CopyCAN

An Error-Handling Protocol based Intrusion Detection System for Controller Area Network

Stefano Longari, Matteo Penco, Michele Carminati, Stefano Zanero
Politecnico di Milano

11 Nov 2019 - ACM CPS-SPC
Controller Area Network

De-facto standard in the automotive World
Is CAN key to automotive attacks?

Tesla hackers explain how they did it at Defcon

At the digital security conference, Kevin Mahaffey and Marc Rogers explain how they hacked a Tesla Model S -- and why you shouldn't be too alarmed.

BY ANTUAN GOODWIN | AUGUST 9, 2015 2:09 PM PDT

Note: As stated below, Tesla has already patched many of the vulnerabilities discussed here in a recent patch.

LAS VEGAS -- It is very difficult to hack a Tesla Model S, but it’s not impossible. Last week, researchers Kevin Mahaffey and Marc Rogers demonstrated that they were able to remotely unlock the Model S’ doors, start the vehicle and drive away. They were also able to issue a

At Defcon, Rogers and Mahaffey (left to right) explain what Tesla does right and where it was weak in designing the Model S’ information systems.

Antuan Goodwin/WIRED

The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse

Security researchers Charlie Miller and Chris Valasek.

ANTHONY GREENBERG | SECURITY 08.16.15 03:30 PM
What weaknesses are commonly abused?

- Keyless Ignition Module
- Engine Control Module
- Body Control Module

CANL 120Ω → CANH 120Ω

Broadcast Unauthenticated → Frame Injection
Can we detect these attacks?

The threat landscape is constantly changing: every new service based on a vehicle’s connectivity opens up new attack vectors. Besides that, attackers are also continuously perfecting their methods in exploiting existing protection mechanisms and finding new vulnerabilities. That’s why it’s not enough to guarantee state-of-the-art security at the point at which the vehicle reboots after production. Instead, the security has to extend to protect against attacks during the vehicle’s operating life. It has to reliably detect and analyze them so that suitable countermeasures can be taken immediately and effectively – for the vehicle in question and, if necessary, for the entire fleet.
How do automotive IDS work?

Industrial secret, however we can make an educated guess at some methods

- **Frequency** based
  - CAN messages are usually **periodic**
- **Specification** based
  - Set rules for the data field of the message
  - Potentially **dynamic** depending from message history
- **Machine Learning** based
  - Generally similar to specification based ones
  - **Mainly Academic**
How to evade an automotive IDS

- Specification based: Comply with the rules
- Frequency based: Comply with the frequency
- ML based: different forms of mimicry attacks
What if we manipulate/substitute a real frame?

- Specification based: Comply with the rules
- Frequency based: Comply with the frequency
- ML based: different forms of mimicry attacks
Data frames
CAN bus values

5V

2.5V

0V

1 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1
Dominant beats recessive - arbitration

ECU1

1 0 1

ECU2

1 0 0 1 1

ECU3

1 0 0 1 0 1 1 0

Time

\[\text{Data} = \{0000000010100000000001000000\}\]
CAN error handling

Reasons:
- Transceiver Fail
- CRC Computation error
- Channel Noise
- Faulty Device
- ...

120Ω
Can send error active flags
“000000”

ERROR
ACTIVE
CAN fault confinement

Can send error active flags
“000000”

Can send error passive flags
“111111”

ERROR ACTIVE

ERROR PASSIVE

counter > 127
reset or
counter < 128
CAN fault confinement

- Can send error active flags "000000"
- Can send error passive flags "111111"
- Shuts itself off the bus

- Counter > 127
- Counter > 255
- Reset or
- Counter < 128
- Reset or
- Detect 11 sequential "1" x128 bit times

ERROR ACTIVE → ERROR PASSIVE
BUS OFF
How do we Exploit this?

How do we convince the target ECU to kick itself off the network?

Keyless Ignition Module

Engine Control Module

Body Control Module

120Ω
How do we Exploit this?

For example like this.

0 overwrites 1.
How do we Exploit this?

The attacker can write that 0 over a 1. We just deleted the packet -> error counter +8

Now do it 32 times and the victim is Bus Off
Denial of Service for the sake of Denial of Service

e.g. Ransomware
Attack scenarios

Detection avoidance for spoofing attacks

- *Shut down* the victim ECU
- *Send* spoofed data
Can we detect the DoS?

We can read data from the bus
We can detect the attacker once he tries to spoof data after the DoS
We need to study more CAN specs! :(

List of rules that change the counters:

1. When a RECEIVER detects an error, the RECEIVE ERROR COUNT will be increased by 1, except when the detected error was a BIT ERROR during the sending of an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

2. When a RECEIVER detects a 'dominant' bit as the first bit after sending an ERROR FLAG the RECEIVE ERROR COUNT will be increased by 8.

3. When a TRANSMITTER sends an ERROR FLAG the TRANSMIT ERROR COUNT is increased by 8.

   Exception 1:
   If the TRANSMITTER is 'error passive' and detects an ACKNOWLEDGMENT ERROR because of not detecting a 'dominant' ACK and does not detect a 'dominant' bit while sending its PASSIVE ERROR FLAG.

   Exception 2:
   If the TRANSMITTER sends an ERROR FLAG because a STUFF ERROR occurred during ARBITRATION, and should have been 'recessive', and has been sent as 'recessive' but monitored as 'dominant'.

   In exceptions 1 and 2 the TRANSMIT ERROR COUNT is not changed.

4. If an TRANSMITTER detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG the TRANSMIT ERROR COUNT is increased by 8.

5. If an RECEIVER detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG the RECEIVE ERROR COUNT is increased by 8.

6. Any node tolerates up to 7 consecutive 'dominant' bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive 'dominant' bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive 'dominant' bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive 'dominant' bits every TRANSMITTER increases its TRANSMIT ERROR COUNT by 8 and every RECEIVER increases its RECEIVE ERROR COUNT by 8.

7. After the successful transmission of a message (getting ACK and no error until END OF FRAME is finished) the TRANSMIT ERROR COUNT is decreased by 1 unless it was already 0.

8. After the successful reception of a message (receipt without error up to the ACK SLOT and the successful sending of the ACK bit), the RECEIVE ERROR COUNT is decreased by 1, if it was between 1 and 127. If the RECEIVE ERROR COUNT was 0, it stays 0, and if it was greater than 127, then it will be set to a value between 119 and 127.

9. A node is 'error passive' when the TRANSMIT ERROR COUNT equals or exceeds 128, or when the RECEIVE ERROR COUNT equals or exceeds 128. An error condition letting a node become 'error passive' causes the node to send an ACTIVE ERROR FLAG.

10. A node is 'bus off' when the TRANSMIT ERROR COUNT is greater than or equal to 256.

11. An 'error passive' node becomes 'error active' again when both the TRANSMIT ERROR COUNT and the RECEIVE ERROR COUNT are less than or equal to 127.

12. An node which is 'bus off' is permitted to become 'error active' (no longer 'bus off') with its error counters both set to 0 after 128 occurrence of 11 consecutive 'recessive' bits have been monitored on the bus.
Not all of them...

List of rules that change the counters:

1. When a RECEIVER detects an error, the RECEIVE ERROR COUNT will be increased by 1, except when the detected error was a BIT ERROR during the sending of an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

2. When a RECEIVER detects a 'dominant' bit as the first bit after sending an ERROR FLAG the RECEIVE ERROR COUNT will be increased by 8.

3. When a TRANSMITTER sends an ERROR FLAG the TRANSMIT ERROR COUNT is increased by 8.

   Exception 1:
   If the TRANSMITTER is 'error passive' and detects an ACKNOWLEDGMENT ERROR because of not detecting a 'dominant' ACK and does not detect a 'dominant' bit while sending its PASSIVE ERROR FLAG.

   Exception 2:
   If the TRANSMITTER sends an ERROR FLAG because a STUFF ERROR occurred during ARBITRATION, and should have been 'recessive', and has been sent as 'recessive' but monitored as 'dominant'.

   In exceptions 1 and 2 the TRANSMIT ERROR COUNT is not changed.

4. If an TRANSMITTER detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG the TRANSMIT ERROR COUNT is increased by 8.

5. If an TRANSMITTER detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG the RECEIVE ERROR COUNT is increased by 8.

6. Any node tolerates up to 7 consecutive 'dominant' bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive 'dominant' bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive 'dominant' bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive 'dominant' bits every TRANSMITTER increases its TRANSMIT ERROR COUNT by 8 and every RECEIVER increases its RECEIVE ERROR COUNT by 8.

7. After the successful transmission of a message (getting ACK and no error until END OF FRAME is finished) the TRANSMIT ERROR COUNT is decreased by 1 unless it was already 0.

8. After the successful reception of a message (reception without error up to the ACK SLOT and the successful sending of the ACK bit), the RECEIVE ERROR COUNT is decreased by 1, if it was between 1 and 127. If the RECEIVE ERROR COUNT was 0, it stays 0, and if it was greater than 127, then it will be set to a value between 119 and 127.

9. A node is 'error passive' when the TRANSMIT ERROR COUNT equals or exceeds 128, or when the RECEIVE ERROR COUNT equals or exceeds 128. An error condition letting a node become 'error passive' causes the node to send an ACTIVE ERROR FLAG.

10. A node is 'bus off' when the TRANSMIT ERROR COUNT is greater than or equal to 256.

11. An 'error passive' node becomes 'error active' again when both the TRANSMIT ERROR COUNT and the RECEIVE ERROR COUNT are less than or equal to 127.

12. An node which is 'bus off' is permitted to become 'error active' (no longer 'bus off') with its error counters both set to 0 after 128 occurrence of 11 consecutive 'recessive' bits have been monitored on the bus.
Modify the CAN Controller
Complete IDS process: CopyCAN

1) Define which ECUs/IDs to defend
2) Monitor the bus from the beginning of communication
3) Count the TEC (Transmit Error Counter) of each ECU
4) Detect when the ECU goes Bus Off
5) If the ECU writes on the bus again, flag as attack.
Proof of Concept implementation:

Testbed to detect rules 4 and 6 “in the wild”:

Tests done 50
Frames sent per test 15000

IDS Never failed
## Our Miscalculation (and solution)

<table>
<thead>
<tr>
<th>...Data Field</th>
<th>Error Flag (Passive)</th>
<th>...Data Field</th>
<th>CRC... (End of Packet)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Victim</strong></td>
<td></td>
<td><strong>Attacker</strong></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0 0 1 1 1 1 1 1 1</td>
<td></td>
<td>0 1 1 0 1 1 1 0 1 1 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td><strong>Bus</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our Miscalculation (and solution)

The attacker has to send a valid packet... But the victim wants to send it too!

```
Victim +8 +8
Attacker
Bus x16
```

```
Victim +8 -1 +8 -1
Attacker
Bus x19
```

```
Victim +8 -1 -1 -1 ..x7.. +8 -1
Attacker
Bus x847
```

-> Easy Frequency Analysis
Conclusions

DoS for CAN is not preventable...

... but the goals of the attacker may be!

Thanks!

For any questions:

✉️ < stefano.longari@polimi.it >  🐦 @ascarecrowhat
Rule 6

Any node tolerates up to 7 consecutive ‘dominant’ bits after sending an **ACTIVE ERROR FLAG**, **PASSIVE ERROR FLAG** or **OVERLOAD FLAG**. After detecting the 14th consecutive ‘dominant’ bit (in case of an **ACTIVE ERROR FLAG** or an **OVERLOAD FLAG**) or after detecting the 8th consecutive ‘dominant’ bit following a **PASSIVE ERROR FLAG**, and after each sequence of additional 8 consecutive ‘dominant’ bits every **TRANSMITTER** increases its **TRANSMIT ERROR COUNT** by 8 and every **RECEIVER** increases its **RECEIVE ERROR COUNT** by 8.
Rule 6

Cannot let the attacker *bypass* the whole IDS, so we always consider *case 1*

<table>
<thead>
<tr>
<th>Bus</th>
<th>0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Victim 1</td>
<td>0 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>Attacker 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Victim 2</td>
<td>0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>Attacker 2</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>