attack leading to the potential exposure of CWPK / CW key. The other one is an idea for a potential CW extraction from PTI core. The final idea is about a possible detection of rogue cards abusing vulnerabilities in ST chipsets for plaintext CW extraction and arbitrary content distribution over the Internet.

The ideas presented in this document have not been verified in practice or fully explored by Security Explorations. Regardless of the above, we still see a value in disclosing them. The reasons are twofold.

First, we do hope all interesting parties (chipset / set-top-box / CAS vendors and security researchers in particular) find some of them useful or simply inspiring while dealing with security of ST chipsets and their vulnerabilities' impact. These vulnerabilities are still a mystery to many and we keep receiving inquiries about them regardless of the fact that almost 6 years had passed since the disclosure. STMicroelectronics, although out of set-to-box and DVB chipset business [2], has not provided us with any details regarding the impact of the issues found (no response to our last inquiry from 11-Apr-2017 [3] when we asked for a list of ST chipsets that were vulnerable to the issues found and reported as part of SE-2011-01 project). This is disrespectful not only to reporters of the issues, but primarily to the customers and all parties interested to fully understand the impact of ST flaws and seeking for an answer whether the vulnerable component (such as TKD core of STi7111) is present in other solutions[1] (i.e. set-top-boxes, digital television sets[2], DVD and Bluray players).

Second, presented ideas are just a few of the many that needed to be explored while investigating security of a modern digital SAT TV platform[3]. They however perfectly illustrate the complexity of security and threat models associated with a SAT TV ecosystem where co-existence of security solutions from many software and hardware vendors along with their closed nature does not necessarily fit together from a security point of view. Their operation as a whole is rarely fully understood by arbitrary TV operators who likely have no choice than to blindly trust the PR slogans of various security providers and their claims that "independent auditors[4] call [these solutions] more effective against pirate attacks than many competitors' smartcard-based solutions" [4].

POTENTIAL CRYPTO ATTACK AGAINST ST CHIPSETS

As a result of SE-2011-01 research, direct access to ST chipsets functionality could be gained and arbitrary crypto (TDES or AES) operations could be performed by attackers on chosen data blocks at will.

SLIM Core processors implement hardware support for both TDES and AES operations. Selection of a target cipher can occur either at:

[1] STi7100 and STi7111 are still in use by NC+ Platform in Poland (main chipsets of "the refurbished" ITI-5800S, ITI-5800SX, ITI-2849ST and ITI-2850ST set-top-boxes that NC+ still offers to its customers along the new boxes).
[2] we confirmed STi7111 to be the CPU used by certain Samsung SmartTVs.
[3] the disclosure and publications from 2012 were all limited to discovered and confirmed vulnerabilities only.
[4] as in the case of Conax security certifications awarded by "renown security evaluation laboratories", the auditors are usually anonymous.

1) device driver level by the means of AES_NOT_TDES bit of SLIM Core TK_CONFIG (0x4018) register:

```
l_01db  0x00d00090   sync
  01dc  0x00a30006   ld r3,[r0,0006] // 0x4018 = 0x00000000       ;TK CONFIG register
  01dd  0x00733c20   bitval r3,r3,#00000001                      ;AES_NOT_TDES bit
                                                                 ;(bit 1 of configureTK)
  01de  0x009c11e8   jne,s l_01e8                                ;-> jump for AES_NOT_TDES
  01df  0x004cc000   and r12,r12,#0000                           ;setup TDES
  01e0  0x00709c27   tst r9,#00000080                            ;TDES bit value
  01e1  0x009c11e8   jne,s l_01e8
  01e2  0x00d00090   sync
  01e3  0x00a30006   ld r3,[r0,0006] // 0x4018 = 0x00000000       ;TK CONFIG register
  01e4  0x00733c25   bitval r3,r3,#00000020
  01e5  0x00793027   bitset r9,r3&0x01<<7                        ;bit 0x80 of TKD command
  01e6  0x00d01e18   jmp l_01e8
```

2) SLIM Core firmware level by the means of executing dedicated SLIM Core instructions (copAES and copTDES[5]):

```
################################################
SUB l_04fd
initialization of a single crypto key
INPUT: r9 = 1 for AES
          = 0 for TDES
################################################

l_04fd  0x00409900   tst r9,00                                    ;AES ?
  04fe  0x008c1506   jne l_0506                                   ;-> jump for AES
  04ff  0x00fa4000   copTDES                                      ;handle TDES
  0500  0x000f083c   mov r15,r8                                   ;TK CMD
  0501  0x008e1501   wait1
  0502  0x00d00004   rpt 4
  0503  0x000f003c   mov r15,r0                                   ;CBUF <- 0
  0504  0x008e1504   wait1
  0505  0x008c050c   j l_050c

l_0506  0x00f54000   copAES                                       ;handle AES
  0507  0x000f083c   mov r15,r8
  0508  0x008d8508   wait2
  0509  0x00d00004   rpt 4
  050a  0x00af0000   ld r15,[r0,0000] // 0x4000 = 0x03740312
  050b  0x008d850b   wait2

l_050c  0x00d00004   rpt 4
  050d  0x00000f3c   mov r0,r15                                   ;IN <- rpt 4 []
  050e  0x00840d00   jmp r13                                      ;ret from sub
```

Before we discovered security issues in ST chipsets, we thought about investigating the possibility to break CWPK key (or even CWs themselves) by discovering a correlation of the results of TDES / AES crypto operations conducted in parallel for specially chosen data blocks (same data blocks fed at the input to both ciphers).

This would in particular require crypto / differential analysis of the results of TDES / AES cipher block operations, verifying whether any correlation between the input and output bits, S-boxes or P-boxes for both ciphers exists when done for same input in parallel, which could be exploited for the CWPK key extraction (i.e. single bit by bit extraction, etc.).

[5] these and other SLIM Core opcode names may not necessarily correspond to the real names of SLIM Core CPU instructions. These names were assigned by Security Explorations in a way, so that they were intuitive (self-descriptive and resembling names of similar instructions from other CPU architectures).

Our initial approach assumed the analysis with the representation of TDES / AES ciphers as boolean functions corresponding to their parallel operation.

The crypto attack was treated as some potentially interesting and likely novel idea (we did not find any publication on the topic at the time of the research), but also the attack of the very last resort (if everything else fails). The described potential crypto attack has not been pursued further for the following reasons:

- we lack proper expertise in breaking real life crypto ciphers and cryptoanalysis in particular,
- some other way to successfully compromise security of ST chipsets was discovered (Issues 17-19 [5]).

POTENTIAL CW EXTRACTION FROM PTI CORE

As a result of SE-2011-01 research, direct access to PTI core of both STi7100 and STi7111 processors could be gained. This core is responsible for handling MPEG transport streams, their filtering, descrambling and dispatch.

We were able to access the code of PTI firmware (embedded in and initialized by `ptiinit.ko` device driver). This code implemented an unknown CPU instruction set, with multiple hints pertaining to its nature embedded in `stpti4_core.ko` device driver and `/proc` file system in particular:

```
box> go 1
box> id
uid=555(stb) gid=10(stb)
box> root
uid=0(root) gid=0(root)
box> cd /proc/stpti4_core/PTI_0_0
box> cat STPTI_TCParameters_t
PTI_VERSION: STPTI4_DVB-REL.8.0.1
 TC_VERSION: STPTI4 803 (TC_ID is 31)
TC_CodeStart                  : 0x00002f88
TC_CodeSize                   : 0x00002f88
TC_DataStart                  : 0xfe238000
TC_LookupTableStart           : 0xfe238000
TC_GlobalDataStart            : 0xfe23850c
TC_StatusBlockStart           : 0xfe238560
TC_MainInfoStart              : 0xfe238598
TC_DMAConfigStart             : 0xfe238e98
TC_DescramblerKeysStart       : 0xfe239578
TC_TransportFilterStart       : 0xfe239c1c
TC_PESFilterStart             : 0xfe239cdc
TC_SubstituteDataStart        : 0xfe2381c8
TC_SFStatusStart              : 0xfe239cdc
TC_InterruptDMAConfigStart    : 0xfe23b0dc
TC_EMMStart                   : 0xfe23b2a4
TC_ECMStart                   : 0xfe23b2a4
TC_MatchActionTable           : 0x00000000
TC_SessionDataStart           : 0xfe23b0ec
TC_NumberCarousels            : 1
TC_NumberDMAs                 : 55
TC_NumberDescramblerKeys      : 25
TC_NumberIndexs               : 96
```

```
TC_NumberPesFilters           : 0
TC_NumberSectionFilters       : 128
TC_NumberSlots                : 96
TC_NumberTransportFilters     : 0
TC_NumberEMMFilters           : 0
TC_NumberECMFilters           : 96
TC_NumberOfSessions           : 5
TC_NumberSCDFilters           : 24
TC_AutomaticSectionFiltering  : FALSE
TC_MatchActionSupport         : FALSE
TC_SignalEveryTransportPacket : TRUE
box> cat TCDevice_t
TCDevice_t @ address 0xfe230000:
PTIIntStatus0:0x00000000   PTIIntStatus1:0x00000000   PTIIntStatus2:0x00000000
PTIIntStatus3:0x00000000
PTIIntEnable0:0x00000003   PTIIntEnable1:0x00000000   PTIIntEnable2:0x00000000
PTIIntEnable3:0x00000000

TCMode       :0x00000001   DMAempty_STAT:0x00000000   DMAempty_EN  :0x00000000
TCPaddng_0   :0x00000000

PTIAudPTS_31_0: 0x00000000   PTIAudPTS_32:0x00000000
PTIVidPTS_31_0: 0x00000000   PTIVidPTS_32:0x00000000

STCTimer0     : 0x012864c3   STCTimer1   :0x00000000

DMAs:
0 - Base:0x41f1f000 Top:0x420f4fff Write:0x42039234 Read:0x420366e0
Setup:0x00000008 DMA0Holdoff:0x00000000 Status:0x00000000
1 - Base:0x00000000 Top:0x00000000 Write:0x00000000 Read:0x00000000
Setup:0x00000000 DMA0Holdoff:0x00000000 CDAddr:0x00000000
2 - Base:0x00000000 Top:0x00000000 Write:0x00000000 Read:0x00000000
Setup:0x00000000 DMA0Holdoff:0x00000000 CDAddr:0x00000000
3 - Base:0x00000000 Top:0x00000000 Write:0x00000000 Read:0x00000000
Setup:0x00000000 DMA0Holdoff:0x00000000 CDAddr:0x00000000

Enable:0x00000001   SecStart:0x41b7d942   Flush:0x00000000   PTI3Prog:0x00000000

IIFCAMode :0x00000001
IIF 0xfe236000

FIFOCount :0x0000005c AltFIFOCount:0x00000000 FIFOEnable:0x00000001

AltLatency:0x00000000   SyncLock    :0x00000000   SyncDrop  :0x00000000
SyncConfig:0x00000001

SyncPeriod:0x000000c2

TCRegA :0x0001   TCRegB :0x0131   TCRegC :0x003f   TCRegD :0x0002
TCRegP :0x8580   TCRegQ :0x8038   TCRegI :0x0012   TCRegO :0x0002
TCRegE0:0x0000   TCRegE1:0xb144   TCRegE2:0x4203   TCRegE3:0x483f
TCRegE4:0x5c31   TCRegE5:0x0002   TCRegE6:0x0094   TCRegE7:0xb2f0
TCIPtr :0x0108 (FreeRunning)
```

PTI's main responsibility includes transport stream handling and deciphering. Due to its location in the ST SoC (close to PDES - crypto core) as illustrated in Fig. 1 there exists a potential possibility that plaintext CW keys could be accessed from PTI core.

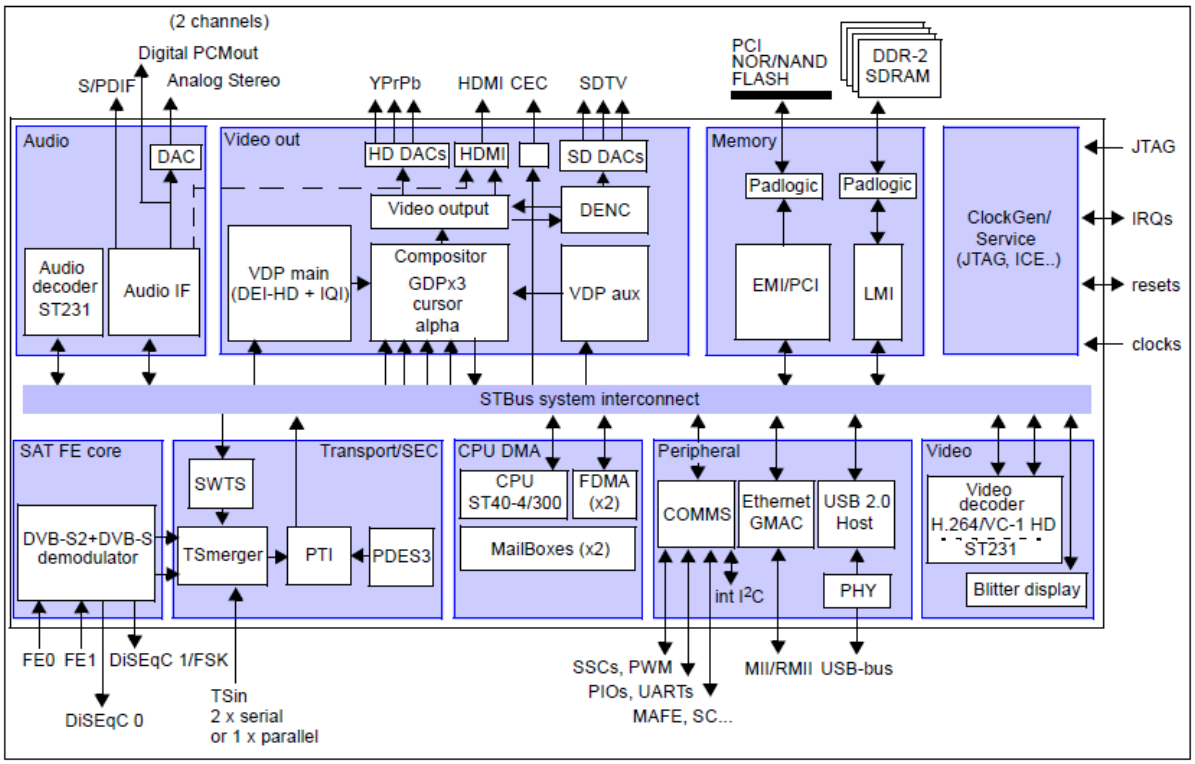

Fig. 1 STi7111 single-chip, high-definition STB decoder (source: st.com).

As a result of a more detailed analysis of the PTI device driver code, we figured out the purpose of various PTI memory locations:

```
0000-00c0 : TC_LookupTableStart SlotLUT (96 entries, 2 bytes in size each)
00c0-     : current MPEG packet
       c0    - x (usually 0)
       c1    - sync_byte = 0x47
       c2    - transport_error_indicator, payload_unit_start_indicator,
               transport_priority, PID (high nibble)
       c3    - PID (low nibble)
       c4    - x
       c5    - transport_scrambling_control, adaptation_field_control,
               continuity_counter
       c6/c8 - data bytes start
01a8-     : TC_SubstituteDataStart
026c-     : TC_GlobalDataStart
02a8-     : TC_StatusBlockStart
02cc-0a4c : TCMainInfo_t (96 entries, 20 bytes in size each)
0a4c-1050 : TC_DMAConfigStart (55 entries, 28 bytes in size each)
1050-11b8 : TC_DescramblerKeysStart (18 entries, 20 bytes in size each)
11b8-     : TC_SFStatusStart
18b8-     : TC_InterruptDMAConfigStart
18c8-1988 : TC_SessionDataStart (3 entries, 64 bytes in size each)
1988-     : TC_EMMStart / TC_ECMStart
```

The above confirmed that PTI core made use of descrambling keys (it maintained dedicated memory area associated with them).

We further investigated the PTI core of STi7100 and STi7111 chipsets in order to discover its interaction with a descrambler component. For that purpose, we implemented a simple STPTI tracer

in our main SE-2011-01 Proof of Concept code (APPENDIX A). The tracer functionality was implemented with the use of a single instruction stepping by enforcing a `TC_SINGLE_STEP` bit in `PTI_TC_MODE` (offset 0x30) configuration register.

The tracer made it possible for us to inspect the operation of PTI core (trace execution flow of single instructions, see instruction's induced changes to PTI registers and memory locations). In its initial, simple form, the tracer did not make it possible to inject arbitrary PTI firmware instructions as in SLIM Core case (TKD tracer tool). However, it was sufficient for us to nail down the location of PTI firmware location (and opcode) likely responsible for a descrambler key (CW) access conducted as a response to handling of a given, scrambled MPEG TS packet:

```
TCIPtr  1d94 opcode 4440052b
 TCRegA  0000  TCRegB  00f6  TCRegC  9050  TCRegD  0001
 TCRegP  9064  TCRegQ  0000  TCRegI  0099  TCRegO  0000
 TCRegE0 831c  TCRegE1 98c8  TCRegE2 0477  TCRegE3 c507
 TCRegE4 03a8  TCRegE5 0000  TCRegE6 20bc  TCRegE7 99d4

 PACKET DATA
size: 000000bc
       0000:  00 47 1b 02 00 99 88 47 8a 16 a6 0b 02 6f 91 19  .G.....G.....o..
       0010:  a1 d3 88 47 99 16 f8 26 02 cd 88 47 ae 16 e0 d9  ...G...&...G....
       0020:  02 3a 9e 1a 88 47 c4 16 09 d3 02 1f d5 1a 88 47  .:...G.........G
       0030:  da 16 37 cc 02 0c 88 47 02 17 3a 13 02 6c 91 18  ..7....G..:..l..
       0040:  88 47 19 17 54 b2 02 37 08 63 88 47 28 17 9e cd  .G..T..7.c.G(...
       0050:  88 47 40 17 bc 6c 02 18 d6 18 05 68 88 47 4f 17  .G@..l.....h.GO.
       0060:  05 88 02 1f 88 47 a0 17 00 8f 02 9f 9b 1c c3 30  .....G.........0
       0070:  70 1b 88 03 03 5c 30 31 3a b4 1e 2e d0 42 98 73  p.....01:....B.s
       0080:  49 e1 e1 d5 88 47 a3 17 22 4e 02 8f 98 18 a0 37  I....G.."N.....7
       0090:  e1 ea d6 92 e9 8c cf 2a 75 80 66 e9 e1 7e 9a 2c  .......*u.f....,
       00a0:  2a b5 43 3a f9 4d f4 b1 e2 03 2c e7 1c 63 a2 f0  *.C:.M....,..c..
       00b0:  96 c8 fd d2 d0 ed 04 66 03 db 60 cc  .......f....

 TCIPtr  1d98 opcode 56c00504
 TCRegA  0000  TCRegB  00f6  TCRegC  9050  TCRegD  0001
*TCRegP  9066  TCRegQ  0000  TCRegI  0099  TCRegO  0000
 TCRegE0 831c  TCRegE1 98c8  TCRegE2 0477  TCRegE3 c507
 TCRegE4 03a8 *TCRegE5 0303  TCRegE6 20bc  TCRegE7 99d4
```

In the example above, `TCRegE5` contains data associated with the first word of a descrambler key (`Vldt` field) and `TCRegP` holds the address of a PTI memory location pointing 2 bytes past the read PTI memory location (0x1064+base offset 0x8000). In the tests conducted, this did not seem to immediately reveal the plaintext value of a real CW key though:

```
#       Vldt Mode Evn3 Evn2 Evn1 Evn0 Odd3 Odd2 Odd1 Odd0
1050:  0303 0000 00f0 0000 0000 0000 00d0 0000 0000 0000
1064:  0303 0000 00f0 0000 0000 0000 00d0 0000 0000 0000
```

It looked as if key contents held in PTI's memory location pointed by `DescramblerKeysStart` address were offsets to some other memory location (such as a descrambler memory), which might have been used by the PTI DMA engine or a descrambler itself.

Similarly to the first idea, we haven't further investigated the operation of PTI core as some other way to successfully compromise security of ST chipsets was discovered (Issues 17-19). Taking into account the functionality of PTI component, its complexity (device driver binary is 250KB+ in size),

SoC location, interaction with a descrambler and use across various ST chipset generations, PTI seems to be a primary target for any further security investigation of DVB chipsets from STMicroelectronics for all concerned parties.

One also needs to keep in mind, that initial tests with TKD / Crypto cores of STi7100 and STi7111 chipsets didn't reveal the actual memory values corresponding to plaintext CW keys as well. This turned out to be possible by discovering a specially crafted memory access sequence though (vulnerabilities / backdoor in SoC hardware implementation).

ROGUE SUBSCRIBER DETECTION / DEACTIVATION AT CONTENT DISTRIBUTION LEVEL

The vulnerabilities in ST chipsets we exposed in 2012 made it possible to extract plaintext values of CWs.

Some of these could be potentially exploited for arbitrary deciphering and content distribution over the Internet. In such a case, a given rogue chipset and paired smartcard are used.

The idea behind detecting and deactivating the card address of such a rogue subscriber assumes the following functionalities exists in a target CAS environment:

1) the ability to send messages to specific group of subscribers (the identification of a target subscriber by an address composed of both card address field and a mask),

2) the ability to "black out" out the signal for a given subscriber (make sure it is not able to decipher AV content broadcasted at a given time period),

3) the ability for a monitoring of the illegal content distribution service in the Internet (the reception of the AV signal or plaintext CWs for which the original card / set of cards are to be detected and deactivated).

We believe that all of the above functionalities exist in most CAS environments. More specifically, condition 2 should be possible by the means of the RSA key change EMM message.

Taking the above assumptions into account, one could think of the following detection scheme described in a context of a sample set of 256 subscribers (the input of card addresses set to 0-255).

1) the operator broadcasts a message to the first half (0-127) of the input range addresses (RSA key change) making sure that they are not able to decipher any content for a given 10s period (CW lifetime period),

2) the operator verifies whether the content distributed by a rogue subscriber over the Internet is "blacked out",

3) if the content is "blacked out" the operator knows that a rogue card is in the first half of a given input range, the operator proceeds with step 1 and a new input range is set to the matched half of addresses (0-127),

4) if the content is not "blacked out" the operator knows that a rogue card is in the second half of a given input, the operator proceeds with step 1 and a new input range set to the unmatched half of addresses (128-255),

5) the process continues following the binary search fashion until a single address is detected (the rogue card that is used to distribute content into the Internet).

The advantage of the presented process is accuracy and speed. The rogue subscriber can be detected in logarithmic time:

$$T_{detection} = T_{blackout\ period} * Log_2 N$$

where N = subscribers' base address range

For 32 bit N and blackout period equal to 10s, the whole detection process could be performed within 320s (5 minutes). Upon detection, the rogue card could be temporarily or permanently deactivated (without any notification to the subscriber - any such a notification is an unnecessary hint to the attacker regarding the cause of improper card operation, it does help attackers plan for / avoid such countermeasures in the future).

The other advantage is that all rogue cards used for the distribution of a given content set can be also detected (if several cards are used to distribute several channels from a given distribution source).

The whole process could be arranged in a less predictable way as well in order to make it impossible for the attacker to figure out the nature of the short "black outs" (strange, short floods of EMM messages), the sequences of the message received by the cards, etc.

FINAL WORDS

Digital SAT TV ecosystem is truly strange. It's like another world in information security. Its members are more likely to come and ask about 3rd parties contacting a security research outfit over security vulnerabilities disclosed by it in DVB chipsets rather than to pursue work aimed at improving security of their own solutions (IRDETO). If by any chance the topic of a mutual cooperation emerges, this all seems to be conditional as in STMicroelectronics' case (a proposal of a business, publicly acknowledged cooperation in exchange for a limited / no vulnerability disclosure). They hardly respond to inquiries regarding reported vulnerabilities and never provide information about affected products or resolution of the issues (Advanced Digital Broadcast, STMicroelectronics). They attempt to initiate business talks under NDA potentially limiting the planned disclosure (ITI Neovision). They do not reveal the results of security evaluations conducted by "renown security evaluation laboratories" for compromised products (STBs and DVB chipsets), so these could be confronted with the findings of independent entities such as ours (Conax). They do not seem to appreciate the fact that a discoverer of critical issue in their products agrees to travel to a foreign location and conducts for them, completely for free, a dedicated technical presentation explaining the nature of the attacks, methodologies and tools used - they do not bother to offer any reimbursement for the costs of an air trip / hotel stay for the party simply doing a courtesy to them (STMicroelectronics). They bring lawyers to meetings (STMicroelectronics). Finally, they often express carefully worded

statements about damages and harm any planned disclosure of the issues found could do to the whole digital SAT TV ecosystem (STMicroelectronics).

From the above, one may have a slightly better understanding of what it takes to be part of information security field these days and what challenges an independent security outfit may face in particular. It is not easy to handle all of this for a one man shop such as ours.

The more important it is to expose security issues in an ecosystem like the one described above, keep challenging the vendors of closed security solutions that are part of it, do not adhere to and oppose to their methods.

REFERENCES

[1] SE-2011-01 Security weaknesses in a digital satellite TV platform
http://www.security-explorations.com/en/SE-2011-01.html
[2] STMicro to exit STB chip business
https://www.broadbandtvnews.com/2016/01/27/losses-force-stmicro-to-end-stb-chip-business/
[3] SE-2011-01 Vendors status
http://www.security-explorations.com/en/SE-2011-01-status.html
[4] NAGRA and Canal+ Group renew and expand content protection partnership
https://www.nagra.com/media-center/press-releases/nagra-and-canal-group-renew-and-expand-content-protection-partnership
[5] SE-2011-01 Issues #17-19
http://www.security-explorations.com/materials/se-2011-01-st.pdf

---

About Security Explorations

Security Explorations (http://www.security-explorations.com) is a security start-up company from Poland, providing various services in the area of security and vulnerability research. The company came to life in a result of a true passion of its founder for breaking security of things and analyzing software for security defects. Adam Gowdiak is the company's founder and its CEO. Adam is an experienced Java Virtual Machine hacker, with over 100 security issues uncovered in the Java technology over the recent years. He is also the Argus Hacking Contest co-winner and the man who has put Microsoft Windows to its knees (the original discoverer of MS03-026 / MS Blaster worm bug). He was also the first one to present successful and widespread attack against mobile Java platform in 2004.

APPENDIX A

The code of STPTI Java language class implementing basic PTI core tracing functionality.

```
/*## (c) SECURITY EXPLORATIONS    2011 poland                                  #*/
/*##     http://www.security-explorations.com                                  #*/

/* RESEARCH MATERIAL:     SE-2011-01                                           */
/* [Security weaknesses in a digital satellite TV platform, STPTI tracer]      */

/* THIS SOFTWARE IS PROTECTED BY DOMESTIC AND INTERNATIONAL COPYRIGHT LAWS     */
/* UNAUTHORISED COPYING OF THIS SOFTWARE IN EITHER SOURCE OR BINARY FORM IS    */
/* EXPRESSLY FORBIDDEN. ANY USE, INCLUDING THE REPRODUCTION, MODIFICATION,     */
/* DISTRIBUTION, TRANSMISSION, RE-PUBLICATION, STORAGE OR DISPLAY OF ANY       */
/* PART OF THE SOFTWARE, FOR COMMERCIAL OR ANY OTHER PURPOSES REQUIRES A       */
/* VALID LICENSE FROM THE COPYRIGHT HOLDER.                                    */

/* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS     */
/* OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,*/
/* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL     */
/* SECURITY EXPLORATIONS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, */
/* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF   */
/* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE      */
/* SOFTWARE.                                                                   */

import java.lang.*;
import java.io.*;

public class STPTI {
 public static final int HERMES_BASE = 0xb9230000;
 public static final int CARBO_BASE  = 0xfe230000;

 public static final int TCRegA   = 0x0000;
 public static final int TCRegB   = 0x0004;
 public static final int TCRegC   = 0x0008;
 public static final int TCRegD   = 0x000c;

 public static final int TCRegP   = 0x0010;
 public static final int TCRegQ   = 0x0014;
 public static final int TCRegI   = 0x0018;
 public static final int TCRegO   = 0x001c;

 public static final int TCIPtr   = 0x0020;
 public static final int TCRegE0  = 0x0024;
 public static final int TCRegE1  = 0x0028;
 public static final int TCRegE2  = 0x002c;
 public static final int TCRegE3  = 0x0030;

 public static final int TCRegE4  = 0x0034;
 public static final int TCRegE5  = 0x0038;
 public static final int TCRegE6  = 0x003c;
 public static final int TCRegE7  = 0x0040;

 public static final int TCRegNum = 0x11;

 public static final int PTI_TC_MODE    = 0x30;
 public static final int PTI_TC_REGS    = 0x7000;

 public static final int TC_EN          = 0x01;
 public static final int TC_RST_IPTR    = 0x02;
```

```
public static final int TC_SINGLE_STEP = 0x04;

private static int base;

public static int regs[];
public static boolean changed[];

public static int code[];
public static int ptr;

public static final String regnames[]={
 "TCRegA ", "TCRegB ", "TCRegC ", "TCRegD ", "TCRegP ", "TCRegQ ", "TCRegI ",
 "TCRegO ", "TCIPtr ", "TCRegE0", "TCRegE1", "TCRegE2", "TCRegE3", "TCRegE4",
 "TCRegE5", "TCRegE6", "TCRegE7"
};

static {
 try {
  if (Config.isHERMES()) {
   base=HERMES_BASE;
  } else

  if (Config.isCARBO()) {
   base=CARBO_BASE;
  }

  regs=new int[TCRegNum];
  changed=new boolean[TCRegNum];
  code=new int[15000];


  ClassLoader cl=Thread.currentThread().getContextClassLoader();

  //tc_firmware.dat needs to be extracted from ptiinit.ko driver
  InputStream is=cl.getResourceAsStream("tc_firmware.dat");

  int pos=0;

  while(true) {
   int v1=is.read();
   if (v1<0) break;
   int v2=is.read();
   int v3=is.read();
   int v4=is.read();

   int v=((v4&0xff)<<24)|((v3&0xff)<<16)|((v2&0xff)<<8)|(v1&0xff);
   code[pos]=v;
   pos+=4;
  }

  ptr=NativeCode.getInstance().malloc(0x15000);
  ApiMonitor.log("STPTI inited");
 } catch(Throwable t) {}
}

public static void disable() {
 ApiMonitor.log("STPTI disable");
 IOMem.out_dword(base+PTI_TC_MODE,0);
 Utils.sleep(100);
}
```

```
public static void enable() {
 ApiMonitor.log("STPTI enable");
 IOMem.out_dword(base+PTI_TC_MODE,TC_EN);
 Utils.sleep(100);
}

public static void reset() {
 ApiMonitor.log("STPTI reset");
 IOMem.out_dword(base+PTI_TC_MODE,0);
 Utils.sleep(100);

 IOMem.out_dword(base+PTI_TC_MODE,TC_RST_IPTR);
 Utils.sleep(100);

 IOMem.out_dword(base+PTI_TC_MODE,0);
 Utils.sleep(100);

 while(read_TCIPtr()!=0) {
  Utils.sleep(100);
 }
}

public static void trace() {
 IOMem.out_dword(base+PTI_TC_MODE,TC_EN|TC_SINGLE_STEP);

 while((IOMem.in_dword(base+PTI_TC_MODE)&TC_EN)!=0) {
 }
}

public static int read_TCRegA() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegA);
}

public static int read_TCRegB() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegB);
}

public static int read_TCRegC() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegC);
}

public static int read_TCRegD() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegD);
}

public static int read_TCRegP() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegP);
}

public static int read_TCRegQ() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegQ);
}

public static int read_TCRegI() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegI);
}

public static int read_TCRegO() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegO);
}
```

```
public static int read_TCIPtr() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCIPtr);
}

public static int read_TCRegE0() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE0);
}

public static int read_TCRegE1() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE1);
}

public static int read_TCRegE2() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE2);
}

public static int read_TCRegE3() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE3);
}

public static int read_TCRegE4() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE4);
}

public static int read_TCRegE5() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE5);
}

public static int read_TCRegE6() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE6);
}

public static int read_TCRegE7() {
 return IOMem.in_dword(base+PTI_TC_REGS+TCRegE7);
}

public static int[] read_regs() {
 int regs[]=new int[TCRegNum];
 int addr=base+PTI_TC_REGS;

 for(int i=0;i<TCRegNum;i++) {
  regs[i]=IOMem.in_dword(addr);
  addr+=4;
 }

 return regs;
}

public static String rvalue(int idx) {
 String s;
 if (idx!=8) {
  if (changed[idx]) s="*";
   else s=" ";
 } else s=" ";

 s+=regnames[idx]+" "+Utils.hex_value(regs[idx],4);
 return s;
}

public static void print_regs() {
 int nregs[]=read_regs();
```

```
 for(int i=0;i<TCRegNum;i++) {
  if (regs[i]!=nregs[i]) changed[i]=true;
   else changed[i]=false;
  regs[i]=nregs[i];
 }

 int iptr=regs[8];
 String s=rvalue(8)+" opcode "+Utils.hex_value(code[iptr],8);

 ApiMonitor.log(s);

 s=rvalue(0)+" "+rvalue(1)+" "+rvalue(2)+" "+rvalue(3);
 ApiMonitor.log(s);

 s=rvalue(4)+" "+rvalue(5)+" "+rvalue(6)+" "+rvalue(7);

 ApiMonitor.log(s);

 s=rvalue(9)+" "+rvalue(10)+" "+rvalue(11)+" "+rvalue(12);
 ApiMonitor.log(s);

 s=rvalue(13)+" "+rvalue(14)+" "+rvalue(15)+" "+rvalue(16);
 ApiMonitor.log(s);

 ApiMonitor.log(" ");
}

public static void dump_mem(int off,int SIZE) {
 int addr=base+off;

 for(int i=0;i<SIZE>>2;i++) {
  int v=IOMem.in_dword(addr+4*i);
  Memory.write_dword(ptr+4*i,v);
 }

 byte tab[]=Utils.buf2barray(ptr,SIZE);

 ApiMonitor.log_buf(tab);
}

public static void dump_packet() {
 ApiMonitor.log("PACKET DATA");

 dump_mem(0x80c0,0xbc);
}

public static void run_trace(int ins_cnt) {
 disable();

 //dump PTI memory
 ApiMonitor.log("MEM DATA");
 dump_mem(0x0000,0x10000);

 //dump current TS packet
 dump_packet();

 print_regs();

 trace();

 int i=0;
```

```
  while(i<ins_cnt) {
   print_regs();
   dump_packet();
   trace();
   i++;
  }

  enable();
 }
}
```