SYNACKTIV

0-click RCE on Tesla Model 3 through TPMS Sensors

Hexacon 2024

October 4th 2024

Who are we

SYNACKTIV

David Berard
@_p0ly_

Thomas Imbert
@masthoon

Vincent Dehors
@vdehors

Synacktiv

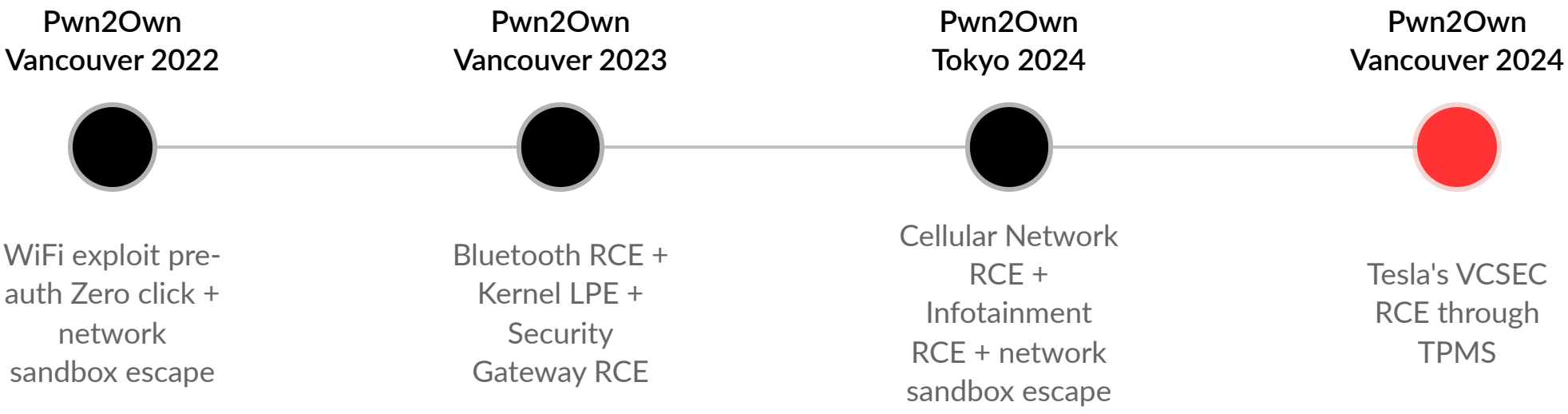
- Offensive security
- 170 Experts
- Pentest, Reverse Engineering, Development, Incident Response

Reverse Engineering team

- 50 reversers
- Low level research, reverse engineering, vulnerability research, exploit development, etc.

Synacktiv vs Tesla: Previous work

SYNACKTIV



Architecture

Architecture

VCSEC ECU

SYNACKTIV

Features

- Endpoint for the Tesla Mobile App
  - Open the car, start driving, basic remote control
- NFC
  - Open the car, start driving
- TPMS
  - Measure tires pressure and temperature

Connectivity

- Bluetooth Low Energy
  - Tesla App + TPMS sensors
- Ultra Wide Band
  - Tesla App
- Vehicule CAN
  - UDS for maintenance / provisionning
  - Standard CAN signals

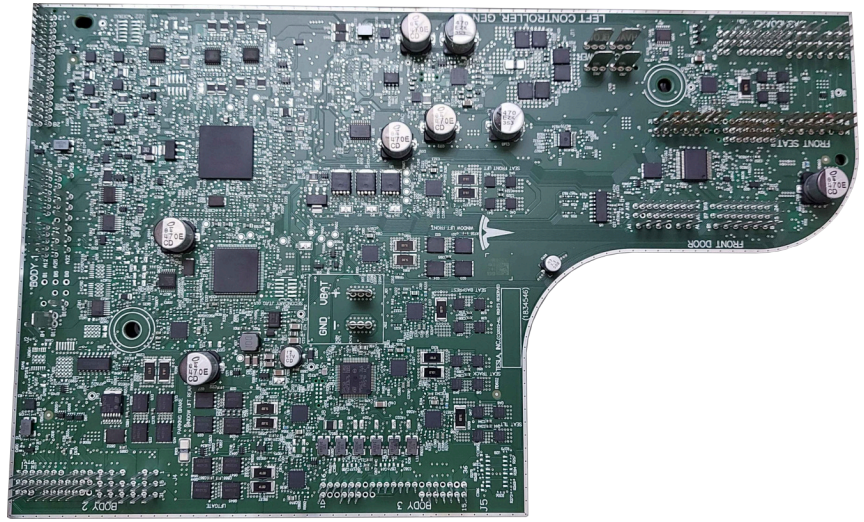
Architecture

VCSEC ECU

SYNACKTIV

Hardware

- Now embedded in VCLeft ECU, used to be a standalone ECU
- Still a dedicated SoC on this ECU
- **PowerPC SoC: SPC56 from ST**
  - run in **VLE** mode
- No BLE connectivity on the PCB
  - Multiple BLE Endpoints connected with UART to CAN transceivers



Architecture

SYNACKTIV

VCSEC ECU

Firmwares

- VCSEC is updated by the Security Gateway that fetches firmware from the infotainment
- **Present in the rootfs filesystem of the infotainment** for many board revisions
- Firmwares are not encrypted

Software

- Operating system: **FreeRTOS**
- Look to be Tesla code
- Standard librairies used: **Mbed-TLS**, **nanopb**

Reverse engineering

- `PPC VLE` is well supported by IDA Pro, a decompiler is available
- `PPC VLE` emulators are not widespread, qemu does not support it
- SPC56 SDK gives many useful information
  - Used version of FreeRTOS
  - Low level drivers, etc...

Architecture

VCSEC ECU

SYNACKTIV

Protocol

- BLE and UWB interfaces use Protobuf messages
- The .proto is shared between differents features (TPMS & Tesla Application for example)
- The size of the protobuf message is just prepended to the message
- Some projects on Github already extracted the .proto from the Tesla App

TeslaProtobufs / vcsec.proto

Code | Blame | 1219 lines (1071 loc) · 32.2 KB

```
307       bytes response = 1;
308     }
309
310     message FromVCSECMessage {
311       oneof sub_message {
312         VehicleStatus vehicleStatus = 1;
313         SessionInfo sessionInfo = 2;
314         AuthenticationRequest authenticationReq
315         CommandStatus commandStatus = 4;
316         PersonalizationInformation personalizat
```

TPMS

TPMS Sensor

Tire-Pressure Monitoring System

- Report real-time tire-pressure information
- Mandatory in new vehicles
- One for each wheel `FL/FR/RL/RR`

TPMS Sensor

Monitor and Alert

SYNACKTIV

⚠ Warn the user on any tire abnormalities

- Older `TPMS` used 433 MHz Radio for connectivity
- Now, `TPMS` leverages **B**luetooth **L**ow **E**nergy
- 5 *BLE* endpoints in Tesla cars
  - `Center/Left/Right/Rear/Rear Left`
  - Used to locate TPMS position

TPMS Sensor

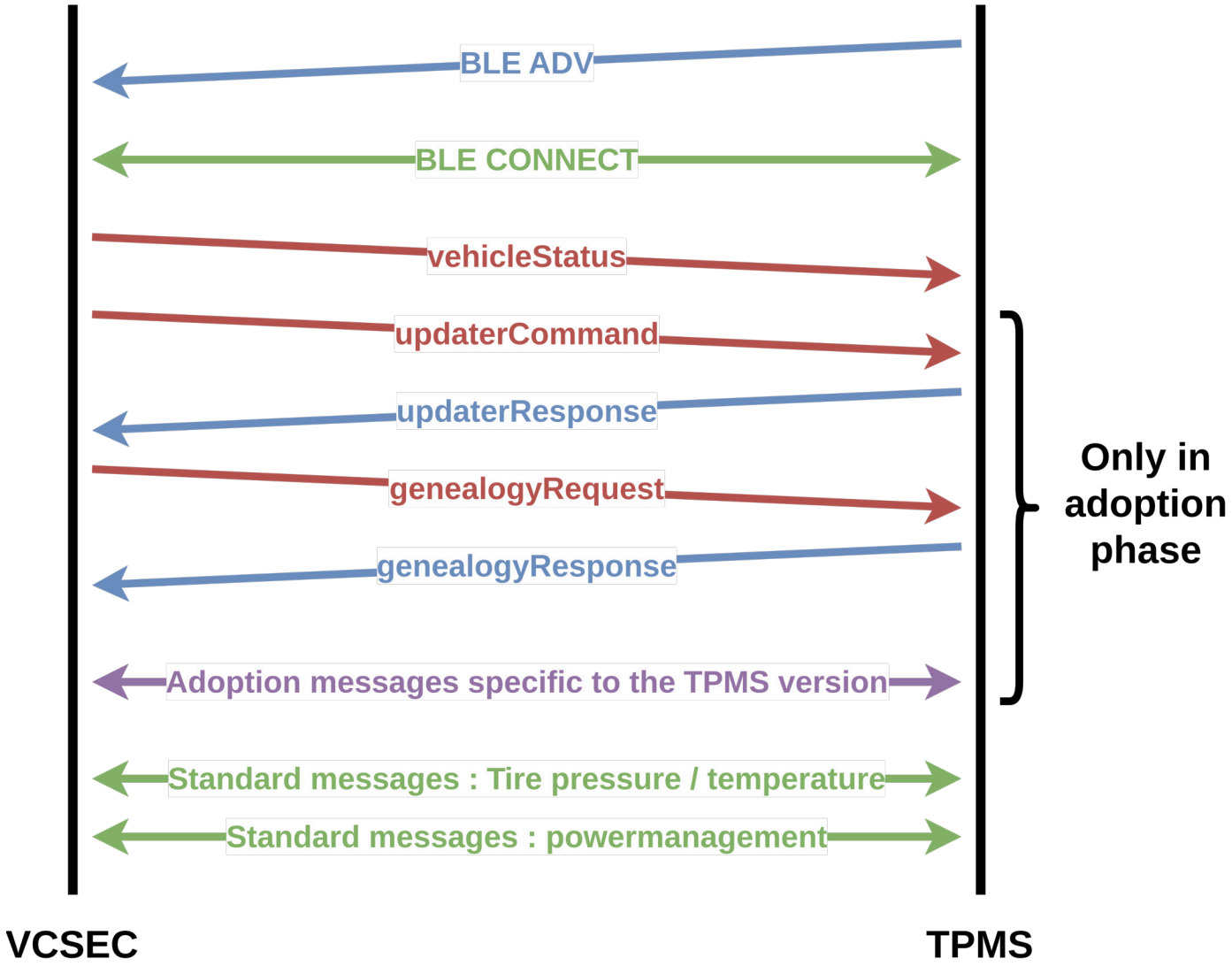
Connectivity

SYNACKTIV

- *Only BLE implementation studied*
- Messages exchanged over BLE GATT characteristics
  - `00000211-b2d1-43f0-9b88-960cebf8b91e` GATT service
  - But VCSEC does not use UUIDs but only BLE handles 😒
- Data serialized with Protocol Buffer

TPMS Sensor

Protocol



TPMS Sensor

Example of standard messages

SYNACKTIV

- Received from TPMS

```
// Received:
TPData {
    pressure: 101
    temperature: 22
}

// Received:
TPWheelUnitInfo {
    TIAppCRC: "[..]"
    MLXAppCRC: "[..]"
    batteryVoltage_mV: 3011
}
```

VCSEC Vulnerability and Exploitation

VCSEC

How to start reverse engineering

SYNACKTIV

The Plan

- Study why they are multiple firmwares of VCSEC
- Choose the firmware that matches the test setup and is valid for Pwn2Own
- Reverse engineering

Reality

- Picked firmware with the largest `number`
- Found vulnerability
  - Not present on other firmwares
- Understood the `number`, which is the hardware revision
- Asked the vendor (thanks Tesla!) for the right hardware and if it was valid for Pwn2Own
- Used half of research time to have a working setup

VCSEC

Model3 versions

SYNACKTIV

```
# Firmwares on the infotainment filesystem
$ cat signed_metadata_map.tsv |grep vcsec
vcsec:50397185  vcsec/7/UDSBoot-VCSEC-P_3-A_0-U_0-CONFIG_1704-GIT_AE006F26D00A5C6D.bhx
vcsec:117440513 vcsec/23/UDSBoot-VCSEC-P_7-CONFIG_700-GIT_8D34551F13E4371E.bhx
vcsec:134217729 vcsec/24/UDSBoot-VCSEC-P_8-CONFIG_702-GIT_3ACAF2AD323CEBCC.bhx
vcsec:50397185  vcsec/7/UDSBoot-VCSEC-P_3-A_0-U_0-CONFIG_1705-GIT_AE006F26D00A5C6D.bhx
vcsec:117440513 vcsec/23/UDSBoot-VCSEC-P_7-CONFIG_701-GIT_8D34551F13E4371E.bhx
vcsec:134217729 vcsec/24/UDSBoot-VCSEC-P_8-CONFIG_703-GIT_3ACAF2AD323CEBCC.bhx
vcsec:50397185  vcsec/7/VCSEC_ConfigID_7_crc_formatted_lithium-signed.bhx
vcsec:117440513 vcsec/23/VCSEC_ConfigID_23_crc_formatted_lithium-signed.bhx
vcsec:134217729 vcsec/24/VCSEC_ConfigID_24_crc_formatted_lithium-signed.bhx
```

Version analyzed: hw-id 134217729

- Seems to be used in recent Model 3 version ("highland" since October 2023)
- **VCSEC_ConfigID_24** is the main application code
- **VCSEC_ConfigID_23** (HW_ID 117440513) has a very similar code (don't know where is it used)

VCSEC

Reversing

SYNACKTIV

- IDA decompiler for `PPC VLE`
- Time consuming to reverse
  - ~1MB (3k functions)
  - No symbols
  - Large structures and function callbacks used everywhere
  - Not many strings

```
case 28:
  v30 = (int *)sub_108F382(connection);
  sub_1091C6E(v30, *((_WORD *)decoded + 3));
  sub_1091C8E(v30, *((_WORD *)decoded + 5));
  v31 = sub_109A590(&v102, (char *)&unk_400BE519, &dword_400ABC68, 0x28u, 0x20u, 0x1Bu, 12);
  sub_1091CAE((unsigned int)v30, v31, v102 == 0);
  sub_108E4F2((unsigned int)v30);
  sub_10B6948(10, "TPData from sensor %u", v30);
  sub_10B6948(10, "Pressure %u", *((_DWORD *)decoded + 1));
  sub_10B6948(10, "Temperature %d", *((_DWORD *)decoded + 2));
```

```
sub_10A7B5C();
sub_10A5B44();
sub_10A44D0();
nullsub_58();
if ( sub_10CC626() != 1 )
{
  sub_1043BAA();
  sub_1084E90();
  sub_104EA6A();
  sub_105049C();
  sub_1050316();
  sub_104FCDC();
  sub_1084D00();
  sub_106C3CE();
  sub_1095C10();
  sub_1085FB6();
  sub_1069718();
  sub_10967BC();
  sub_10C8782();
  sub_1063DDE();
  sub_1054750();
  sub_105AD7C();
  sub_1059BBA();
  nullsub_25();
  sub_106BE10();
  sub_1088D06();
  sub_1089786();
  sub_1088BB2();
  sub_1087B58();
  sub_108868C();
  sub_1087366();
  sub_1087F66();
  sub_10877F0();
  sub_1086288();
  sub_1088B18();
  sub_1087CFA();
  sub_10859BE();
  sub_107E5C6();
  sub_107968C();
  sub_10786C0();
  sub_107A1CC();
  sub_107A798();
  sub_107D878();
  sub_105272C();
  sub_106D314();
  sub_10440C8();
```

Main task

VCSEC

Reversing Protobuf messages

SYNACKTIV

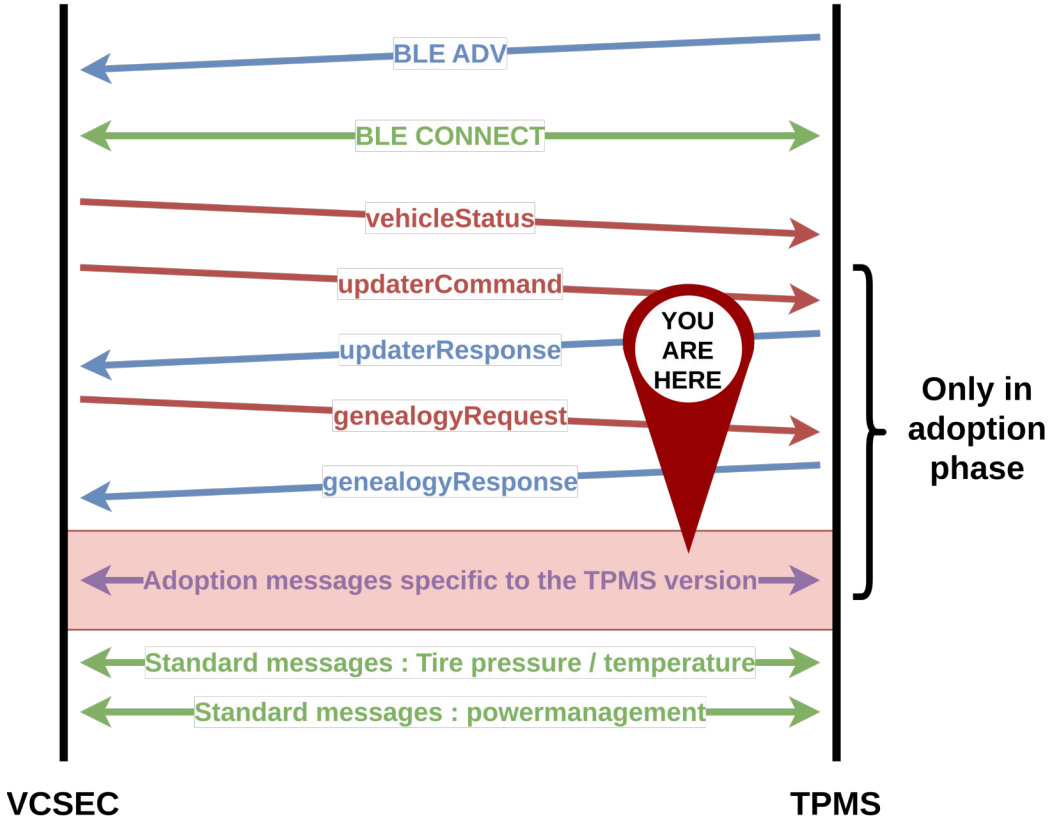
- Used `anvilsecure/nanopb-decompiler` to retrieve Protobuf
  - Patched for BE support (version 3 with 16-bit fields)

```
// ToVCSECMessage
message Message_10403C1 {
    required Message_10416BD field_1 = 1; // SignedMessage
    required Message_10407AE field_2 = 2; // UnsignedMessage
}
// SignedMessage
message Message_10416BD {
    required bytes field_1 = 1 [(nanopb).max_size = 20];
    required bytes field_2 = 2 [(nanopb).max_size = 282];
// ...
```

VCSEC Vulnerability

x509 Certificate in parts

- During enrollment, VCSEC can ask for the TPMS certificate
- Only for type 5 TPMS
- TPMS of this type are not in production yet 😆



VCSEC Vulnerability

Protobuf certificate in parts

SYNACKTIV

- Certificate x509 sent in parts
- Part encoded with Protocol Buffer `CertificateResponse`

```
message CertificateInParts {
  uint32 startIndex = 1;
  uint32 certificateSize = 2;
  bytes data_ = 3;  // nanopb.max_size:128
}
```

VCSEC Vulnerability

SYNACKTIV

Integer overflow in certificate reassembly

- Integer overflow in the validation of `startIndex`
- Results in Out-Of-Bounds write with a negative `startIndex`

```
char g_cert_buffer[512];

void handle_certificate_response(
    u32_t tpms_id,
    u8_t *data,                 // certificateInParts.data_.bytes
    u32_t data_size,            // certificateInParts.data_.size
    u32_t start_index,          // certificateInParts.startIndex
    u32_t certificate_size)     // certificateInParts.certificateSize
{
    // Integer overflow ex: (start_index:-8 + data_size:64) = 56
    if (data_size <= 512 && (u32_t)(start_index + data_size) <= 512)
    {
        // startIndex can be negative -> OOB write before the global
        memcpy(g_cert_buffer+start_index, data, data_size);
```

VCSEC Vulnerability

SYNACKTIV

Exploitation primitive

- Maximum `data_` buffer size is 128 (enforced by `nanopb`)
- Could overwrite up to 128 bytes of global data before `g_cert_buffer`
- Pointer to a structure containing a **function pointer** just before the buffer

```
struct tpms_auth_s {
    bool (*validate_subject_name)(/*...*/);
    // ...
};

struct tpms_auth_s * g_tpms_auth;
u8 tpms_auth_id;
u8 tpms_auth_state;
char g_cert_buffer[512];
```

- Sending a valid x509 certificate triggers the `validate_subject_name` call

VCSEC Exploitation

Mitigation

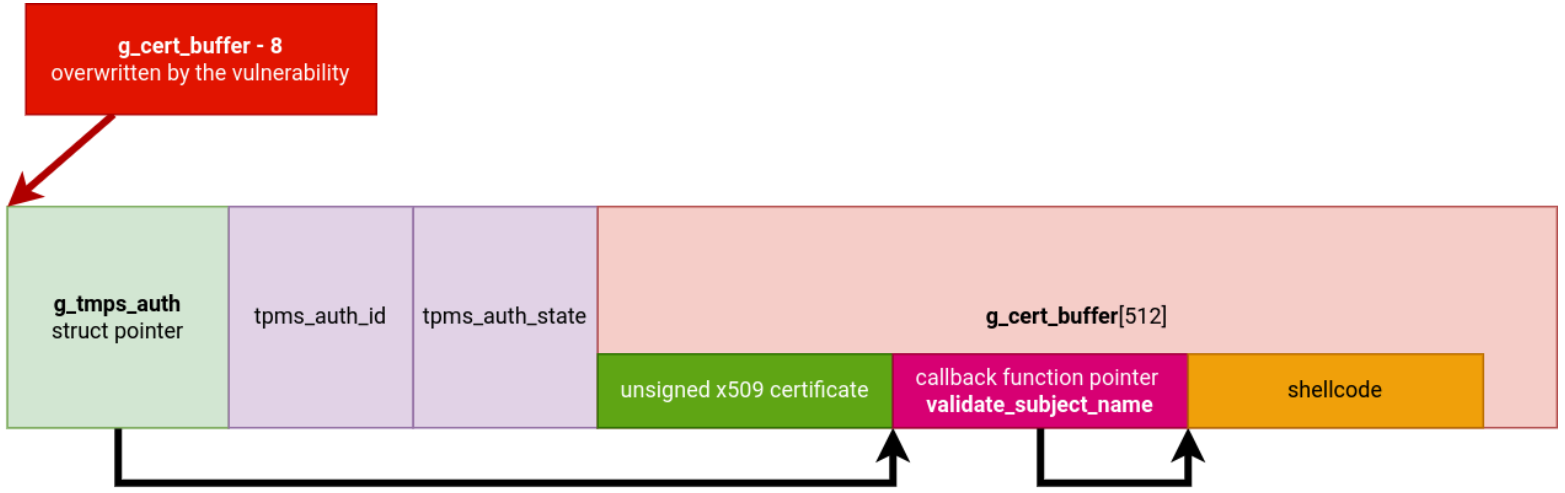
SYNACKTIV

- No CFI
- No ASLR
- MMU/MPU not configured
  - Everything RWX

VCSEC Exploitation

Exploitation

- Overwrite structure pointer to point to the controlled buffer
- Function pointer points to the controlled buffer
- During certificate parsing, it jumps to shellcode (`PPC VLE`)
- Shellcode can be built from C code using a `powerpc-eabivle toolchain`
  - C code can directly call firmware functions (used in post-exploit)



TPMS auto learn

VCSEC Exploitation

1-click to 0-click

SYNACKTIV

⚠ 1-click to 0-click

- The vulnerability requires VCSEC to adopt a new TPMS sensor
- UDS was used to configure VCSEC to add/remove TPMS sensors
  - This is not valid for pwn2own
  - Need another way to adopt TPMS
- Look at the auto learn mechanism

TPMS

Auto learn

Why VCSEC needs a TPMS auto learn feature

- User can have two set of wheels
- Users are encouraged to switch front and back wheels to level tire wear

Auto learn is started if

- Car is moving for more than 90s
- Speed is at least 25 km/h
- VCSEC will compute TPMS position based on its BLE endpoints measurements

Adoption of new TPMS

If a TPMS is disconnected during the auto learn phase VCSEC try to adopt new ones based on BLE advertisements

TPMS

Force the adoption of new TPMS

SYNACKTIV

TPMS BLE connection mechanism

1. TPMS sensor wakes up by the movement of the wheels.
2. TPMS sends BLE advertisements.
3. VCSEC receives BLE advertisement.
4. VCSEC connects to TPMS if the MAC address matches its list of enrolled sensors.
5. BLE connection is established, TPMS stop advertising.

Act as a TPMS sensors

- There is no security except the MAC address list.
- If an attacker sends advertisement with the correct MAC address VCSEC will connect on the fake sensor and accept messages (like fake tire pressure or temperature)
- **In that case VCSEC will not do the TPMS adoption phase** (where our vulnerability lives)

TPMS

SYNACKTIV

Force the adoption of new TPMS

How to DoS a TPMS sensor

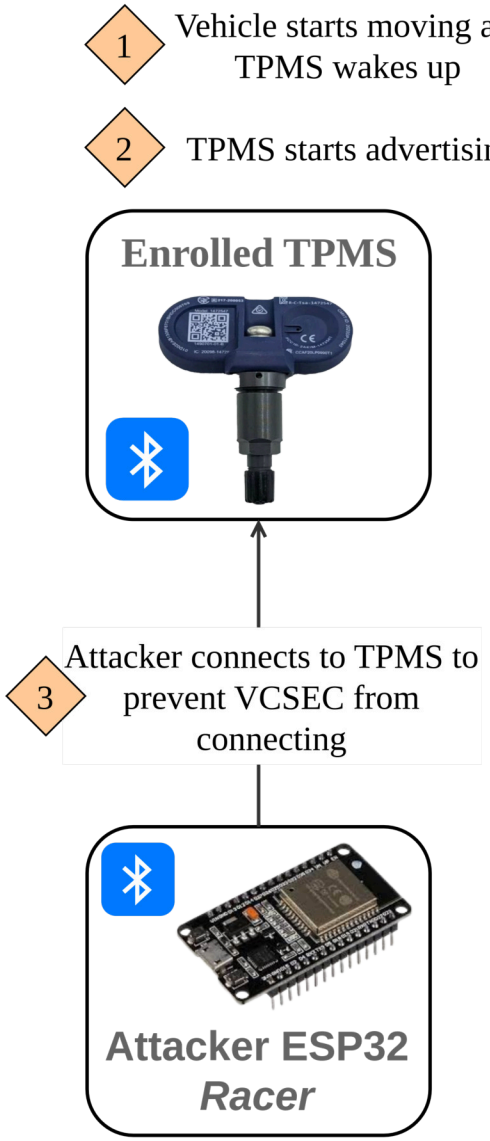
- Just connect on it before VCSEC does when the sensor wake up
- Probably many other ways: JAM signal etc...
- **During auto learn phase, having a disconnected sensor allows to enroll arbitrary new sensors**

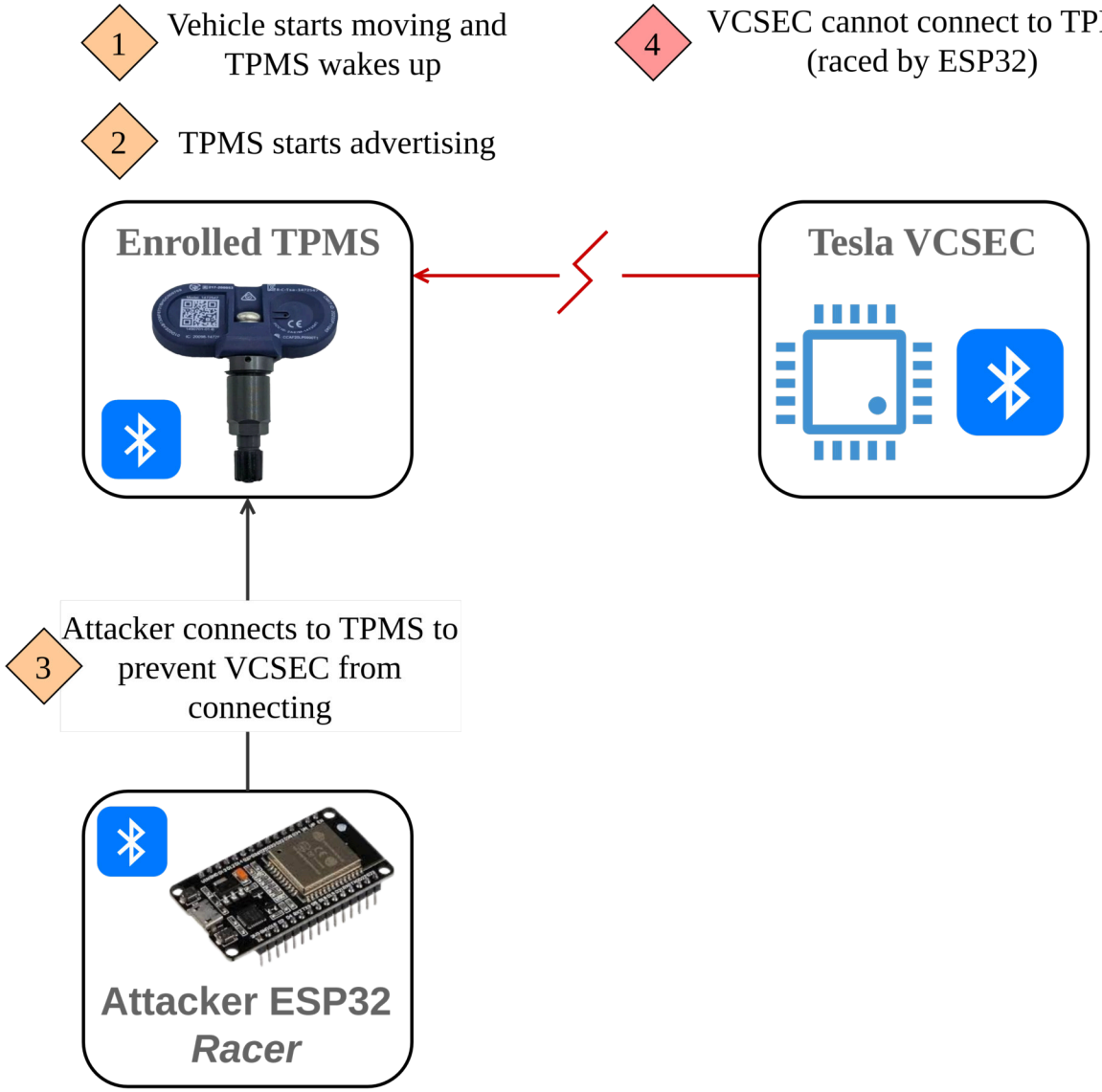
Two ESP32 to the rescue

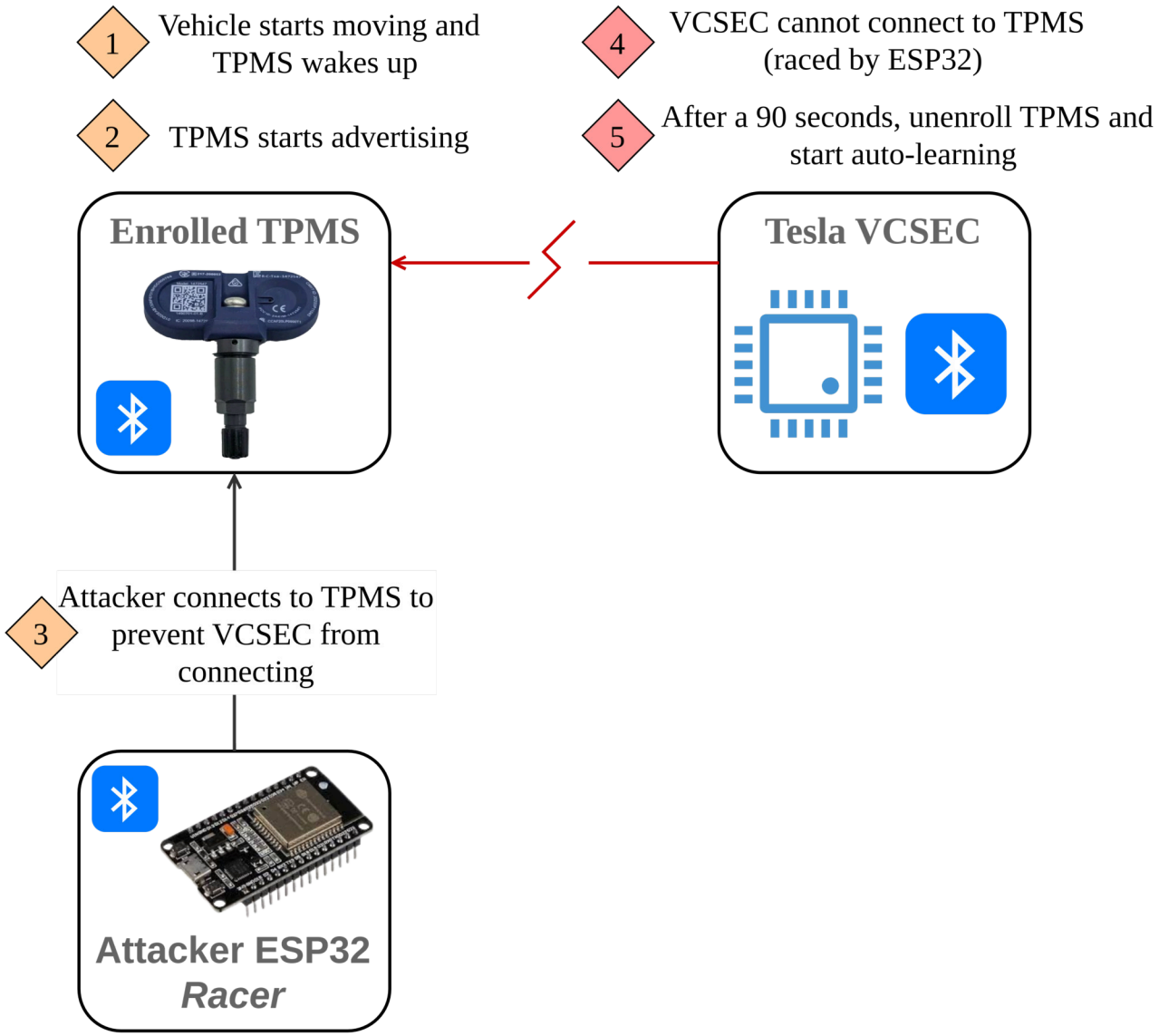
- First try with BlueZ (one of the major bluetooth stack on Linux) but was too slow, VCSEC connects before us
- Automatic connection on advertised TPMS was implemented on ESP32: good success rate in racing VCSEC
- TPMS simulator implemented on another ESP32, VCSEC enrolls it during the auto learn phase
- **TPMS adoption messages are sent to the TPMS simulator, vulnerability can be exploited**

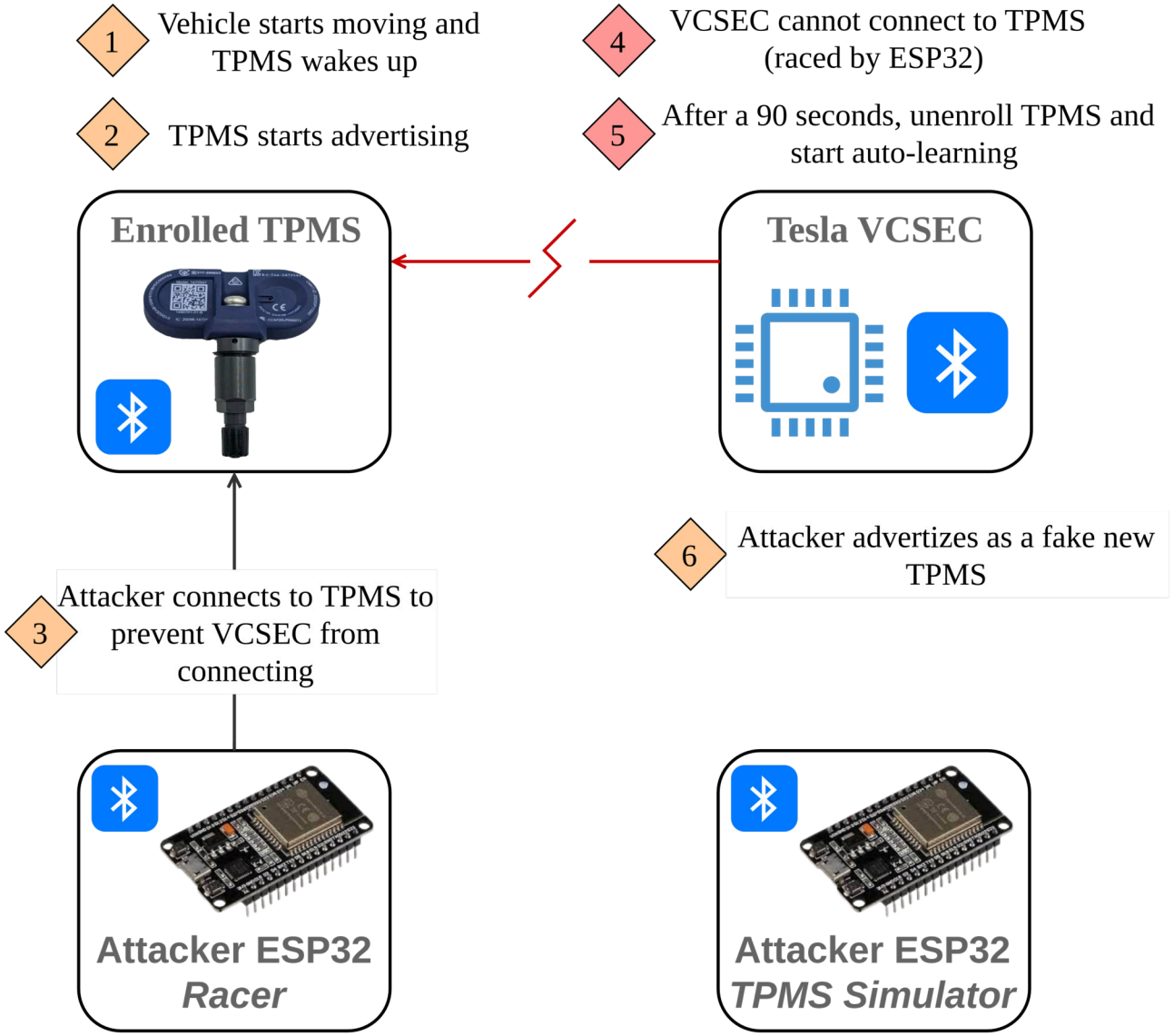
1 Vehicle starts moving and TPMS wakes up

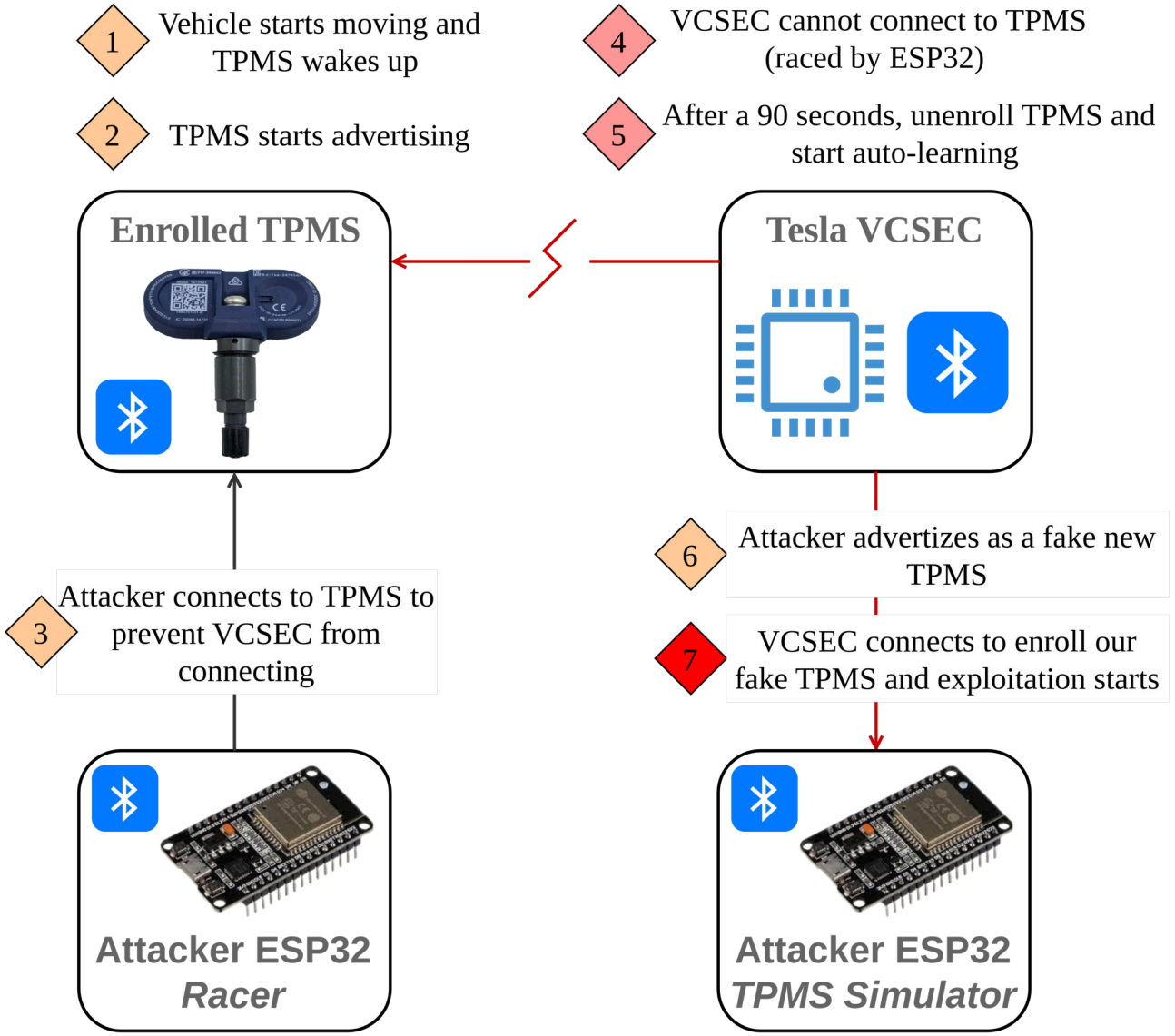
2 TPMS starts advertising


1
Vehicle starts moving and TPMS wakes up
2
TPMS starts advertising
Enrolled TPMS
3
Attacker connects to TPMS to prevent VCSEC from connecting
Attacker ESP32
Racer

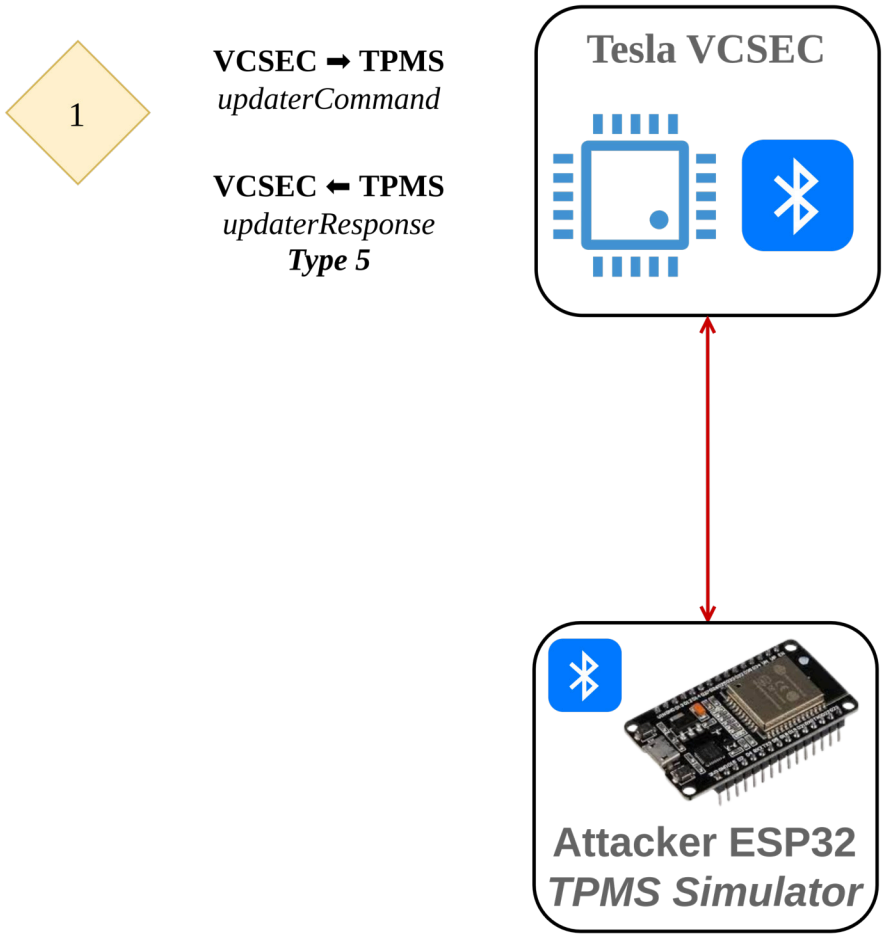

1
Vehicle starts moving and TPMS wakes up
2
TPMS starts advertising
4
VCSEC cannot connect to TPMS (raced by ESP32)
Enrolled TPMS
Tesla VCSEC
3
Attacker connects to TPMS to prevent VCSEC from connecting
Attacker ESP32
Racer


1
Vehicle starts moving and TPMS wakes up
2
TPMS starts advertising
3
Attacker connects to TPMS to prevent VCSEC from connecting
4
VCSEC cannot connect to TPMS (raced by ESP32)
5
After a 90 seconds, unenroll TPMS and start auto-learning
Enrolled TPMS
Tesla VCSEC
Attacker ESP32
Racer


1
Vehicle starts moving and TPMS wakes up
2
TPMS starts advertising
4
VCSEC cannot connect to TPMS (raced by ESP32)
5
After a 90 seconds, unenroll TPMS and start auto-learning
Enrolled TPMS
Tesla VCSEC
6
Attacker advertizes as a fake new TPMS
3
Attacker connects to TPMS to prevent VCSEC from connecting
Attacker ESP32
Racer
Attacker ESP32
TPMS Simulator


1
Vehicle starts moving and TPMS wakes up
2
TPMS starts advertising
4
VCSEC cannot connect to TPMS (raced by ESP32)
5
After a 90 seconds, unenroll TPMS and start auto-learning
Enrolled TPMS
Tesla VCSEC
3
Attacker connects to TPMS to prevent VCSEC from connecting
6
Attacker advertizes as a fake new TPMS
7
VCSEC connects to enroll our fake TPMS and exploitation starts
Attacker ESP32
Racer
Attacker ESP32
TPMS Simulator

TPMS Enrollment Mode



TPMS Enrollment Mode

1

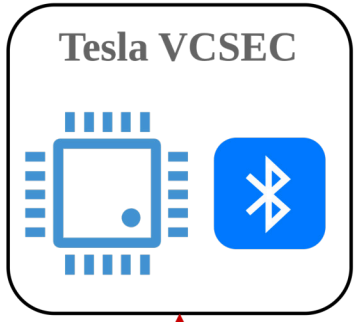
VCSEC ➡ TPMS
updaterCommand

VCSEC ⬅ TPMS
updaterResponse
Type 5

2

VCSEC ➡ TPMS
certificateRead

VCSEC ⬅ TPMS
certificateResponse
Exploit vulnerability



TPMS Enrollment Mode

1

VCSEC ➡ TPMS
updaterCommand

VCSEC ⬅ TPMS
updaterResponse
Type 5

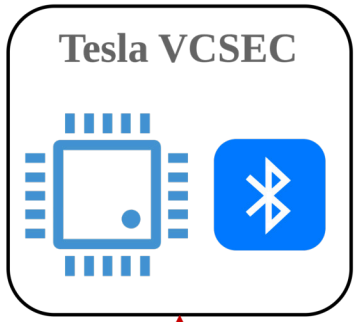
2

VCSEC ➡ TPMS
certificateRead

VCSEC ⬅ TPMS
certificateResponse
Exploit vulnerability

3

Code execution on VCSEC



Conclusion

Conclusion

SYNACKTIV

Final payload

- Shellcode: send `SYNACKTI` `V<3TESLA` on the vehicule CAN on CAN ID `0x444` :
- Locate and use the fonction in the firmware to send CAN messages
- Used as proof of exploitation for Pwn2Own: Abritrary CAN message on vehicule CAN from a remote connection

```
#define fnsend_can_raw ((void (*)(char *msg, int id))0x10B7D60)

int main_payload()
{
   while(1) {
       fnsend_can_raw("\x44\x44\x08SYNACKTI", 60);
       fnsend_can_raw("\x44\x44\x08V<3TESLA", 60);
   }
   return 0;
}
```

VCSEC Exploitation

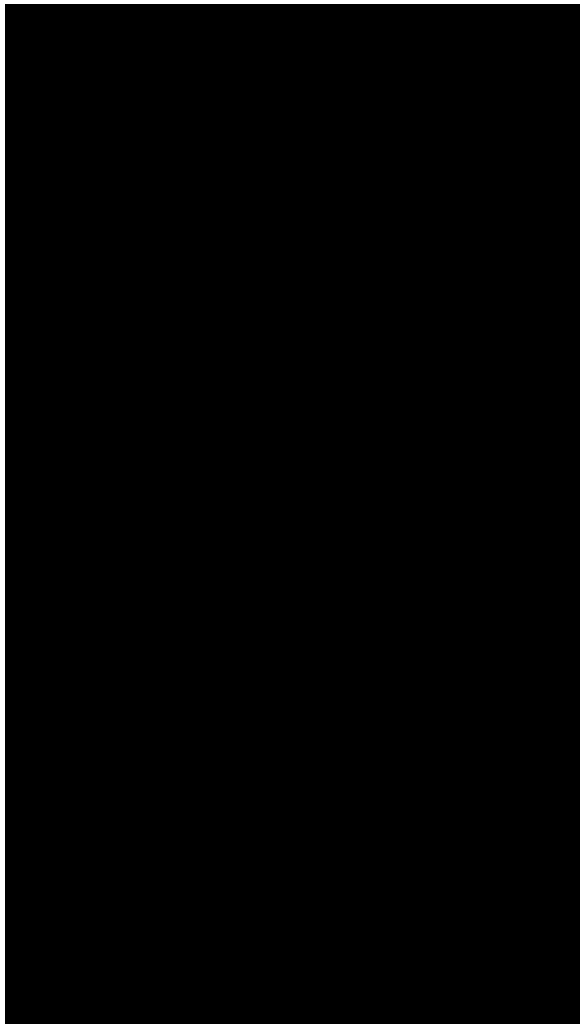
Result

SYNACKTIV

- First try at Pwn2Own Vancouver 2024 (March)
- Win: 200 000$ plus a Tesla Model 3 (2024)

- A lot more easier than our three other Tesla Pwn2own entries (2022, 2023, january 2024)
  - Infotainment attacks are difficult because of good defense in depth

    User isolation, sandboxing, ASLR, PIE, ...
- Try your luck, standalone ECUs are a good candidate to start



Conclusion

SYNACKTIV

Impacts

- VCSEC is a critical ECU for the car security
  - It manages access to the car and grants the user the right to start the car
  - It has access to the vehicule CAN and can send messages to do some action on the car
- Having code execution in this ECU gives an attacker the ability to perform these actions
- Attack can be implemented on very small devices

Fixes

- Tesla quickly released a new version that fixes the bug
  - Vulnerability is fixed and other variables are also checked

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://twitter.com/synacktiv

https://synacktiv.com