CANSPY
a Platform for Auditing CAN Devices

Arnaud Lebrun
Jonathan-Christofer Demay
Auditing conventional IT systems

• **Penetration testing**
  • A form of security audit
  • Assess the risks of intrusion
  • Actual tests instead of a review process
  • The point of view of a real attacker (the “black-box” approach)
  • Relevant evaluation of impact and exploitability

• **Limitations**
  • Less time
  • Less resources
  • More ethics

• **Counter-measure: the “grey-box” approach**
The CISO’s dilemma

- **The hand they are dealt with**
  - Huge scope of responsibility
  - Continuous changes
  - Major security threats
  - Risk of substantial damages
  - Limited budget

- **Their response**
  - They rely on penetration testing
  - They welcome the “gray-box” approach
  - They rely on risk analysis first and foremost
  - They divide perimeters accordingly

Image: "And over there we have the labyrinth guards. One always lies, one always tells the truth, and one stabs people who ask tricky questions."
What about car manufacturer?

- They are starting to include cyber-security along with conventional safety

  - A lot of new functionalities
  
  - Using more complex software
  
  - Also, security researchers...
What about car manufacturer?

- They are starting to include cyber-security along with conventional safety.
What about car manufacturer?

- They are starting to include cyber-security along with conventional safety
What about car manufacturer?

- They are starting to include cyber-security along with conventional safety.
What about security audit for cars?

• **The same approach can be applied**
  
  - While True
    - Conduct risk analysis
    - Prioritize ECUs
    - Conduct penetration tests accordingly
    - Carry out corrective actions
  
  - End While

• **Some ECUs can be common to several vehicles**

• **Corrective actions may be difficult to carry out**
It always begins with...

- Consumer-grade connectivity
  - Wi-Fi, Bluetooth and USB → Nothing new here!
  - However CAN sniffing is already useful for analysis
It always begins with…

- Mobile broadband connectivity
  - Setting up an IMSI catcher and then...
  - Deal with conventional protocols (TCP, HTTP, ...) \(\rightarrow\) Again, nothing new here!
It always begins with...

**CAN attacks**
- Bypass CAN bus segmentation (architecture-dependant)
- Reverse-engineer higher-layer/custom protocols
- Break the Security Access challenge (ISO 14229)
CAN architectures

• One serial bus (to rule them all)
  • ID-based priority mechanism
  • Congestion issues
  • Acknowledgment by anyone

ECU  ECU  ECU  ECU  ECU
CAN High
CAN Low
CAN architectures

- Multiple separate buses
  - Some ECUs have to be connected to multiple buses
  - They can be used to bypass the segmentation
CAN architectures

- **Multiple interconnected buses**
  - A gateway is routing frames between CAN buses
  - It may take into account the state of the vehicle
  - Both safety and cyber-security can be considered
Crafting CAN attacks

• Several attack vectors
  • Misuse of intrinsic capabilities (e.g., remote diagnostic tool)
  • Exploit a higher-level parsing vulnerability
  • Break the Security Access challenge
  • Etc.

• This will imply a substantial amount of work
  • Unsolder EEPROM or identify on-chip debug (JTAG/BDM) and conventional debug (UART/WDBRPC) interfaces
  • Extract the firmware
  • Reverse-engineer the aforementioned items
  • Craft actual attacks
The Man In The Middle

• **Taking advantage of the client-server model**
  - Insert yourself in-between them
  - Do not alter traffic until you see something interesting
  - Then start to drop/alter/replay/...
  - Finalize with targeted reverse-engineering

• **In theory, this is transposable to the CAN bus**
  - We are auditing one device
    → We could proxy the traffic from and to that device
  - We are working with the car manufacturer
    → We can ask for a restricted devices (e.g., a remote diagnostic tool)
    → This is limited by third-parties intellectual properties
However, in practice...

- **CAN is a serial bus**
  - Physically cut the bus and insert yourself in-between
  - Forward traffic between the split parts
  - Etc.

- **2 possible options (other than deep diving into the car)**
  - Emulate the car from the point of view of the audited device
  - Use an integration bench provided by the car manufacturer
However, in practice...

- **CAN is a serial bus**
  - Physically cut the bus and insert yourself in-between
  - Forward traffic between the split parts
  - Etc.

- **2 possible options (other than deep diving into the car)**
  - Emulate the car from the point of view of the audited device
  - Use an integration bench provided by the car manufacturer
What about existing (open-source) tools?

- **CAN was designed to meet timing constraints**
  - Bridging two devices could add high latencies
  - Slow Arduino-like microcontrollers will drop frames
What about existing (open-source) tools?

- **CAN was designed to meet timing constraints**
  - Bridging two devices could add high latencies
  - Slow Arduino-like microcontrollers will drop frames

- **UART (over USB) is a bottleneck**
  - The default is usually 115 200 bauds (and even at max speed it is limiting)
  - CAN buses can go as far as 1Mbit/s (OBD-II is 250 or 500 Kbit/s)
  - We need two of them (cf. timing constraints)
What about existing (open-source) tools?

- **CAN was designed to meet timing constraints**
  - Bridging two devices could add high latencies
  - Slow Arduino-like microcontrollers will drop frames

- **UART (over USB) is a bottleneck**
  - The default is usually 115 200 bauds (and even at max speed it is limiting)
  - CAN buses can go as far as 1Mbit/s (OBD-II is 250 or 500 Kbit/s)
  - We need two of them (cf. timing constraints)

- **Lack of a mature framework**
  - We get frustrated when we cannot use Scapy 😞
  - Federate higher-layers reverse-engineering efforts
CANSPY objectives

- **Two dedicated CAN interfaces**
  - Using independent CAN cores
  - With the ability to manipulate acknowledgments
- **Frame forwarding w/ or w/o filtering**
  - Low latencies (even with filtering)
  - At the full data rate of the CAN standard
- **Sniffing and injection capabilities**
  - CAN interfaces $\leftarrow\rightarrow$ Ethernet (with Wireshark dissector compatibility)
  - CAN interfaces $\leftarrow\rightarrow$ UART (mostly for setting/debugging purposes)
- **PCAP and settings read/write from SD card (autonomous mode)**
- **Configurable settings via Ethernet (fully scriptable)**
CANSPY hardware

• STM32F4DISCOVERY board
  • 168 MHz 32bit ARM Cortex M4
  • COTS ($20)
CANSPY hardware

- **STM32F4DISCOVERY board**
  - 168 MHz 32bit ARM Cortex M4
  - COTS ($20)

- **STM32F4DIS-BB extension board**
  - 1 RS232 interface
  - 1 Ethernet port
  - 1 SD card drive
  - COTS ($40)
CANSPY hardware

- STM32F4DISCOVERY board
  - 168 MHz 32bit ARM Cortex M4
  - COTS ($20)
- STM32F4DIS-BB extension board
  - 1 RS232 interface
  - 1 Ethernet port
  - 1 SD card drive
  - COTS ($40)
- DUAL-CAN extension board
  - Configurable resistors, power supplies and circuit grounds
  - 2 CAN interfaces and easy to build
  - Custom-made ($30 worth of PCB and components)
**CANSPY firmware**

- **Non-preemptive priority scheduler**
  - High-priority cyclic executive for synchronous services
  - Low-priority cyclic executive for asynchronous services

**Components**

- Service
- Transmit buffer
- Receive buffer
- Mutex
- Device handler

**Hardware Abstraction Layer**

- Device driver

**Source**

https://bitbucket.org/jcdemay/canspy
**CANSPY firmware**

- **Event-driven scheduler**
  - Asynchronous I/O operations
  - Low latency processing
- **1 functionality == 1 service**
  - Start only what you need
  - Read from all devices, write to only one
  - Mutual exclusion is possible
- **Autonomous mode**
  - In-built filtering/altering engine
  - SD card for read or write operations
  - Power supply from the car battery
- **Real-time approach**
- **Open source licensed**
- **Built-in services**
  - CAN: Forward/Filter/Inject
  - Ethernet: Wiretap/Bridge
  - SDCard: Capture/Replay/Logdump
  - UART: Monitor/Logview/Shell
- **CAN devices**
  - Two independent handlers
  - Support all standard speeds
  - Throttling mechanisms
Handling congestion issues

• **MITM setups can tamper with congestion**
  - Filtering or dropping will modify the available bandwidth
  - ECUs behavior may thus be impacted

• **Two possible throttling mechanisms**
  - Dummy frame injection
  - Delaying acknowledgments
CAN over Ethernet

• The SocketCAN format
• Ethertype 0x88b5
• Different MAC addresses
• Acknowledgments
CAN over Ethernet

- The SocketCAN format
- Ethertype 0x88b5
- Different MAC addresses
- Acknowledgments

```python
class SocketCAN(Packet):
    name = "SocketCAN"
    fields_desc = [
        BitEnumField("EFF", 0, 1, {0:"Disabled", 1:"Enabled"}),
        BitEnumField("RTR", 0, 1, {0:"Disabled", 1:"Enabled"}),
        BitEnumField("ERR", 0, 1, {0:"Disabled", 1:"Enabled"}),
        XBitField("id", 1, 29),
        FieldLenField("dlc", None, length_of="data", fmt="B"),
        ByteField("__pad", 0),
        ByteField("__res0", 0),
        ByteField("__res1", 0),
        StrLenField("data", "", length_from = lambda pkt: pkt.dlc),
    ]
    def extract_padding(self, p):
        return "",p

bind_layers(Ether, SocketCAN, type=0x88b5)
```
CAN over Ethernet

- The SocketCAN format
- Ethertype 0x88b5
- Different MAC addresses
- Acknowledgments

```python
class SocketCAN(Packet):
    name = "SocketCAN"
    fields_desc = [
        BitEnumField("EFF", 0, 1, {0: "Disabled", 1: "Enabled"}),
        BitEnumField("RTR", 0, 1, {0: "Disabled", 1: "Enabled"}),
        BitEnumField("ERR", 0, 1, {0: "Disabled", 1: "Enabled"}),
        XBitField("id", 1, 29),
        FieldLenField("dlc", None, length_of="data", fmt="B"),
        ByteField("__pad", 0),
        ByteField("__res0", 0),
        ByteField("__res1", 0),
        StrLenField("data", "", length_from=lambda pkt: pkt.dlc),
    ]

def extract_padding(self, p):
    return "", p

bind_layers(Ether, SocketCAN, type=0x88b5)
```

#wireshark -X lua_script:ethcan.lua

local sll_tab = DissectorTable.get("sll.ltype")
local can_hdl = sll_tab:get_dissector(0x000C)
local eth_tab = DissectorTable.get("ethertype")
eth_tab:add(0x88b5, can_hdl)
The OBD-II use case

• No need to physically cut anything
  • Buy a Goodthopter-compatible OBDII-to-DB9 cable
  • Build its female counterpart ($10 worth of components)
  • Setup the DUAL-CAN extension properly
  • Have fun 😊

• Several interesting cases
  • Professional/consumer car diagnostic tools
  • Usage-based policies from insurance companies
  • Air-pollution control from law enforcement

• They expose sensitive networks/hosts
Demonstration bench

Man-In-The-Middle

Emulated ECUs

OBD2 Device

OBD2 Diagnostics

MITM

Emulated ECUs

CAN1

CAN2

ETH
Demonstration bench

Start of emulation

Start of filtering (frame modification)

Vehicle speed (Speed)

255 km/h

0.0

30 27 24 21 19 16 13 10 8 5 2

OBD2 Diagnostics

MITM

Emulated ECUs

CAN2

ETH

CAN1

Start of emulation

Start of filtering (frame modification)
Demonstration bench

• **What about buffer overflows?**
  - ISO-TP layer provided for Scapy
  - Identify fragmented responses
  - E.g., VIN request (17 ASCII characters)
  - Increase response length
  - Debug and exploit

• **We need more Scapy layers!**
  - For documented standards (e.g., SAE J1939)
  - For proprietary standards (i.e., reversing...)

---

**Arnaud Lebrun**
Jonathan-Christofer Demay

**CANSPY**
a Platform for Auditing CAN Devices
Thank you for your attention

https://bitbucket.org/jcdemay/canspy