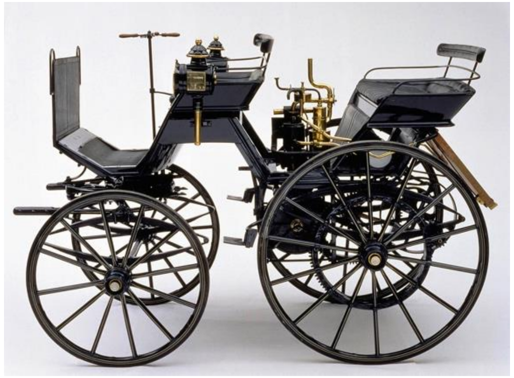
Automotive

CS463/ECE424

University of Illinois

Automobiles Security



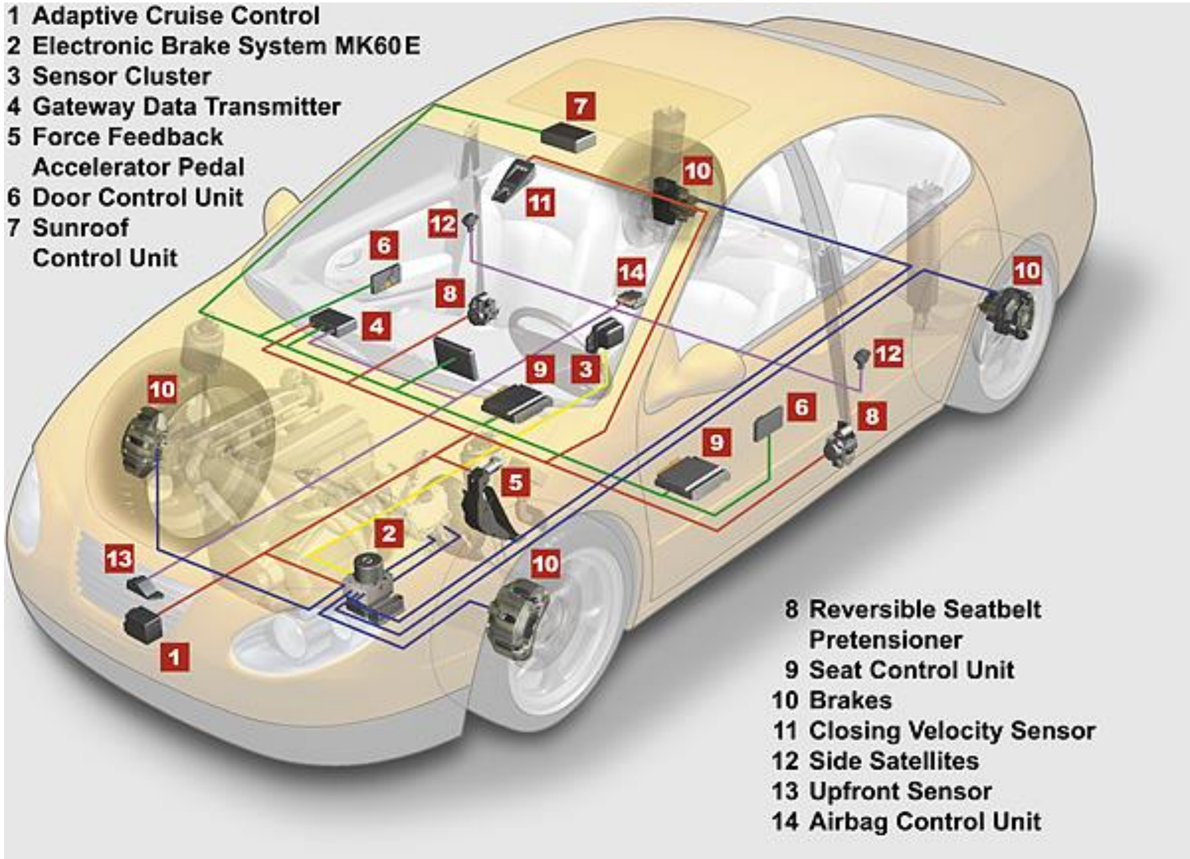

1 Adaptive Cruise Control
2 Electronic Brake System MK60E
3 Sensor Cluster
4 Gateway Data Transmitter
5 Force Feedback Accelerator Pedal
6 Door Control Unit
7 Sunroof Control Unit
8 Reversible Seatbelt Pretensioner
9 Seat Control Unit
10 Brakes
11 Closing Velocity Sensor
12 Side Satellites
13 Upfront Sensor
14 Airbag Control Unit

Image from https://www.aa1car.com/library/2004/bf110412.htm

Digital Controls in Automobiles

- Digital control, in the form of self-contained embedded systems called Electronic Control Units (ECUs), entered US production vehicles in the late 1970s.
- This was driven by legislation like the Clean Air Act of 1970 and pressure from increasing gasoline prices.
- By dynamically measuring the oxygen present in exhaust fumes, the ECU could adjust the fuel/oxygen mixture before combustion, thereby improving efficiency and reducing pollution.

Software in Cars

- F-35 Joint Strike Fighter – 5.7M lines of code
- Windows XP – 40M lines of code
- **A premium car in 2009 – 100M lines of code,**
  - 70-100 ECUs
- Mercedes Benz S-class
  - Radio and navigation system – 20M lines of code
  - Has as many ECUs as Airbus A 380.
- Cost of software and electronics can reach 35%-40% of the cost of the car.

Current Software Functionality

- Car controls
- Performance monitoring
- Automatic crash response
- Emergency call system
- Theft recovery system
- Remote execution
- Navigation system
- App integration – Twitter, Facebook, Google Search
- Driver monitoring capabilities

Android Auto and Apple CarPlay

- Allows smartphone to connect to the car with a rich interface
- Car can exchange data and access Internet using the smartphone
- Majority of car manufacturers support this
- Android Auto
- iOS CarPlay

References

- [Koscher 10]

Experimental Security Analysis of a Modern Automobile

Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno (UW), Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage (UCSD).

IEEE S&P 2010.

- [Checkoway 11]

Comprehensive Experimental Analyses of Automotive Attack Surfaces

Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage (UCSD), Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno (UW). **USENIX Security 2011.**

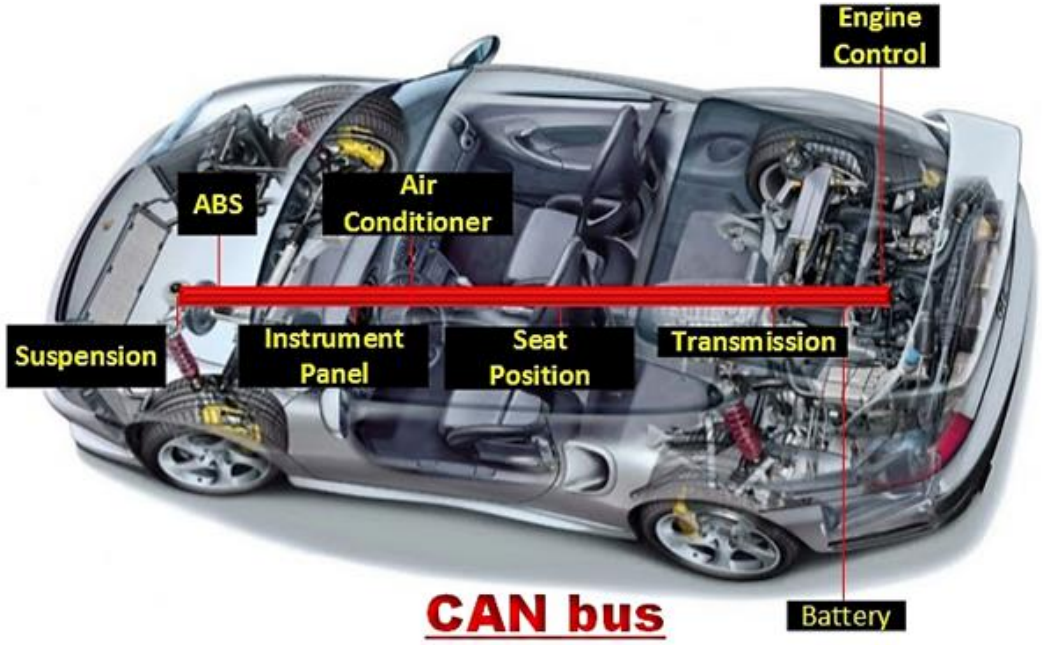
First Case Study [Koscher 2010]

- **[Why?]** Cars have advanced internal vehicular networks of many computers creating risks of cyber exploits on these networks
- **[What?]** Empirically tested a car to find vulnerabilities available to attackers who obtain physical access to the cabin of the car.
- **[Wow!]** Demonstrated serious vulnerabilities such as disabling the brakes and stopping the engine while the car is being driven.

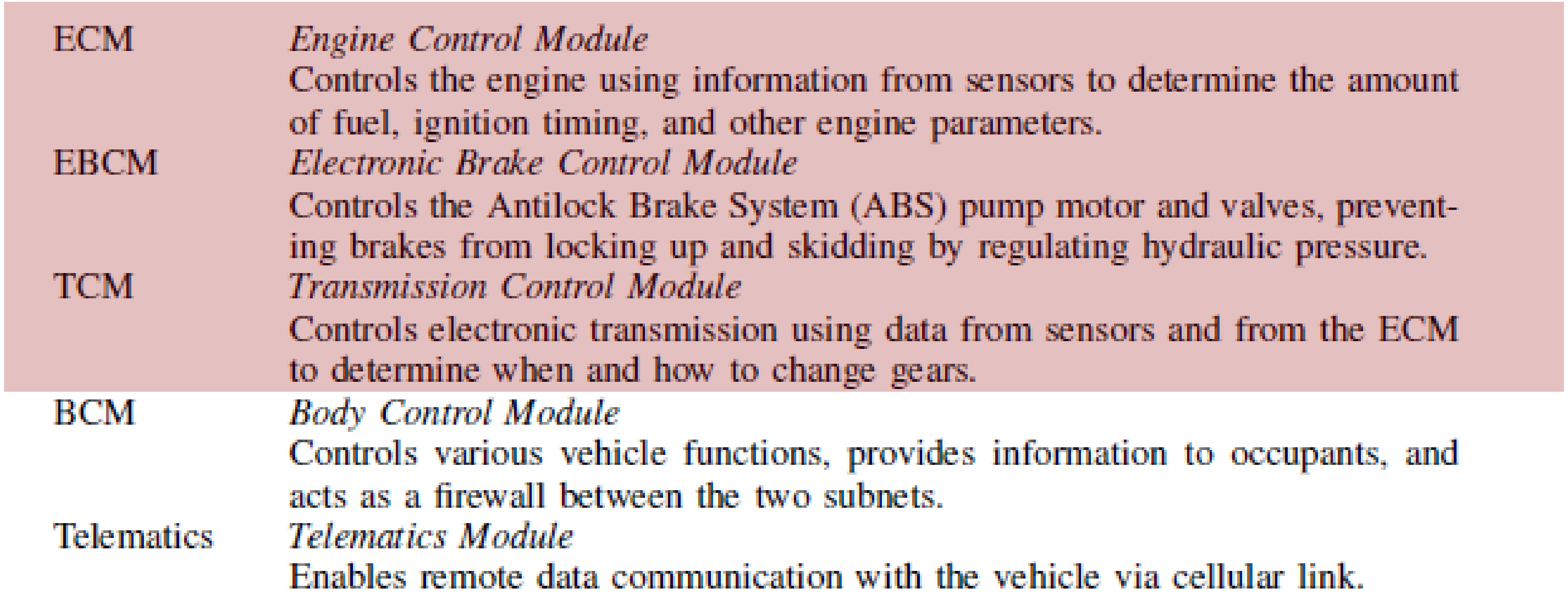
Connectivity enabled by OBD-II Port

- On-Board Diagnostics II (OBD-II) port gives access to vehicle buses via their communication protocol
- CAN - Controller Area Network - ISO 11898-[1-6]
  - Originated by Bosch in 1983.
  - **Message based with division into high speed and low speed buses.**
  - Used by BMW, Ford, GM, Honda, Volkswagen, and others.

CAN Bus and ECUs

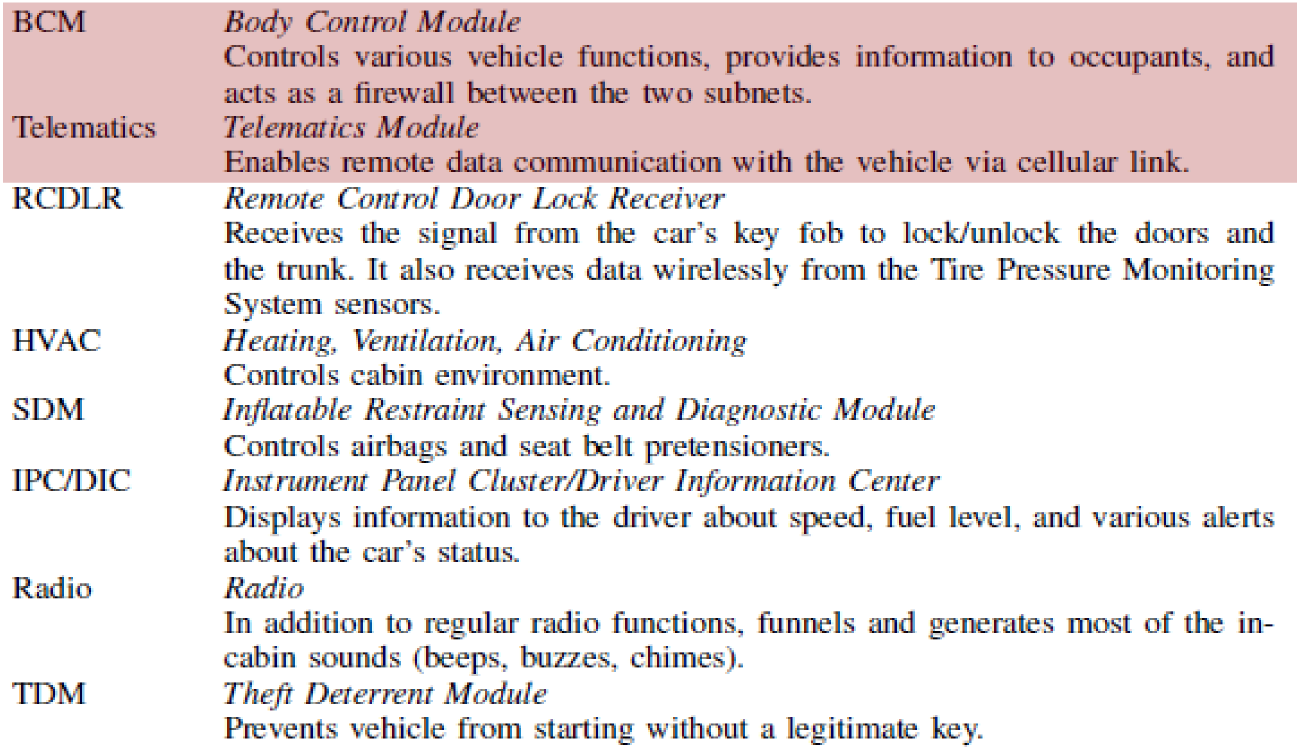


Sample ECUs on High-Speed CAN Bus of Test Car



ECM	*Engine Control Module*
	Controls the engine using information from sensors to determine the amount of fuel, ignition timing, and other engine parameters.
EBCM	*Electronic Brake Control Module*
	Controls the Antilock Brake System (ABS) pump motor and valves, preventing brakes from locking up and skidding by regulating hydraulic pressure.
TCM	*Transmission Control Module*
	Controls electronic transmission using data from sensors and from the ECM to determine when and how to change gears.
BCM	*Body Control Module*
	Controls various vehicle functions, provides information to occupants, and acts as a firewall between the two subnets.
Telematics	*Telematics Module*
	Enables remote data communication with the vehicle via cellular link.

Sample ECUs on Low-Speed CAN Bus of Test Car



BCM	*Body Control Module* Controls various vehicle functions, provides information to occupants, and acts as a firewall between the two subnets.
Telematics	*Telematics Module* Enables remote data communication with the vehicle via cellular link.
RCDLR	*Remote Control Door Lock Receiver* Receives the signal from the car's key fob to lock/unlock the doors and the trunk. It also receives data wirelessly from the Tire Pressure Monitoring System sensors.
HVAC	*Heating, Ventilation, Air Conditioning* Controls cabin environment.
SDM	*Inflatable Restraint Sensing and Diagnostic Module* Controls airbags and seat belt pretensioners.
IPC/DIC	*Instrument Panel Cluster/Driver Information Center* Displays information to the driver about speed, fuel level, and various alerts about the car's status.
Radio	*Radio* In addition to regular radio functions, funnels and generates most of the in-cabin sounds (beeps, buzzes, chimes).
TDM	*Theft Deterrent Module* Prevents vehicle from starting without a legitimate key.

CAN Security Issues

- **[Issue 1]** Packets are **broadcast to all nodes**: the nodes decide whether to accept the information or not
- **[Issue 2]** Packet flooding attack: priority-based arbitration allows node to **assert dominant state indefinitely**
- **[Issue 3]** No source Identifier fields in the packets: **any source can send packets to any other node in the car**

CAN Security Issues (Continued)

- Access control is based on challenge response sequence to protect ECU against certain actions without authorization.

- **[Issue 4]** ECUs are expected to use a fixed challenge (seed) for each of these challenge-response pairs.

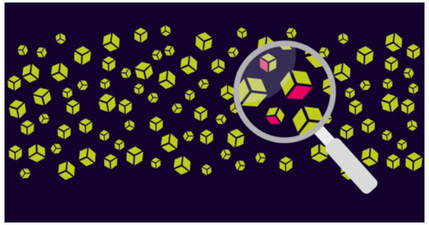
- **[Issue 5]** The responses (keys) are also fixed and stored in the ECUs.
  - Challenges and responses are both 16 bits – can be brute forced
  - Many of the seed-to-key algorithms in use today are known by the car tuning community

Deviations from Standards

There are some standards, but not every manufacturer follows

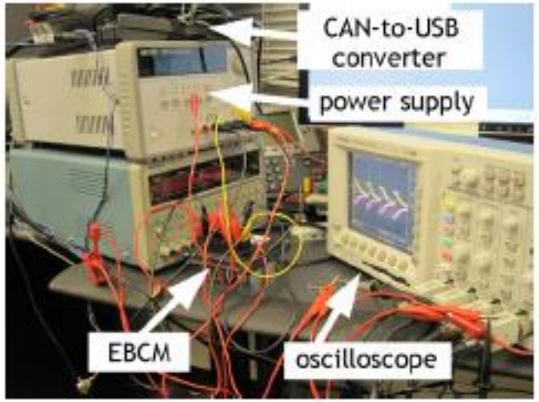
- ✗ ECUs should reject commands like “disable CAN communications” when it is unsafe (viz. when the car is moving).
- ✗ An ECU should reject a request to initiate a programming event if the engine is running.
- ✗ It should be possible to re-flash the unit only after authentication.
- ✗ Reading data from sensitive memory areas (viz. challenge responses) should not be allowed.
- ✗ Gateways between the two networks must only be re-programmable on the high-speed network to prevent a low-speed device from compromising a gateway to attack the high-speed network.

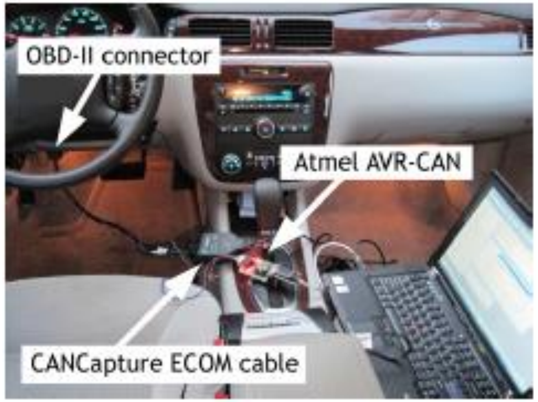
Attack Methodology



- **[Attack 1] Packet sniffing** and targeted probing
- **[Attack 2] Fuzzing:** the range of valid CAN packets is small, significant damage can be done by simple fuzzing of packets
- **[Attack 3] Reverse engineering:** notably the telematics unit – dumped the code via the CAN ReadMemory service and used a third-party debugger (IDA Pro)

Testing Environments





Attacks

- All the attacks used the above-mentioned methodology to determine the message for each function
- The authors modified the code to replay the messages to have the desired effect
- The details on how to modify the code etc. **are omitted from the paper**
  - Why?

Power of Exploits

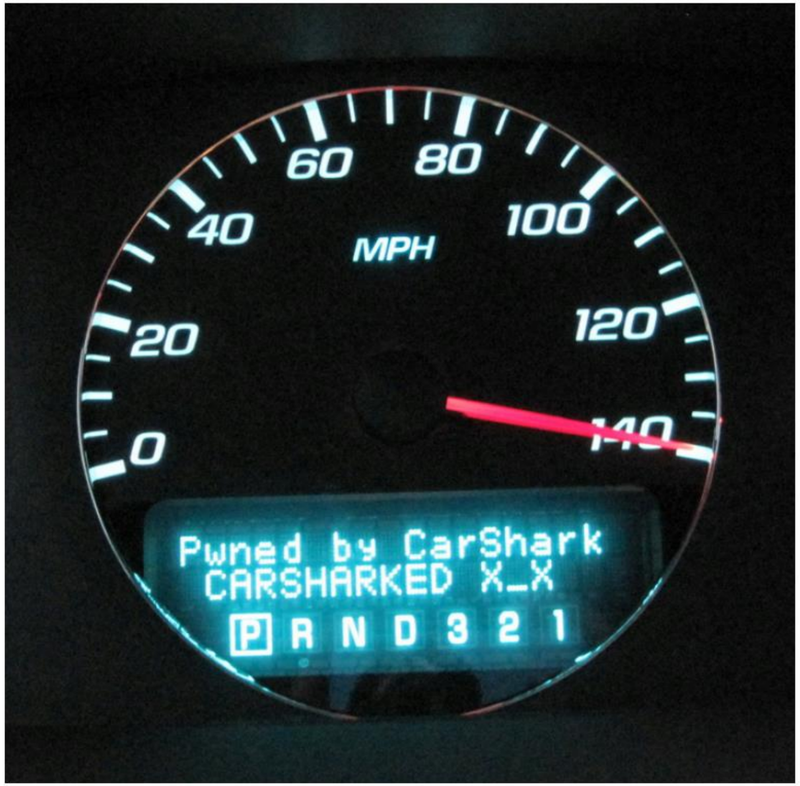
- Control of radio, disable user control, increase volume, etc.
- Display arbitrary messages on the instrument panel cluster
- Honk the horn, lock doors, shoot windshield fluids etc.
- Boost engine RPM, disturb engine timing, disable all cylinders
- Deploy airbags

- Lock individual brakes, release brakes, prevent enabling of brakes
- Turn on/off fans and AC, Control lights
- Make an arbitrary offset to reported speed
- Self destruct, self-wiping code

Hacked Car

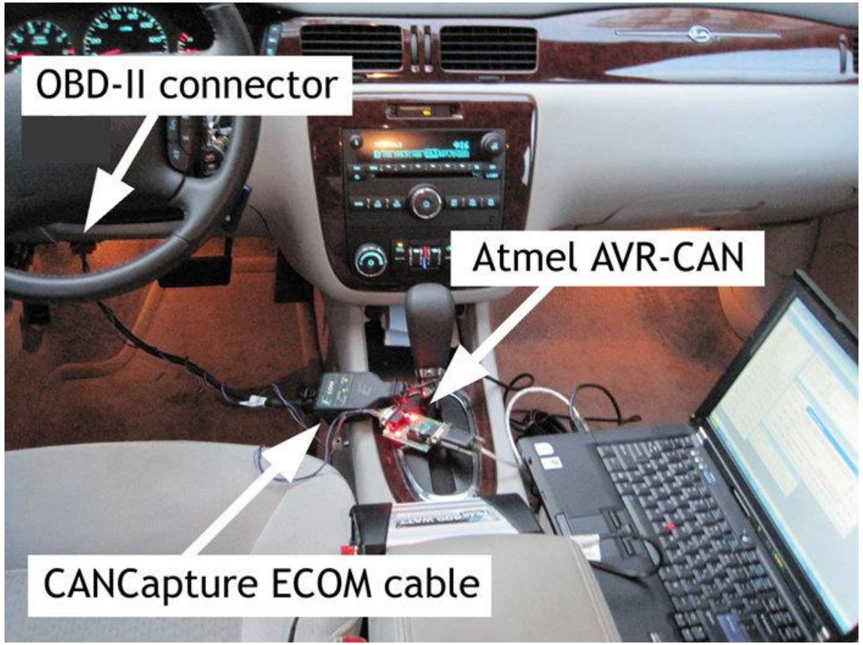
Displaying an arbitrary message and a false speedometer reading on the Driver Information Center.

<u>Note that the car is in Park.</u>



Limitation of This Study

- Need physical access to the car (OBD-II port)



Second Case Study

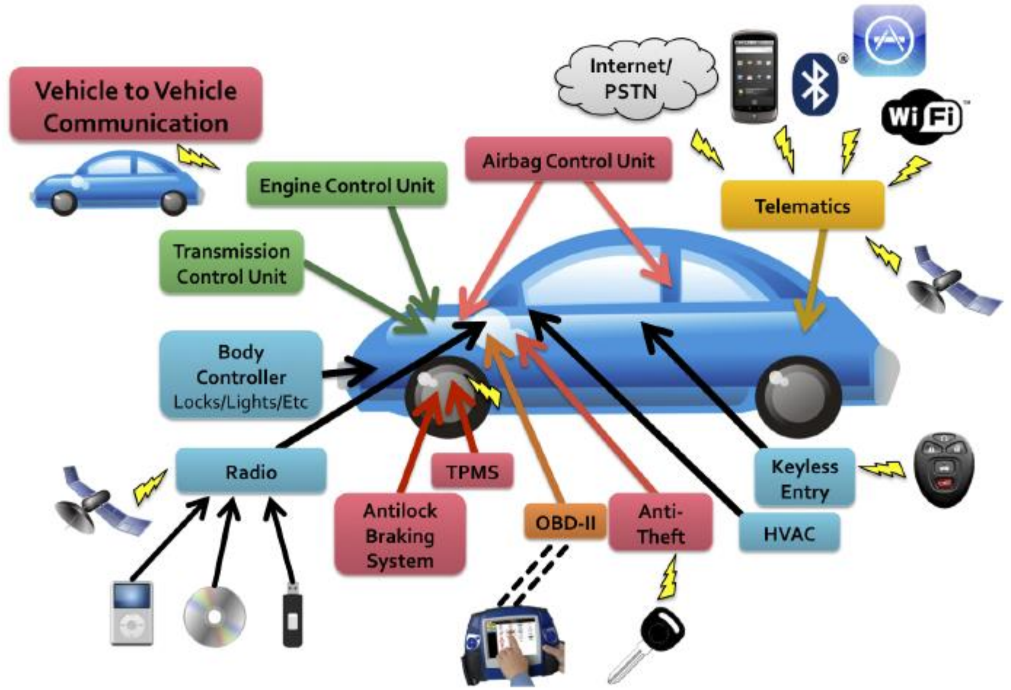
- Internal networks in cars are insecure but there has been insufficient study of attack surfaces for those networks.

- The authors provide a comprehensive experimental view of current status with two experimental cars.

- **[Wow!!]** Exploitation is shown to be possible through several pathways including **mechanics tools, CD players, Bluetooth, and cellular radio.**

Threat Model

Previous paper required physical connection to the OBD port (weak threat model). This paper looks into three more general avenues for access.



- **Indirect physical**
  - Audio CD, USB, iPod
- **Short Range Wireless**
  - Bluetooth devices (phones), WiFi Access Points
  - Remote Keyless entry, RFID car keys
  - Wireless tire pressure monitoring sensors
- **Long Range Wireless**
  - Telematics Units Broadcast channels like radio, cellular

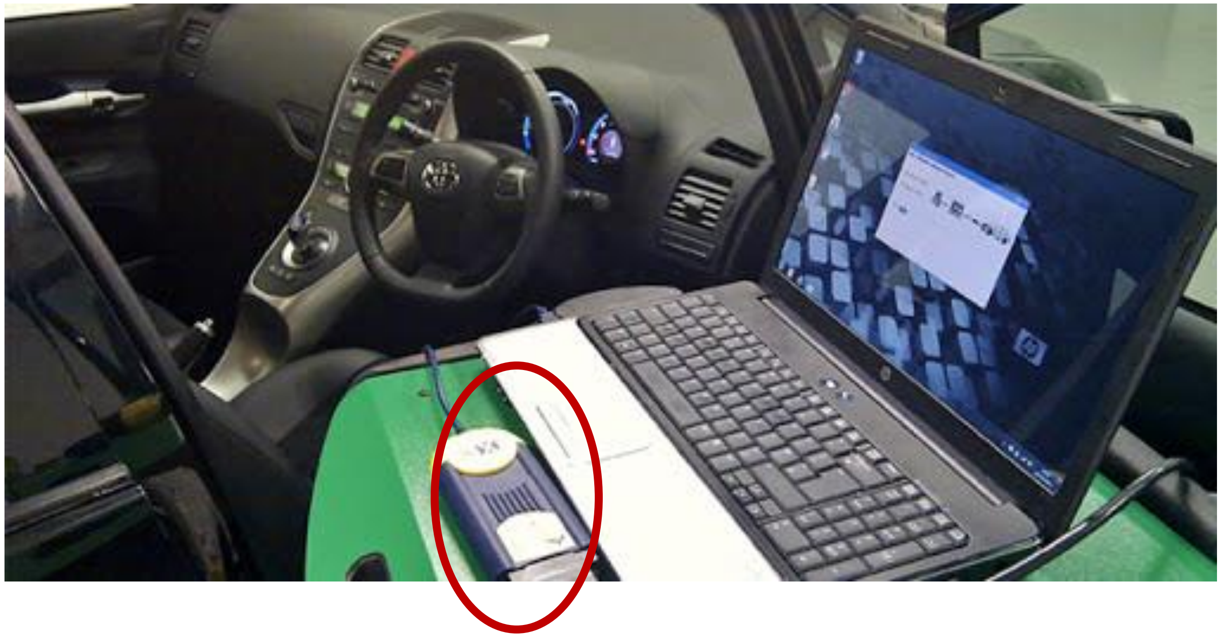
Attack 0: Media Player Attack

- **[What?]** One of the file read functions **makes strong assumptions about input length.** But, there is a path through the (music) WMA parser (for handling an undocumented aspect of the file format) that **allows arbitrary length reads to be specified**
  - These two features together allow a buffer overflow
- **[How?]** Authors developed their own native code debugger for the media player to determine how to overwrite function pointers as well as contents of the state variables
- **[Wow!!]** They modified a WMA audio file such that, when burned onto a CD, it plays perfectly on a PC but sends arbitrary CAN packets of attacker's choosing when played by the car's media player

Background: PassThru for OBD-II

- Since 2004, the Environmental Protection Agency has mandated that all new cars in the U.S. support the SAE J2534 “PassThru” standard.
  - Windows API that provides a standard, programming interface to communicate with a car’s internal buses.
- Typically implemented as a Windows dynamic link library (DLL) that communicates over a wired or wireless network with the reprogramming and diagnostic tool called the “PassThru device”.
  - The device consists of a popular SoC microprocessor running a variant of Linux as well as multiple network interfaces, including USB, WiFi, and a connector for plugging into the car’s OBD-II port.

Pass-Thru Diagnostics



Attack 1 on OBD-II PassThru

- After booting up, the device periodically advertises its presence by sending a UDP multicast packet on each network to which it is connected, **communicating both its IP address and a TCP port for receiving client requests**.
- **Problem:** Communication between the client application and the PassThru device is **unauthenticated**; it relies on security of the communication link (e.g., WiFi).
  - An attack on this network **can reprogram the PassThru device and send it instructions** to upload malicious software to the car.

Attack 2 on OBD-II

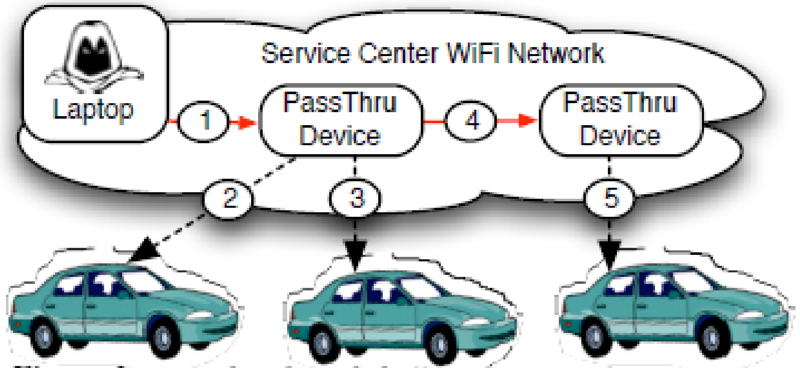


Figure 2: *PassThru-based shell-injection exploit scenario.* The adversary gains access to the service center network (e.g., by compromising an employee laptop), then (1) compromises any PassThru devices on the network, each of which compromise any cars they are used to service (2 and 3), installing Trojan horses to be activated based on some environmental trigger. The PassThru device also (4) spreads virally to other PassThru devices (e.g., if a device is loaned to other shops) which can repeat the same process (5).

Why Possible?

- **Background:** Recall that the PassThru device **exports a proprietary, unauthenticated API** for configuring its network state (e.g., for setting with which WiFi SSID it should associate).
- **Issue 1:** Input validation bugs in the implementation allow an attacker to **run arbitrary (Bourne) Shell commands via shell-injection.**
- **Issue 2:** Underlying implementation had telnet, ftp and nc (netcat) which **could be used to open more connections** (exacerbated by a poor choice of root password).
- **Combined:** Made a worm that actively seeks out and spreads to other PassThru devices in range.

Attack 3: Bluetooth Attacks

- A popular embedded implementation of the Bluetooth protocol stack and a sample hands-free application were present in the car.
  - The interface to this program and the rest of the telematics system appeared to be custom-built.
- In the custom interface code, over **20 calls to strcpy** were present, providing opportunities for **buffer attacks**. Why?
  - It performs no bounds checking on the destination buffer
  - It blindly copies until it hits a null terminator in the source string
  - It provides no way to limit how many bytes are copied
- Authors explored and found one such attack.
  - However, attack requires its device to be paired with the car.

Attack 3 Cont’d: Pairing attack on Bluetooth

- **Step 1:** Attacker must learn car’s Bluetooth MAC address
  - One can sniff it when the car’s Bluetooth starts
  - One can also sniff it via a previously paired device (owner’s cell phone)
- **Step 2:** After initiating the pairing request on the MAC address, car requires the user to enter the PIN number on the phone
  - Using a simple laptop to issue pairing requests, one can brute force this PIN at a rate of 8 to 9 PINs/min for an average of 10 hours per car.
- This requires a significant effort in development time and an extended period of proximity to the vehicle but might have promise **in a public garage, for any arbitrary car** (not a specific target).

Attack 4: Long Range – Cellular (Background)

- Telematics unit is equipped with a cell phone interface supporting voice, SMS and 3G data.
  - Unit uses voice channel for critical telematics functions (crash notification).
- Manufacturer uses the Airbiquity *aqLink* software modem to covert between analog waveforms and digital bits.
  - aqLink software is common to virtually all popular North American telematics offerings today.
- **How did they launch the attack?** The authors also reverse engineered telematics unit's own proprietary command protocol.

Cellular Gateway Vulnerability

- **Issue 1:** AqLink code explicitly supports packet sizes up to **1024 bytes**.
  - The custom code **assumes that packets will never exceed 100 bytes** (presumably since well-formatted command messages are always smaller).
- **Outcome:** Leads to an exploitable stack-based **buffer overflow vulnerability**.
- **How?** Attack uses lowest level of the protocol stack to bypass higher-level authentication checks.

- **Issue 2:** Hard to deploy directly – buffer overflow required sending over **300 bytes**.
- **Caveat 1:** aqLink protocol has a maximum effective throughput of about **21 bytes a second** so it takes 14 seconds to transmit.
- **Caveat 2:** The protocol closes connection if there is no response in **12 seconds**.

Solution: get more time, modify this 12s timeout value to a larger value via another vulnerability

Attack 5: Remote Monitoring

- **How?** Modified the telematics unit to support an IRC client
  - Showed how attackers controlling the audio system could record conversations in the car
- Due to the high bandwidth, they were able to easily upload the saved files using FTP provided by the telematics unit.
- Attackers could have the car periodically “tweet” the GPS location of the vehicle.

Hacking Automobiles

These 20-something hackers won $375,000 and a Model 3 for finding a Tesla bug, March 22, 2019

https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ July 21, 2015
Kill a Jeep’s engine remotely

https://techcrunch.com/2020/08/06/security-bugs-mercedes-benz-hack/, August 6, 2020
Qihoo 360, found more than a dozen vulnerabilities in a Mercedes-Benz E-Class car that allowed them to remotely open its doors and start the engine

https://www.wired.com/story/hackers-can-clone-millions-of-toyota-hyundai-kia-keys/
March 5, 2020
New vulnerabilities are found in the encryption systems used by immobilizers, the radio-enabled devices inside of cars that communicate at close range with a key fob to unlock the car's ignition and allow it to start.

Discussion

- What changes are occurring in the automotive sector?
  - How might these changes affect security and privacy?

- What threats might exist for other safety-critical embedded devices like medical devices?