How to Hack Your Mini Cooper: *Reverse Engineering CAN Messages on Passenger Automobiles*

Jason Staggs

isec™

The University of Tulsa
Institute for Information Security

Who is this guy?

- Jason Staggs
  - Graduate Research Assistant
    - Institute for Information Security (iSec)
    - Crash Reconstruction Research Consortium (TU-CRRC)
  - TRUE Digital Security
    - Cyber Security Analyst

Why do we hack cars?

- Related work
  - *“Comprehensive Experimental Analyses of Automotive Attack Surfaces”*
  - *“Experimental Security Analysis of a Modern Automobile”*
- Understanding computer and network systems on cars
  - Underlying CAN protocol and components lack of authentication and verification of messages
- Understanding potential points of vulnerability
  - Vehicle network security is in its infancy
- But most importantly…

To prevent this..

From turning into this..

Because of this..

CAN Clock Project

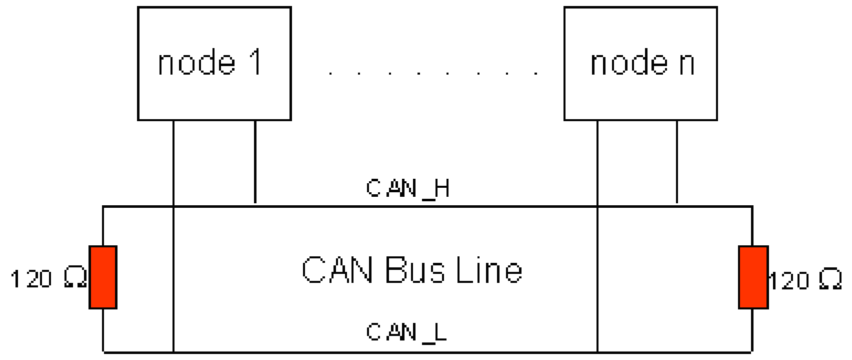
- Research project developed as a proof of concept
  - Manipulating CAN nodes via CAN network
  - Reverse Engineering CAN messages
  - 2003 Mini Cooper

Background of vehicle communication networks

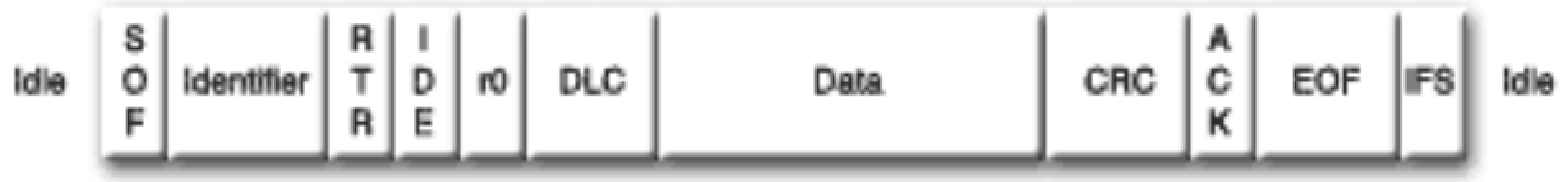
- Began in 1980s with General Motors
- Common vehicle Protocols
  - CAN (Most widely used among manufactures)
  - FlexRay
  - KW2000
  - LIN
  - J1850 (GM/Chrysler)
  - J1939 (Heavy Trucks)
  - J1708/J1587 (Being phased out due to J1939)
- 2008: All US cars use CAN for mandated EPA diag.

Controller Area Networks

- Bosh CAN standard
  - Developed in the 80s
  - European Manufactures were early adopters
  - Standard Format
    - 11-bit ID header
    - Mfg. use of proprietary IDs for each of their CAN components
  - Extended Format
    - 29-bit ID header
    - Used extensively by J1939

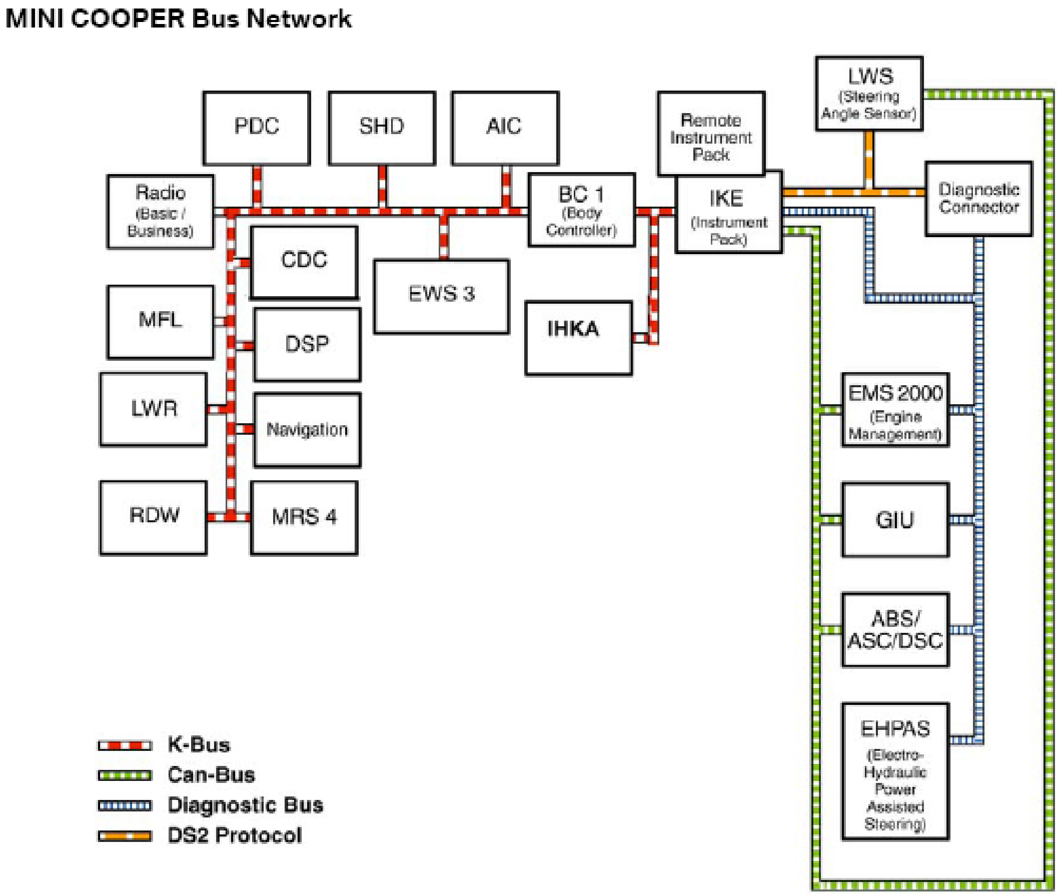


CAN Frame



- SOF – Start of Frame
- Identifier – Unique identifier for message along with priority
- RTR – Remote Transmission Request
- IDE – Identifier extension (distinguishes between CAN standard and CAN extended)
- DLC – Data Length Code (frames have up to 8 bytes of data)
- CRC – Cyclic Redundant Check sum
- ACK – Acknowledge
- EOF – End of Frame
- IFS – Intermission Frame Space

Interconnected vehicle networks



Electronic Control Units (ECUs)

- ECUs designed to control :
  - Vehicle safety systems
    - Engine control unit
    - ABS braking system
    - Door locks
  - Infotainment systems
    - Radio Deck
    - HID units
  - The list goes on
- Programmable ECUs
  - Allows MFGs to update firmware on ECUs
- Average modern day car has ~70 ECUs

Reverse Engineering CAN Messages

- What we want to do:
  - Manipulate CAN enabled vehicle components
- Problem:
  - Manufactures do not publish CAN message ID information about their various CAN components
- Solution:
  - A method for visually correlating physical system interactions with identifiable patterns. (Humans are good at this)
  - Brute force (Tedious, and messy)

Reverse Engineering CAN Messages

- Passively captured CAN data during a staged test run
  - In this case it was a staged automotive collision.. ☺
  - Mini Cooper vs. GMC Envoy (Check out TU-CRRC website for killer videos)
  - Data capture lasted for roughly 90 seconds
- Data Log gives us ~106,000 data entries of CAN messages

crash.wmv

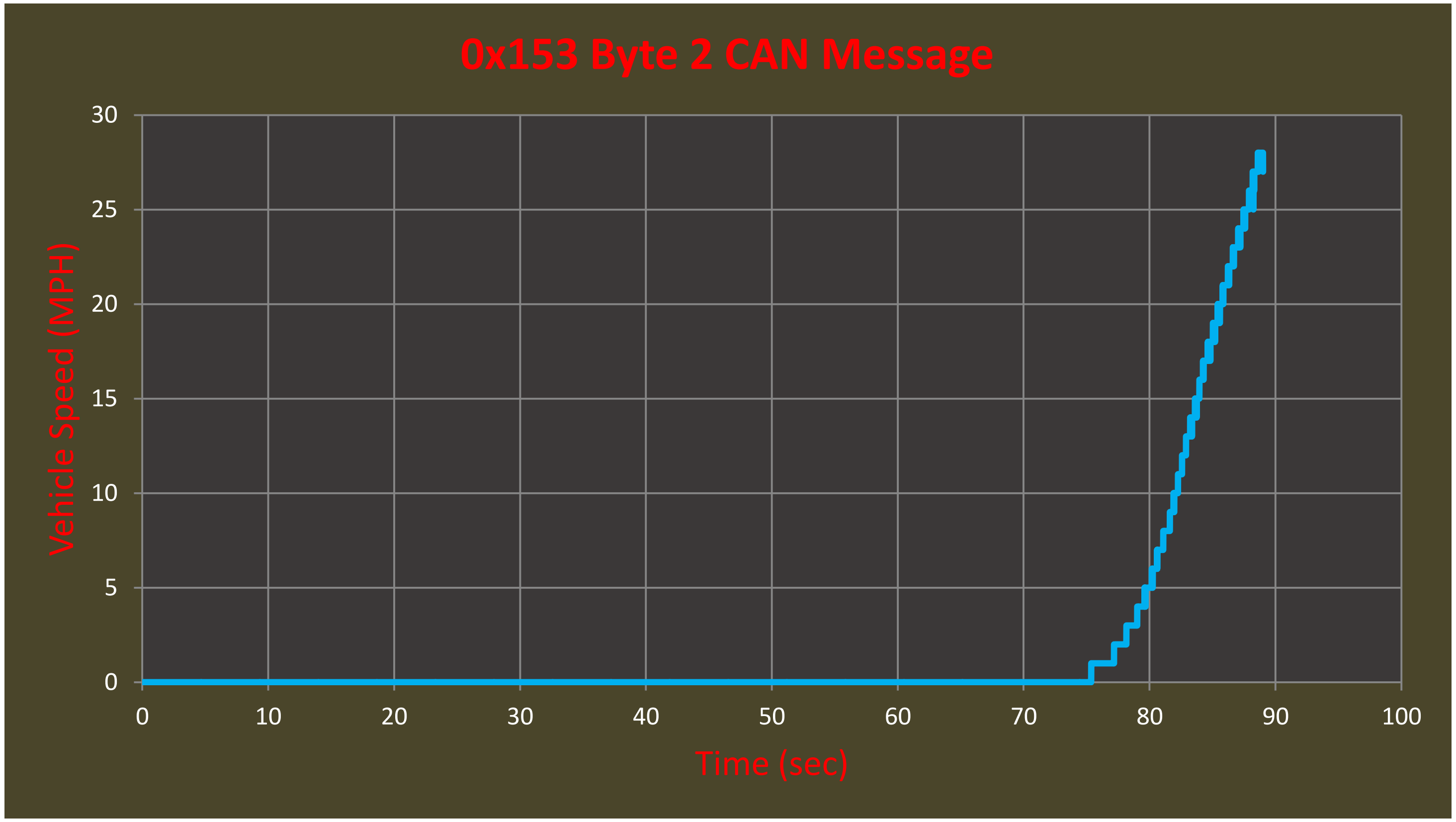
```
Dearborn Group Format x15
Head on Crash for IATAI
Tue Sep 20 16:34:00 2011

Tue Sep 20 16:35:47 2011

106600
 Trigger Frame
Absolute
Timestamp,Channel,Frame ID,Frame Acronym,Protocol,DataCount,Data,Tx/Rx
11:55:49:668:810.2.316.316.CAN - STD.8.01 00 00 00 00 00 00 00.Rx.
11:55:49:668:960.2.336.336.CAN - STD.8.00 00 FE 02 6C 12 9C 89.Rx.
11:55:49:669:210.2.329.329.CAN - STD.8.C0 61 00 00 00 00 00 00.Rx.
11:55:49:669:440.2.153.153.CAN - STD.8.10 50 00 00 00 FF 00 80.Rx.
11:55:49:669:690.2.1F0.1F0.CAN - STD.8.0A 20 0A 00 0A 00 0A 00.Rx.
11:55:49:669:930.2.1F3.1F3.CAN - STD.8.80 80 00 FF 41 7F 00 08.Rx.
11:55:49:670:190.2.1F8.1F8.CAN - STD.8.00 00 00 00 FE FF 00 00.Rx.
11:55:49:670:420.2.545.545.CAN - STD.8.12 00 00 00 00 00 00 00.Rx.
11:55:49:670:660.2.565.565.CAN - STD.8.50 20 66 02 00 02 00 63.Rx.
00:00:00:003:000.2.1F5.1F5.CAN - STD.8.60 80 00 00 80 E2 00 00.Rx.
00:00:00:003:310.2.153.153.CAN - STD.8.10 50 00 00 00 FF 00 80.Rx.
00:00:00:003:550.2.1F0.1F0.CAN - STD.8.0A 40 0A 00 0A 00 0A 00.Rx.
00:00:00:003:790.2.1F3.1F3.CAN - STD.8.00 81 00 FF 41 7F 00 08.Rx.
00:00:00:004:040.2.1F8.1F8.CAN - STD.8.00 00 00 00 FE FF 00 00.Rx.
00:00:00:005:820.2.316.316.CAN - STD.8.01 00 00 00 00 00 00 00.Rx.
00:00:00:006:040.2.336.336.CAN - STD.8.00 00 FE 02 6C 12 9C 89.Rx.
00:00:00:006:300.2.329.329.CAN - STD.8.C0 61 00 00 00 00 00 00.Rx.
00:00:00:006:540.2.545.545.CAN - STD.8.12 00 00 00 00 00 00 00.Rx.
00:00:00:006:780.2.565.565.CAN - STD.8.50 20 66 02 00 02 00 63.Rx.
00:00:00:010:360.2.153.153.CAN - STD.8.10 50 00 00 00 FF 00 80.Rx.
00:00:00:010:560.2.1F0.1F0.CAN - STD.8.0A 60 0A 00 0A 00 0A 00.Rx.
00:00:00:010:800.2.1F3.1F3.CAN - STD.8.40 80 00 FF 41 7F 00 08.Rx.
00:00:00:011:060.2.1F8.1F8.CAN - STD.8.00 00 00 00 FE FF 00 00.Rx.
00:00:00:011:390.2.1F5.1F5.CAN - STD.8.60 80 00 00 80 F2 94 05.Rx.
00:00:00:015:830.2.316.316.CAN - STD.8.01 00 00 00 00 00 00 00.Rx.
00:00:00:016:060.2.336.336.CAN - STD.8.00 00 FE 02 6C 12 9C 89.Rx.
00:00:00:016:310.2.329.329.CAN - STD.8.C0 61 00 00 00 00 00 00.Rx.
00:00:00:016:550.2.545.545.CAN - STD.8.12 00 00 00 00 00 00 00.Rx.
00:00:00:016:780.2.565.565.CAN - STD.8.50 20 66 02 00 02 00 63.Rx.
00:00:00:017:360.2.153.153.CAN - STD.8.10 50 00 00 00 FF 00 80.Rx.
```

CAN Data Log

- Contained ~106,000 data entries
- Bash “cut –d. –f3 cooperheadion.txt | sort | uniq –c”
  - Only 15 Unique CAN IDs!?

ID Occurrences	CAN IDs
12706	**153**
12706	1F0
12706	1F3
9460	1F5
12707	1F8
8899	**316**
8899	329

Visually identifying CAN messages of interest



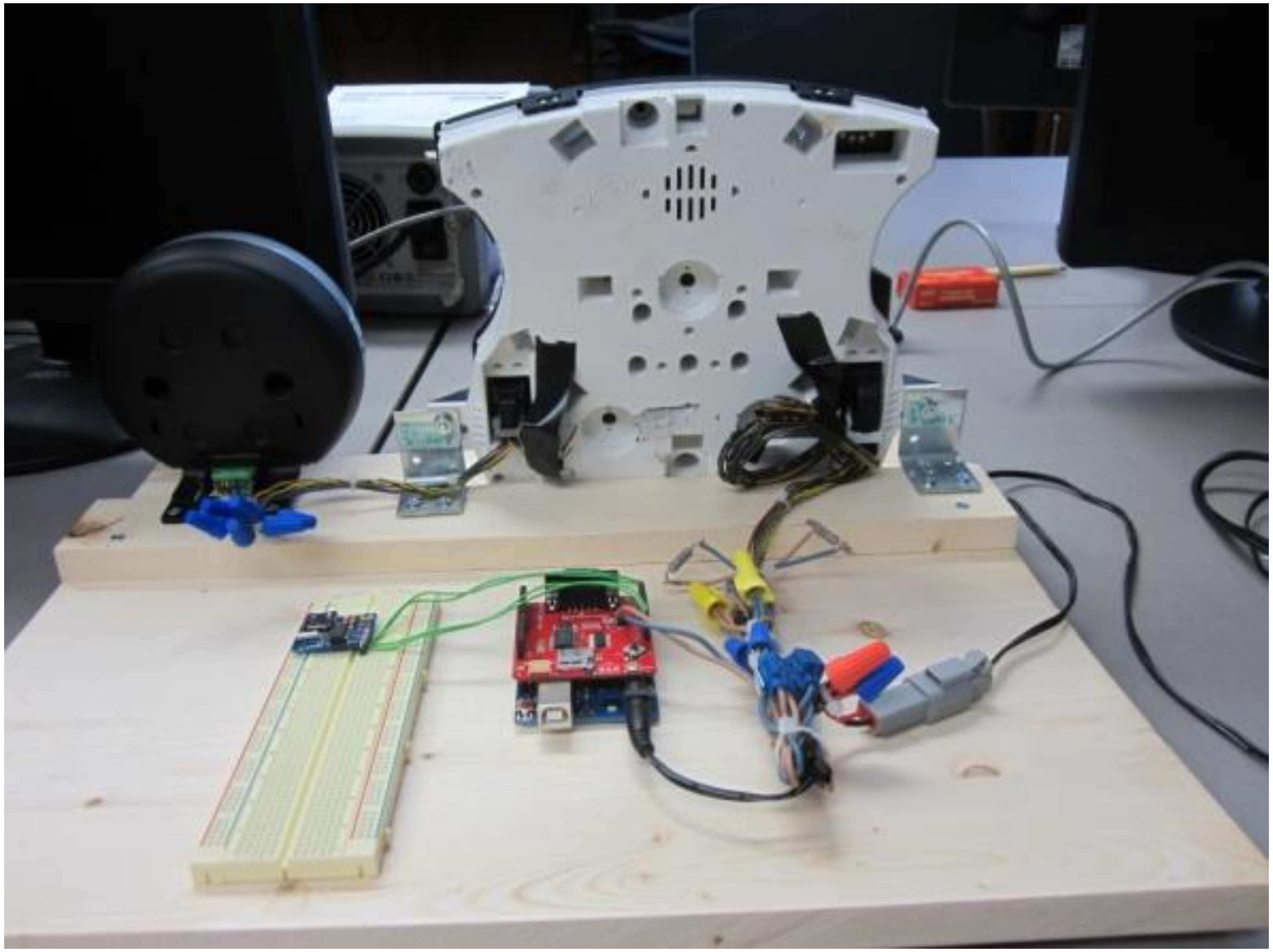
Reverse Engineering CAN Messages

- Speedometer and Tachometer CAN IDs
  - 2 methods
    - For each CAN ID, plot data values vs. timestamp in order to determine physical significance.
    - Given possible CAN IDs, fuzz data fields until needles start moving

CAN Message ID	Description
0x153 Byte 2	Speedometer (Vehicle Speed)
0x316 Byte 3	Tachometer (Engine Speed)
0x329	Various indicator lights
0x61A	Controls the messages being displayed on the tachometer LED screen.
0x61F	Tachometer along with various indicator lights

Building the CAN network

- CAN Bus
  - 18 gauge wire
  - 2 x 120 ohms terminating resistors
  - 12V DC power source
  - Arduino Uno microcontroller
  - CAN Bus Shield
    - MCP2515 CAN controller
    - MCP2551 CAN transceiver
  - Mini Cooper Instrument Cluster
  - Real time clock module RTC (for clock mode)



Proof of Concept

- Talking CAN with Arduino
  - Arduino and CAN Controller Libraries
    - MCP2515 (Communication with CAN transceiver)
    - SPI (Used for communications between Arduino and CAN shield)
- 2 Modes of operation
  - Clock Mode
  - Demo Mode

Demo

Gaining physical access to CAN

- Via OBD2
- Tapping the CAN bus (vampire tap)
  - Under the hood
  - Breaking a powered side view mirror
  - Etc.
- 0 to pwned for less then $100
  - Rogue Arduino CAN node
- Potential conspirators
  - Mechanics
  - Car Rentals
  - Coworkers/Family/Friends/Ex-girlfriends/etc.

Future Work / Conclusion

- Access control between vehicle network components
  - ECU to ECU
  - OBD2 to ECU
- Applying conventional NIPS & firewall methods to CAN
  - Message anomaly prevention depending on context?

For more Information

- TU Research
  - http://isec.utulsa.edu/
  - http://tucrrc.utulsa.edu/ ⟵ Check out our research and crash tests ☺
  - http://tucrrc.utulsa.edu/canclock/
- CAN Standards/Docs
  - http://esd.cs.ucr.edu/webres/can20.pdf (CAN 2.0 Spec)
  - http://www.sae.org/standards/

Questions??

- jason-staggs@utulsa.edu