Hacking cars in the style of Stuxnet

András Szijj¹, Levente Buttyán¹, Zsolt Szalay²
¹ CrySyS Lab, Department of Networked Systems and Services
² Department of Automobiles and Vehicle Manufacturing
Budapest University of Technology and Economics
Introduction

- modern cars are full of embedded controllers (ECUs)
- they are connected by internal networks (e.g., CAN)
- they have a number of external interfaces (e.g., Bluetooth, GPS, ...)

→ cyber attacks against cars became a plausible threat
Comprehensive Experimental Analyses of Automotive Attack Surfaces

Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage
University of California, San Diego

Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno
University of Washington

Abstract
Modern automobiles are pervasively computerized, and hence potentially vulnerable to attack. However, while previous research has shown that the internal networks within some modern cars are insecure, the associated threat model — requiring prior physical access — has justifiably been viewed as unrealistic. Thus, it remains an open question if automobiles can also be susceptible to remote compromise. Our work seeks to put this question to rest by systematically analyzing the external attack surfaces.

This situation suggests a significant gap in knowledge, and one with considerable practical import. To what extent are external attacks possible, to what extent are they practical, and what vectors represent the greatest risks? Is the etiology of such vulnerabilities the same as for desktop software and can we think of defense in the same manner? Our research seeks to fill this knowledge gap through a systematic and empirical analysis of the remote attack surface of late model mass-production sedan.

We make four principal contributions:
Checkoway et al. (Usenix Security, 2011)

- the paper shows that cars can be compromised remotely
  - systematic overview of the attack surface
    - indirect physical access (e.g., mechanics tools, CD players)
    - short range wireless access (e.g., Bluetooth, WiFi, wireless TPM)
    - long range wireless access (e.g., cellular)
  - proof-of-concept demonstrations for all possible attack vectors
    - vulnerable diagnostics equipment widely used by mechanics
    - media player playing a specially modified song in WMA format
    - vulnerabilities in hands-free Bluetooth functionality
    - calling the car’s cellular modem and playing a carefully crafted audio signal encoding both an exploit and a bootstrap loader for additional remote-control functionality
Car Hacking: The Content

By Chris Valasek @nudehaberdasher and Charlie Miller @0xcharlie

Hi Everyone,
As promised, Charlie and I are releasing all of our tools and data, along with our white paper. We hope that these items will help others get involved in automotive security research. The paper is pretty refined but the tools are a snapshot of what we had. There are probably some things that are deprecated or do not work, but things like ECOMCat and ecomcat_api should really be all you need to start with your projects. Thanks again for all the support!

Content: http://illmatics.com/content.zip

Paper:
http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf

Adventures in Automotive Networks and Control Units

By Dr. Charlie Miller & Chris Valasek
Work by Charlie Miller and Chris Valasek

Adventures in Automotive Networks and Control Units

Dr. Charlie Miller (@0xcharlie)
Chris Valasek (@nudehaberdasher)
Work by Charlie Miller and Chris Valasek

- remote wireless control over the Internet
- exploiting a bug in the car’s WiFi hotspot
Other examples

FM 99.9, Radio Virus: Exploiting FM Radio Broadcasts for Malware Deployment

Earlence Fernandes, Bruno Crispo, Senior Member, IEEE, and Mauro Conti, Member, IEEE

Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study

Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh
Wenyuan Xu, Marco Gruteser, Wade Trappe, Ivan Seskar *

Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars

Aurélien Francillon, Boris Danev, Srdjan Capkun
Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland
{aurelien.francillon, boris.danev, srdjan.capkun}@inf.ethz.ch
remote attacks
Putting things in perspectives

- remote attacks
  - are intriguing and scary
  - can attract media attention
  - but their real risk is unclear...
    - need exploitable vulnerability in an interface (e.g., GSM module)
    - finding such vulnerabilities is far from being trivial
      - reverse engineering embedded software (→ difficult)
      - very limited availability of information (→ frustrating)
      - risk of bricking relatively expensive equipment (→ expensive)
    - may not scale
      - a vulnerability in one brand may not work in any other brands of cars

- is there some fruits hanging lower than remote attacks?
How Stuxnet worked?

PC running WinCC PLC
management software

PLC controlling the
uranium centrifuges

uranium centrifuges

Stuxnet infected PCs, and
took over the communication
between the PC and the PLC

then modified
the PLC program

modified program
destroyed centrifuges

- exploited vulnerabilities in Windows
- replaced the DLL responsible for communications with the PLC
A blueprint for attacking embedded system

PC running WinCC PLC management software --> PLC controlling the uranium centrifuges -- uranium centrifuges

PC running a vehicle diagnostic software --> ECU controlling some function of the vehicle --> vehicle
Why is this worrisome?

- PCs in repair shops and garages are vulnerable
  - probably connected to the Internet
  - probably allow for connecting USB sticks
  - probably poorly maintained and administered
  - probably used not only for running diagnostic programs

  → it is relatively easy to infect them even with known malware

- malware can compromise diagnostic applications, and implement stealth functionality
  - almost direct access to internal components (via the OBD2 interface)
  - mainly needs standard reverse engineering skills in a PC environment
  - does not require special car electronics know-how

- this scales better than remote attacks
  - same software is usually compatible with multiple different car brands
  - every car is taken to the repair shop regularly
Proof of Concept

- **objective:**
  - demonstrate *in practice* that a Stuxnet-style attack is easy to implement against cars by minimal modification of a diagnostic application
    - in our test environment we had access to an Audi TT
    - we have chosen a widely-used, third-party diagnostic application that is compatible with cars from the Volkswagen group
  - the modifications should allow for Man-in-the-Middle attacks between the application and the car (i.e., eavesdropping and modifying messages stealthily)

- **assumptions:**
  - we assume that the PC that runs the diagnostic application is already infected by malware
  - the malware can carry out the modifications we propose on the diagnostic application
  - the diagnostic application has the necessary licenses and credentials to access the car when connected via the appropriate diagnostic cable (available in the repair shop)
Outline

- system model
- protection mechanisms
- attack techniques
  - our DLL replacement attack
  - protocol reverse engineering
    - message formats
    - checksum computation
    - encryption scheme
  - man-in-the-middle attacks
    - logging and replaying sessions
    - modifying messages on-the-fly
  - experiments
- conclusions and outlook
System model

- Diagnostic application
- DLLs
- Function driver (FTDI)
- Bus driver (USB)
- Diagnostic cable
- Gateway
- OBD/CAN interface
- Diagnostic PC
- Vehicle

CrySyS Lab, Budapest :: www.crysys.hu
System model

- diagnostic PC
  - diagnostic application
  - DLLs
  - function driver (FTDI)
  - bus driver (USB)

- vehicle
  - holds necessary keys and protocol implementations for accessing a wide variety of cars via the OBD interface
  - implements diagnostic functions (reading and setting parameters)

- diagnostic cable
- gateway
- OBD/CAN interface
System model

- diagnostic application
- DLLs
- function driver (FTDI)
- bus driver (USB)
- diagnostic cable
- gateway
- diagnostic PC
- vehicle

Implement functions for communicating with the cable
System model

- diagnostic application
- DLLs
- function driver (FTDI)
- bus driver (USB)

implement low level communication protocols

- diagnostic cable
- gateway
- OBD/CAN interface

- diagnostic PC
- vehicle
System model

- diagnostic PC
- vehicle

- diagnostic application
- DLLs
- function driver (FTDI)
- bus driver (USB)

- diagnostic cable
  - provides physical connection between the PC and the car
  - contains a microcontroller (e.g., ATmega) for message processing, protocol negotiations, and licence verification

- gateway
- OBD/CAN interface

CrySyS Lab, Budapest :: www.crysys.hu
System model

**diagnostic PC**

**vehicle**

- **OBD**: On-Board Diagnostics standard interface for diagnostic purposes
- **CAN**: Controller Area Network a serial bus protocol for reliable and fast communication between Electronic Control Units (ECUs) inside the vehicle

**diagnostic cable**

- **OBD/CAN interface**

**System model components**

- **diagnostic application**
- **DLLs**
- **function driver (FTDI)**
- **bus driver (USB)**
- **gateway**

**OBD**: On-Board Diagnostics standard interface for diagnostic purposes

**CAN**: Controller Area Network a serial bus protocol for reliable and fast communication between Electronic Control Units (ECUs) inside the vehicle
Protection mechanisms

- **signed DLLs**
  - all the DLLs loaded by the diagnostic software are digitally signed
  - however, signatures on DLLs are not checked (or perhaps checked ”silently”)

- **program obfuscation**
  - the executable (PE) of the diagnostic software is obfuscated with some ”commonly used” methods to prevent static analysis
  - however, the program de-obfuscates itself in memory when launched
  - so, we could access the de-obfuscated binary by attaching a debugger to the running program when its window was displayed on the screen

- **license verification**
  - the running application reads specific memory blocks from the microcontroller in the cable that contains the license, and also from the FTDI chip’s EEPROM
  - it also performs some challenge-response type authentication during cable initialization
  - the cable we bought on-line (for a few tens of dollars) was verified successfully
Implementing a Man-in-the-Middle attack

- our goal was to implement a man-in-the-middle component between the diagnostic application and the vehicle, which can
  - eavesdrop communications (can help reverse engineering the protocol)
  - play back recorded messages to the car or to the diagnostic application
  - inject fake messages in the car

- one option was to modify the FTDI DLL (binary) loaded by the application
  - no strong verification of loaded 3rd party DLLs
Implementing a Man-in-the-Middle attack

- our goal was to implement a man-in-the-middle component between the diagnostic application and the vehicle, which can
  - eavesdrop communications (can help reverse engineering the protocol)
  - play back recorded messages to the car or to the diagnostic application
  - inject fake messages in the car

- one option was to modify the FTDI DLL (binary) loaded by the application
  - no strong verification of loaded 3rd party DLLs

- but it seemed even easier to create our own fake FTDI DLL that tampers with the messages and then redirects calls to the original FTDI DLL
DLL replacement

- Memory:
  - diag app.
  - Import table

- Hard disk:
  - diag app.
  - Import table
  - Fake DLL
    - Contains original DLL in a resource section
  - Original FTDI DLL

Fake DLL with original name
DLL replacement

memory

- diag app.
- import table
- fake DLL
- original FTDI DLL

hard disk

- diag app.
- import table
- fake DLL
- original FTDI DLL

fake DLL with original name
contains original DLL in a resource section
DLL replacement

memory
- diag app.
  - import table
  - fake DLL
  - original FTDI DLL

hard disk
- diag app.
  - import table
  - fake DLL
  - original FTDI DLL
- re-named fake DLL

original FTDI DLL

import table
DLL replacement

When application calls the read or the write function, our code performs MitM and redirects the call to the original function.
when the call returns, we have control again
Could have been made harder...

- verification of digital signature and CRC should be performed before loading any external components (DLLs, data files, …)

- after loading, integrity of external components in memory should be checked regularly

- these checks shouldn’t be triggered by just some condition, but one should rather integrate them into many of the calculations

- common or close to common protocol implementations should be avoided, and unusual and proprietary solutions should also be used

- one should use proper cryptography in an appropriate way
  - XORing with a static mask is not proper crypto
Protocol reverse engineering

- to reverse engineer the protocol we used the following:
  - our fake FTDI DLL to capture data between the diagnostic application and the cable
    - we used modified versions of the original FTDI DLL exports
      - modified FT_Read to capture incoming messages
      - modified FT_Write to capture outgoing messages
  - OllyDbg 2.01 to reverse the diagnostic application
  - CheatEngine 6.0 for memory scanning
  - HxD 1.7.7.0 hex editor to view/edit captured data
  - and some handmade tools for filtering captured data
Protocol reverse engineering

- ordinary messages

<table>
<thead>
<tr>
<th>header</th>
<th>data</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>53 12 B8</td>
<td>00 F2 00 00 00 08 20 00 0C 31 BB 01 03 00</td>
<td>A7</td>
</tr>
</tbody>
</table>
Protocol reverse engineering

- ordinary messages

<table>
<thead>
<tr>
<th>header</th>
<th>data</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>53 12 B8</td>
<td>00 F2 00 00 00 08 20 00 0C 31 BB 01 03 00</td>
<td>A7</td>
</tr>
</tbody>
</table>

- message type
  0xB8 – "request"
  0xB7 – "response"

- message size (including checksum)

- message direction
  0x53 – to cable
  0x4D – to application
Protocol reverse engineering

- ordinary messages

```
Protocol reverse engineering

<table>
<thead>
<tr>
<th>header</th>
<th>data</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>53 12 B8</td>
<td>00 F2 00 00 00 08 20 00 0C 31 BB 01 03 00</td>
<td>A7</td>
</tr>
</tbody>
</table>

- actual data block
- "sync" byte
- data length (including sync byte)
- target identifier
```
### Protocol reverse engineering

- **ordinary messages**

![Diagram of protocol data](image)

- **Header**
  - `53 12 B8`

- **Data**
  - `00 F2 00 00 00 08 20 00 0C 31 BB 01 03 00`

- **Checksum**
  - `A7`

- **Checksum byte computed as the XOR of all bytes of the message**
  - e.g., `0x53 + 0x12 + 0xB8 + ... + 0x00 = 0xA7`
Protocol reverse engineering

- ACK message

<table>
<thead>
<tr>
<th>Header</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D 04 FE</td>
<td>B7</td>
</tr>
</tbody>
</table>

- Message type: 0xFE – ACK message
- Message size (including checksum)
- Message direction:
  - 0x53 – to cable
  - 0x4D – to application
Protocol reverse engineering

- key exchange message

<table>
<thead>
<tr>
<th>header</th>
<th>data</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>53 14 B6</td>
<td>DA 6B 6B 34 34 FC FC C5 C5 8E 8E 56 56 1F 1F 1F</td>
<td>34</td>
</tr>
</tbody>
</table>

- message type: 0xB6 – key exchange
- message size (including checksum)
- message direction:
  - 0x53 – to cable
  - 0x4D – to application
Protocol reverse engineering

- encryption mechanism
  - the application and the cable share a random permutation of all byte values (0x00 – 0xFF) arranged in a table
  - the key value received by the cable in the key exchange message is interpreted as a set of indices into this table
  - the values selected by these indices from the table form a XOR mask
  - messages are encrypted by XORing them with this XOR mask
    - only message content is encrypted, headers remain clear
    - after encryption, checksum is re-computed
Encryption illustrated

key exchange message

53 14 B6
DA 6B 6B 34 34 FC FC C5 C5 8E 8E 56 56 1F 1F 1F 34

table containing shared permutation

XOR mask: 3D B4 B4 20 20 4F 4F 42 42 B5 B5 FC FC 9F 9F 9F

ordinary message

53 12 B8
00 F2 00 00 00 08 20 00 0C 31 BB 01 03 00 A7

encrypted message

53 12 B8
3D 46 B4 20 20 47 6F 42 4E 84 0E FD FF 9F 05
Encryption illustrated

key value: DA 6B 6B 34 34 FC FC C5 C5 8E 8E 56 56 1F 1F 1F

table with permutation

XOR mask: 3D B4 B4 20 20 4F 4F 42 42 B5 B5 FC FC 9F 9F 9F
Logging messages sent to the car

(1) when application calls the FT_write function

(2) we save message content

(3) and pass on the control to the original FTDI DLL
Logging messages received from the car

(1) when application calls the FT_read function
(2) we pass on the control to the original FTDI DLL
(3) when the call returns, we save the content of the response
(4) and return to the application
Logging entire sessions

- before beginning any kind of diagnostic operation the cable needs to be initialized (or “tested”)

- during initialization, the software examines capabilities of the cable (speed, limits, license) and the car (if cable is connected to a car)
  - initialization results are stored in temporary files (d1.bin, d2.bin, d3.bin, ...)

- usually, these tests are also run before any larger operation-block (e.g., before entering Airbag Control Module)

- the diagnostic software also checks license data stored in the cable
  - the cable needs to be connected to the CAN bus
  - one can bypass this by connecting DC 12V to the OBD2 connector pin-16 (+) and pin-4 (-, ground)
Logging entire sessions

- logs are simple binary files that contain messages sent between the application and the cable
- example:

<table>
<thead>
<tr>
<th>Offset (h)</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>05</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>07</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>A5</td>
<td>53</td>
<td>04</td>
<td>02</td>
<td>55</td>
<td>4D</td>
<td>07</td>
<td>02</td>
</tr>
<tr>
<td>00000010</td>
<td>01</td>
<td>54</td>
<td>44</td>
<td>59</td>
<td>0D</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0C</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>53</td>
<td>0D</td>
<td>09</td>
<td>01</td>
</tr>
<tr>
<td>00000020</td>
<td>CC</td>
<td>10</td>
<td>BA</td>
<td>8C</td>
<td>16</td>
<td>DF</td>
<td>A8</td>
<td>39</td>
<td>E4</td>
<td>4D</td>
<td>0C</td>
<td>09</td>
<td>28</td>
<td>87</td>
<td>CD</td>
<td>6D</td>
</tr>
<tr>
<td>00000030</td>
<td>A9</td>
<td>CC</td>
<td>34</td>
<td>6C</td>
<td>7A</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>14</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>53</td>
<td>04</td>
<td>04</td>
</tr>
<tr>
<td>00000040</td>
<td>53</td>
<td>4D</td>
<td>14</td>
<td>04</td>
<td>52</td>
<td>4F</td>
<td>53</td>
<td>53</td>
<td>54</td>
<td>45</td>
<td>43</td>
<td>48</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>A8</td>
</tr>
<tr>
<td>00000050</td>
<td>B0</td>
<td>E7</td>
<td>92</td>
<td>24</td>
<td>13</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>06</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>53</td>
<td>04</td>
<td>82</td>
</tr>
<tr>
<td>00000060</td>
<td>D5</td>
<td>4D</td>
<td>06</td>
<td>82</td>
<td>00</td>
<td>00</td>
<td>C9</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>05</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>53</td>
</tr>
<tr>
<td>00000070</td>
<td>04</td>
<td>0D</td>
<td>5A</td>
<td>4D</td>
<td>05</td>
<td>0D</td>
<td>02</td>
<td>47</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00000080</td>
<td>53</td>
<td>04</td>
<td>B0</td>
<td>E7</td>
<td>4D</td>
<td>04</td>
<td>FE</td>
<td>B7</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00000090</td>
<td>53</td>
<td>04</td>
<td>B1</td>
<td>E6</td>
<td>4D</td>
<td>04</td>
<td>FE</td>
<td>B7</td>
<td>07</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>000000A0</td>
<td>53</td>
<td>07</td>
<td>B2</td>
<td>00</td>
<td>B8</td>
<td>05</td>
<td>5B</td>
<td>4D</td>
<td>04</td>
<td>FE</td>
<td>B7</td>
<td>09</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>04</td>
</tr>
<tr>
<td>000000B0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>53</td>
<td>09</td>
<td>B3</td>
<td>00</td>
<td>FF</td>
<td>E0</td>
<td>FF</td>
<td>00</td>
<td>09</td>
<td>4D</td>
<td>04</td>
<td>FE</td>
<td>B7</td>
</tr>
<tr>
<td>000000C0</td>
<td>09</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>53</td>
<td>09</td>
<td>B3</td>
<td>01</td>
<td>FF</td>
<td>E0</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Logging entire sessions

- we made logs of:
  - port test
    - Port Status: OK, Interface: Found!, K1:OK, K2:OK, CAN:OK
    - operation succeeded
  - auto-scan (full scan)
    - Session Init, Scan of all controllers (ECUs), Session Close
    - operation succeeded
  - Airbag Control Module
    - Session Init, Enable/Disable front passenger airbag, Session Close
    - operation succeeded
  - ABS Brakes Control Module
    - Session Init, Enable/Disable ABS booster, Session Close
    - operation failed (maybe wasn’t supported)
Replaying sessions

- replay files are similar to logs, but they contain only messages to be sent to the cable (FT_Write)
- example:

<table>
<thead>
<tr>
<th>Offset (h)</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>A5</td>
<td>53</td>
<td>04</td>
<td>02</td>
<td>55</td>
<td>53</td>
<td>0D</td>
<td>09</td>
<td>01</td>
<td>0C</td>
<td>9E</td>
<td>7F</td>
<td>9E</td>
<td>93</td>
<td>5C</td>
<td>25</td>
</tr>
<tr>
<td>00000010</td>
<td>ED</td>
<td>22</td>
<td>53</td>
<td>04</td>
<td>04</td>
<td>53</td>
<td>53</td>
<td>04</td>
<td>82</td>
<td>D5</td>
<td>53</td>
<td>04</td>
<td>0D</td>
<td>5A</td>
<td>53</td>
<td>04</td>
</tr>
<tr>
<td>00000020</td>
<td>B0</td>
<td>E7</td>
<td>53</td>
<td>04</td>
<td>B1</td>
<td>E6</td>
<td>53</td>
<td>07</td>
<td>B2</td>
<td>00</td>
<td>B8</td>
<td>05</td>
<td>5B</td>
<td>53</td>
<td>09</td>
<td>B3</td>
</tr>
<tr>
<td>00000030</td>
<td>00</td>
<td>FF</td>
<td>E0</td>
<td>FF</td>
<td>00</td>
<td>09</td>
<td>53</td>
<td>09</td>
<td>B3</td>
<td>01</td>
<td>FF</td>
<td>E0</td>
<td>00</td>
<td>00</td>
<td>F7</td>
<td>53</td>
</tr>
<tr>
<td>00000040</td>
<td>09</td>
<td>B4</td>
<td>00</td>
<td>3F</td>
<td>E0</td>
<td>00</td>
<td>00</td>
<td>31</td>
<td>53</td>
<td>09</td>
<td>B4</td>
<td>01</td>
<td>43</td>
<td>E0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00000050</td>
<td>4C</td>
<td>53</td>
<td>09</td>
<td>B4</td>
<td>02</td>
<td>60</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>8C</td>
<td>53</td>
<td>09</td>
<td>B4</td>
<td>03</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00000060</td>
<td>00</td>
<td>00</td>
<td>ED</td>
<td>53</td>
<td>09</td>
<td>B4</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>EA</td>
<td>53</td>
<td>09</td>
<td>B4</td>
</tr>
<tr>
<td>00000070</td>
<td>EF</td>
<td>40</td>
<td>00</td>
<td>00</td>
<td>44</td>
<td>53</td>
<td>06</td>
<td>B5</td>
<td>00</td>
<td>64</td>
<td>84</td>
<td>53</td>
<td>06</td>
<td>B5</td>
<td>01</td>
<td>60</td>
</tr>
<tr>
<td>00000080</td>
<td>81</td>
<td>53</td>
<td>14</td>
<td>B6</td>
<td>FA</td>
<td>C3</td>
<td>54</td>
<td>1D</td>
<td>E6</td>
<td>AE</td>
<td>3F</td>
<td>08</td>
<td>D1</td>
<td>99</td>
<td>62</td>
<td>2B</td>
</tr>
<tr>
<td>00000090</td>
<td>BC</td>
<td>85</td>
<td>4D</td>
<td>16</td>
<td>9D</td>
<td>53</td>
<td>11</td>
<td>B8</td>
<td>BB</td>
<td>85</td>
<td>90</td>
<td>B2</td>
<td>6C</td>
<td>BF</td>
<td>A2</td>
<td>1B</td>
</tr>
<tr>
<td>000000A0</td>
<td>17</td>
<td>32</td>
<td>7E</td>
<td>06</td>
<td>40</td>
<td>91</td>
<td>53</td>
<td>0B</td>
<td>B8</td>
<td>BB</td>
<td>A0</td>
<td>50</td>
<td>B2</td>
<td>6C</td>
<td>B9</td>
<td>15</td>
</tr>
<tr>
<td>000000B0</td>
<td>D9</td>
<td>53</td>
<td>04</td>
<td>A0</td>
<td>F7</td>
<td>53</td>
<td>04</td>
<td>16</td>
<td>41</td>
<td>53</td>
<td>06</td>
<td>0B</td>
<td>00</td>
<td>00</td>
<td>5E</td>
<td>53</td>
</tr>
<tr>
<td>000000C0</td>
<td>06</td>
<td>0B</td>
<td>01</td>
<td>00</td>
<td>5F</td>
<td>53</td>
<td>06</td>
<td>0B</td>
<td>02</td>
<td>00</td>
<td>5C</td>
<td>53</td>
<td>06</td>
<td>0B</td>
<td>03</td>
<td>00</td>
</tr>
</tbody>
</table>

Szijj, Buttyán, Szalay :: Hacking cars in the style of Stuxnet
CrySyS Lab, Budapest :: www.crysys.hu
Replaying sessions

- our replay tool is a separate process that uses the original FTDI DLL for writing to the cable

- cable initialization requires initializing the FTDI device correctly (i.e., setting baud rate, timeouts, …)
  - following functions are called with appropriate parameters:
    - `FT_SetLatencyTimer(device, 2);`
    - `FT_SetTimeouts(device, 1, 100);`
    - `FT_SetDataCharacteristics(device, 8, 0, 0);`
    - `FT_SetBaudRate(device, 115200);`

- then we can write with function `FT_Write`
  - after writing out a message we wait around 300 ms before writing the next message

- don’t forget to write `0xA5` (one-byte message) at the beginning
  - sort of session initialization
Switching off the airbag

- we could easily replay a previously recorded messages that switched the passenger airbag off
  - easy means that there’s no need to wait for any response, change encoding, ...
  - we just sent a previously recorded messages to the Airbag Control Module
    - Session Init, Disable front passenger airbag, Session Close
    - operation succeeded

- as our replay tool is a separate application, the replay message is invisible to the diagnostic application!
Modification of messages on-the-fly

- application sends messages to the cable in a byte-after-byte manner

- on-the-fly modification of messages requires
  - matching some pre-specifed sample in the byte sequence
  - and replacing follow-up bytes with a pre-specifed pattern

- easily done by the Man-in-the-Middle capability we have in our fake FTDI DLL
On-the-fly modification illustrated

sample to match: 53 12 B8 00 F2
replacement pattern: FF 01

original message

```
53 12 B8 00 F2 00 00 00 08 20 00 0C 31 BB 01 03 00 A7
```

modified message

```
53 12 B8 FF 01 00 00 08 20 00 0C 31 BB 01 03 00 B2
```

match!

MitM
re-compute checksum

original message

```
53 12 B8 00 F2 X
```

match!
Experiments

experiments were carried out during spring 2015
Example – Logging a full scan

- diagnostic application exports scan result as a simple log file
Example – Logging a full scan

Offset(h): 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000005F0: 8A FF 4A FF 00 0F 00 00 00 99 00 00 00 53 0F B8
00000600: 00 65 C0 00 00 05 10 00 02 1A 9B 00 4D 04 FE 00
00000610: 4D 0B B7 01 60 00 CE E7 01 B1 00 4D 12 B7 01 60
00000620: 00 4C 78 08 20 00 30 5A 9B 31 4B 30 00 4D 12 B7
00000630: 01 60 00 CE E7 08 21 39 30 37 35 33 30 4A 00 4D
00000640: 12 B7 01 60 00 4C 78 08 22 20 20 30 31 36 30 10
Offset(h): 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000001840: 00 0F 00 00 00 45 00 00 00 53 0F B8 00 E8 00 00
000001850: 00 05 15 00 02 1A 90 00 4D 04 FE 00 4D 0B B7 01
000001860: 60 00 4C 78 01 B6 00 4D 12 B7 01 60 00 CE E7 08
000001870: 21 00 13 5A 90 54 52 55 00 4D 12 B7 01 60 00 4C
000001880: 78 08 22 5A 5A 5A 38 4A 34 37 00 4D 12 B7 01 60
000001890: 00 CE E7 08 13 31 30 30 33 32 30 00 OB 00 00

Szijj, Buttyán, Szalay :: Hacking cars in the style of Stuxnet
CrySyS Lab, Budapest :: www.crysys.hu
we used our separate replay tool to replay back previously recorded passenger airbag enable and disable messages

after every replay we deleted all temporary config files and started the diagnostic application to check the results

all replays were successful
Conclusions

- cyber attacks on modern vehicles is a plausible threat
- lot of research on remote attacks, but ...
- a Stuxnet-style attack may have a higher risk

- we demonstrated *in practice* that such an attack is easy to implement against cars by minimal modification of a diagnostic application (could be done by a malware)
- our proof-of-concept implementation allows for Man-in-the-Middle attacks between the application and the car
- for illustration purposes, we switched off the passenger airbag stealthily with a replay attack
Outlook

the Internet-of-Things: billions of network enabled embedded devices
Outlook

the Internet-of-Things: billions of network enabled embedded devices

what can we *really* do about this?
Laboratory of Cryptography and System Security (CrySyS Lab)
Budapest University of Technology and Economics
www.crysys.hu

contact:
Levente Buttyán, PhD
Associate Professor, Head of the Lab
buttyan@crysys.hu