The CAN Link-Layer,

or how we implemented a broken protocol and can we fix it?

Stefano Longari,
Politecnico di Milano

HACK IN BO®
Spring 2022 Edition
18° EDIZIONE

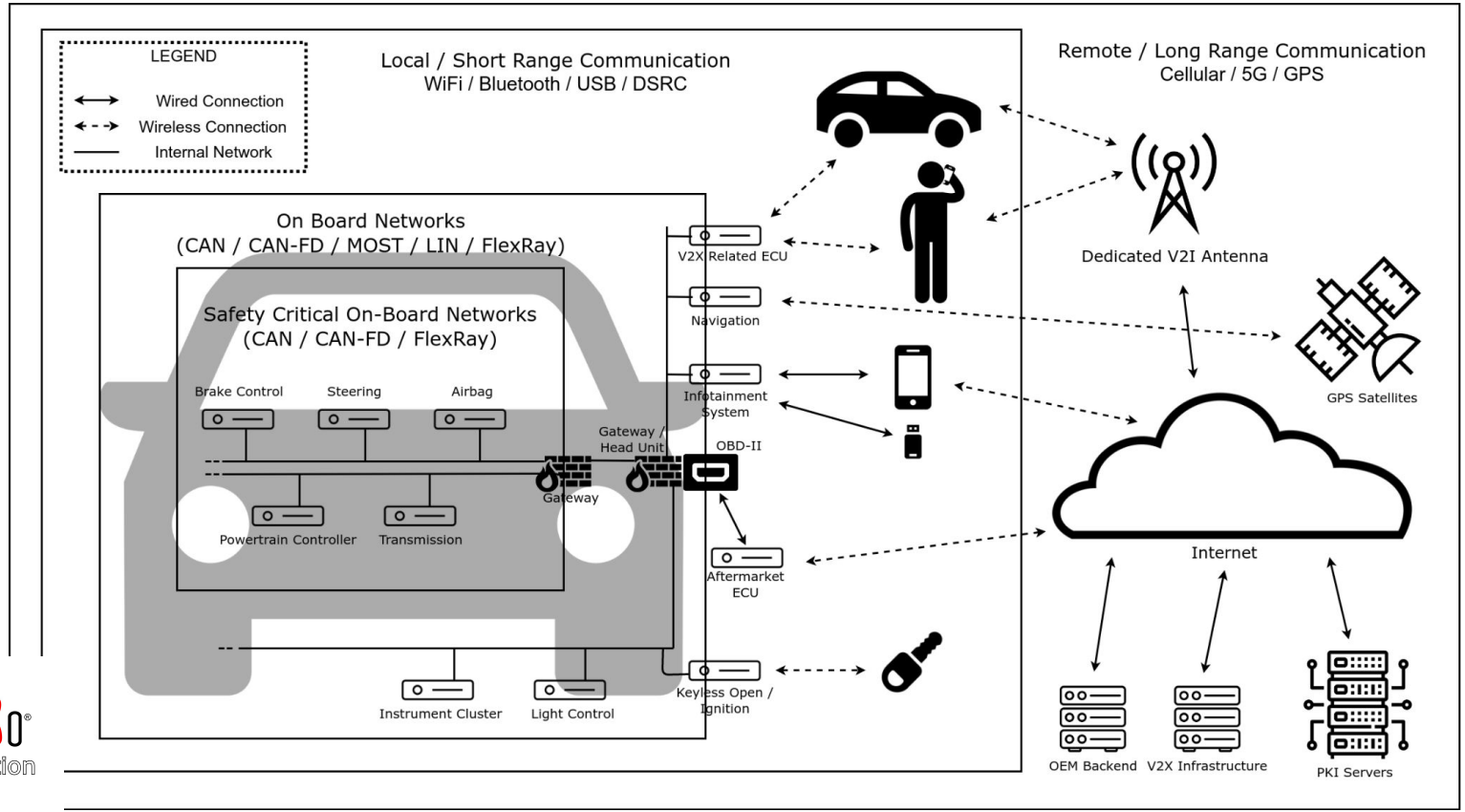
$whoami

I finished my Ph.D. in 2021 and I'm currently a post-doc researcher at Politecnico di Milano.

My main (research) interest is the security of **cyber-physical systems** and, in particular, vehicles.

Outside the university motorcycles, bouldering, and beer :)

The automotive ecosystem



Automotive Attacks and CAN

AUTO TECH

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/CNET

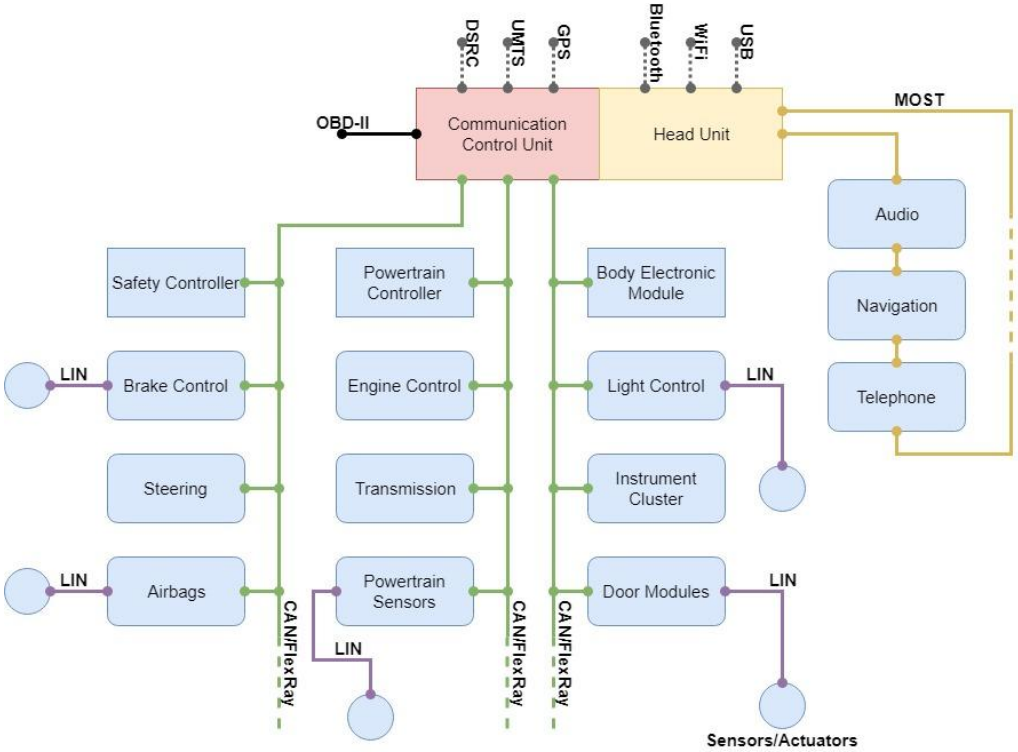
The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse

ANDY GREENBERG SECURITY 08.01.16 03:30 PM

THE JEEP HACKERS ARE BACK TO PROVE CAR HACKING CAN GET MUCH WORSE

Security researchers Charlie Miller and Chris Valasek. WHITNEY CURTIS FOR WIRED

The automotive ecosystem



Controller Area Network

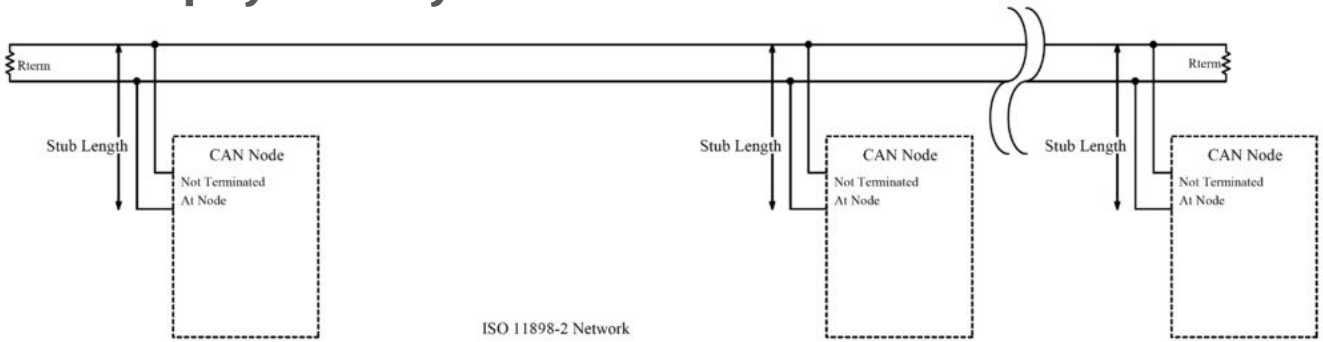
1980's protocol

Developed with focus on safety

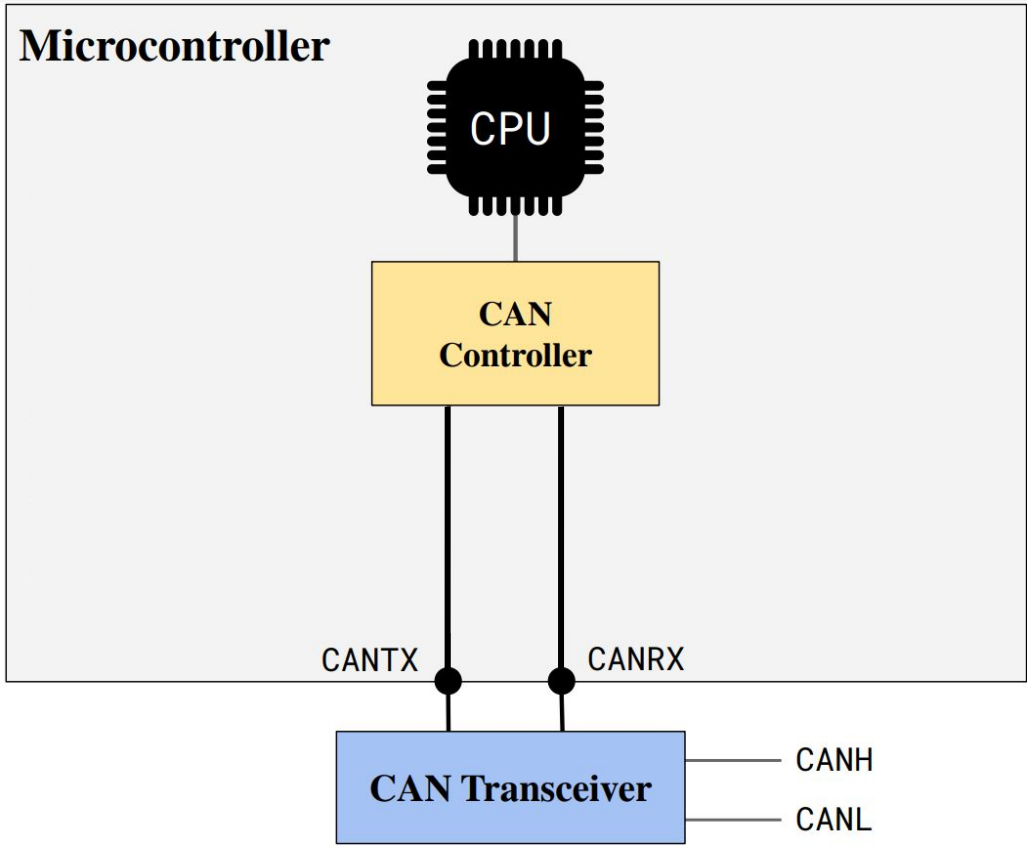
Baud rate of 1Mbps max

Multi-master bus topology

Data-Link and physical layers



Controller Area Network

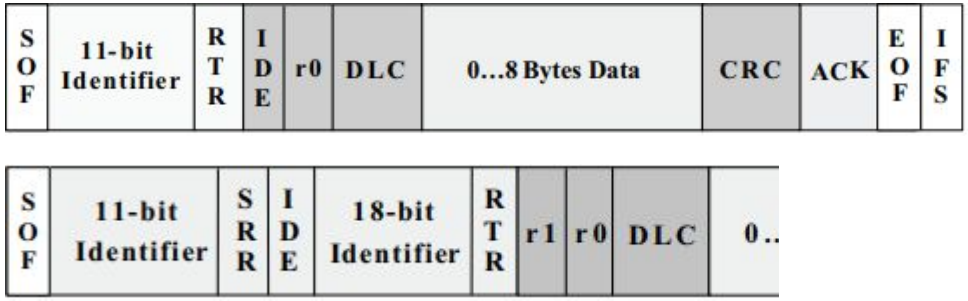


Controller Area Network

CAN Data Frames:

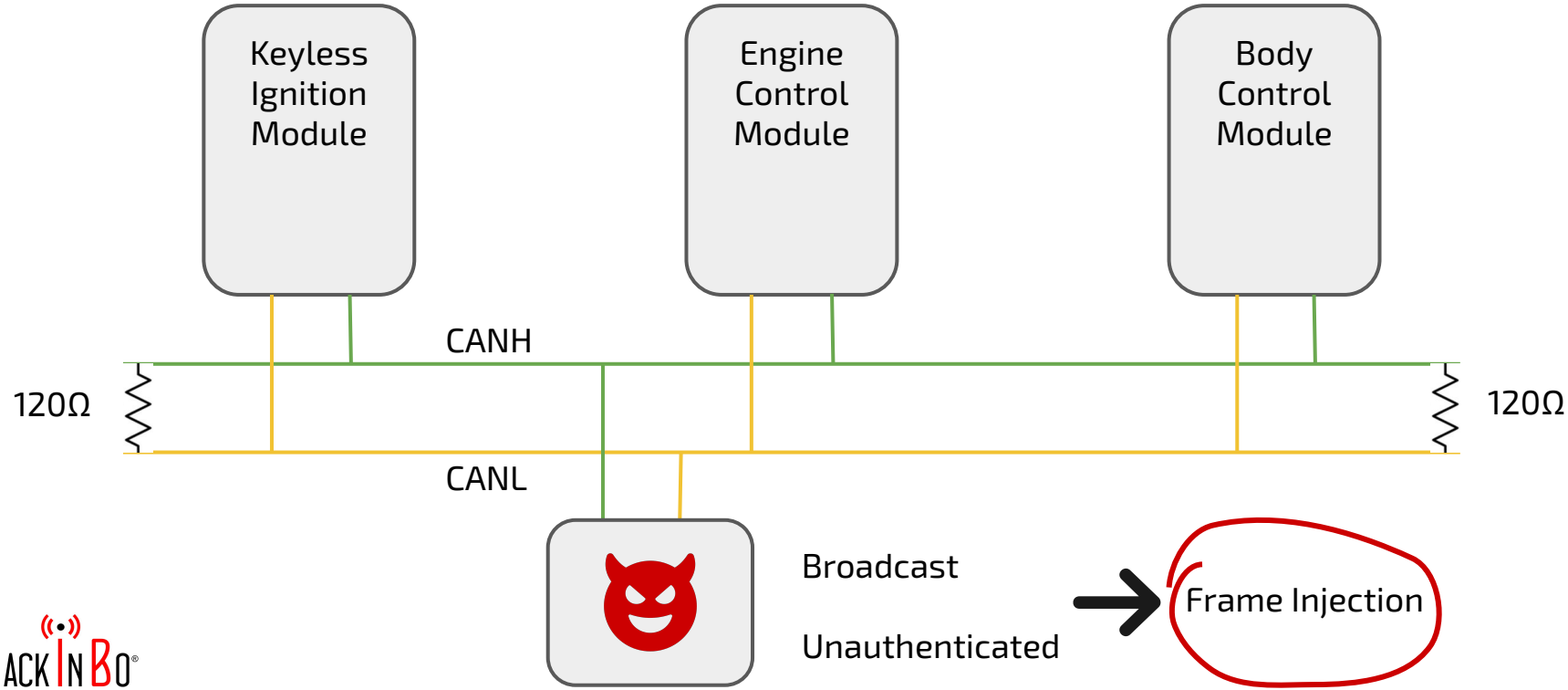
IDs are “owned” by an ECU

Most packets are periodic

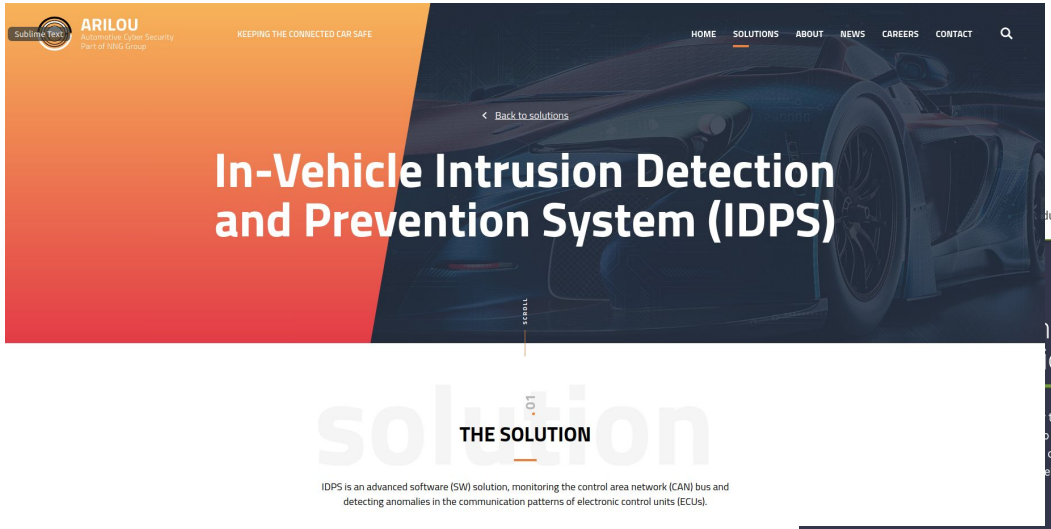


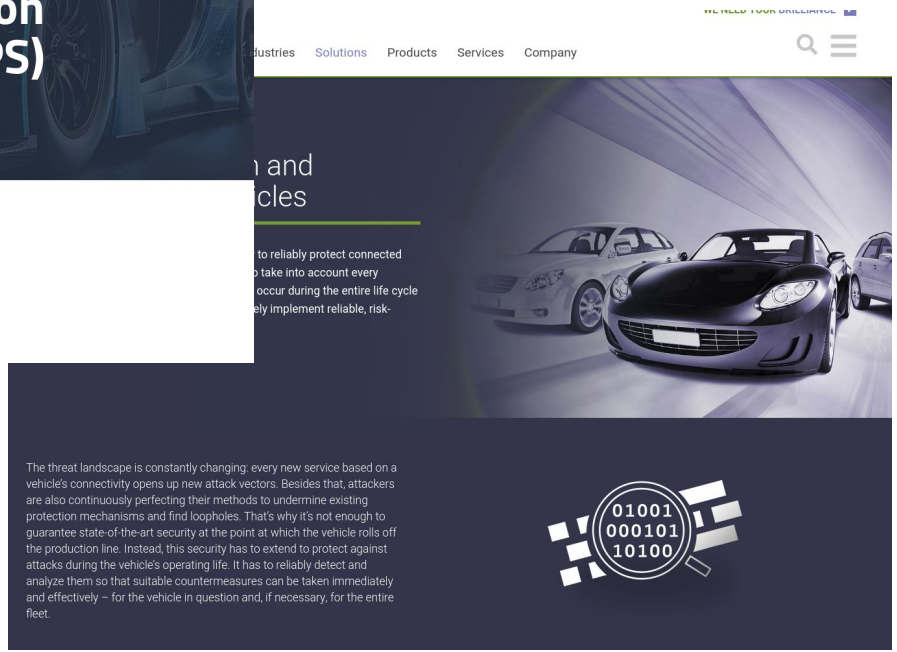

```
31 delta   ID  data ...                 < cansniffer can0 #
0.000000 EXT 0x02214000  00 28 00 00 00 00       .(....
0.000000 EXT 0x02294000  00 00 00 00 02 00 00 00 ........
0.000000 EXT 0x04214001  00 81 90 41 00 00 00 00 ...A....
0.000000 EXT 0x04214006  00 00 00 00 00 00 00 00 ........
0.000000 EXT 0x04294001  01 01 00 00 40 00 00 00 ....@...
0.000000 EXT 0x04394000  00 00 00 3D             ...=
0.000000 EXT 0x06214000  29 04 48 00 00 3A 0B 00 ).H..:..
0.000000 EXT 0x0621401A  00 05 01 00 80 00 00 00 ........
0.000000 EXT 0x06254000  00 00 00 00             ....
0.000000 EXT 0x06314000  40 00 00 00 00 00 00 00 @.......
0.000000 EXT 0x06314003  20 10 70 00 02 08 00 00  .p.....
0.000000 EXT 0x06314005  04 00 00 00 10 00 00 00 ........
0.000000 EXT 0x06314021  00 00 00 00 00 00 00 84 ........
0.000000 EXT 0x06354000  00 00                   ..
0.000000 EXT 0x063D4000  84 4C 00 00             .L..
1.022906 EXT 0x08094021  04 22 48 78 00 00 00 00 ."Hx....
0.000000 EXT 0x08194003  C0                      .
0.000000 EXT 0x08194005  0A 80 00 00 00          .....
0.019534 EXT 0x0A014021  50 00 00 00 00 00 00 02 P.......
e.000000 EXT 0x0A094005  00 58 6C                .Xl
0.000000 EXT 0x0A114000  00 06 1F                ...
0.000000 EXT 0x0A114005  E3 00 00 00 02 00       ......
0.000000 EXT 0x0A194005  00 00 00 00 00 00 80    .......
0.000000 EXT 0x0A394021  55 2A 4D 46 80 00 00 00 U*MF....
0.000000 EXT 0x0C014003  05 51 D2 E9 88 EC 00 00 .Q......
0.000000 EXT 0x0C114003  05 58 3D 01 29 00 39 C0 .X=.).9.
0.000000 EXT 0x0C194003  05 58 38 38 20 05 55 E0 .X88 .U.
0.000000 EXT 0x0C214003  16 56 14 07 20 19       .V.. .
0.000000 EXT 0x0C2D4003  1F 72 82 00 00 00 00 00 .r......
```

What's the idea? (CAN is broken pt.1)



What about countermeasures?



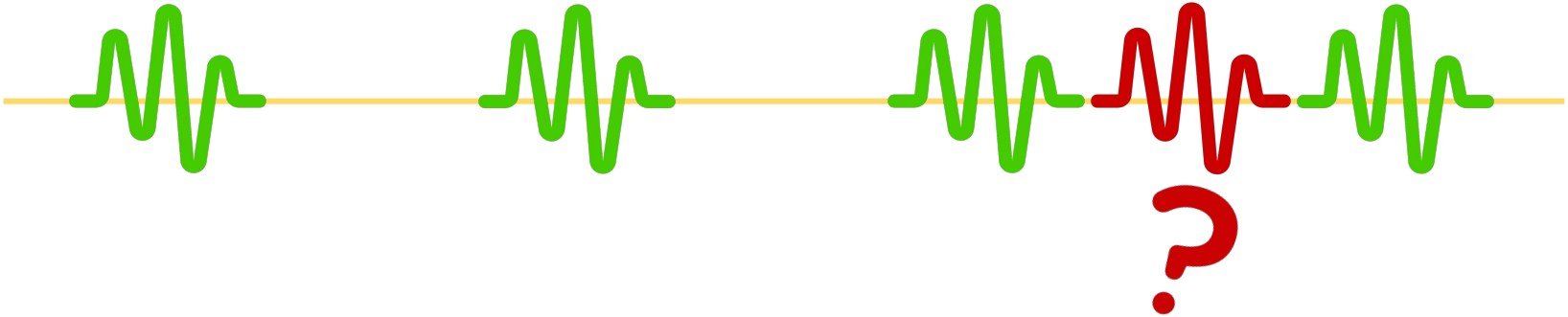


What about countermeasures?

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)
  - Both frequency and specification based
  - Mainly Academic

Now, can we evade them?



- Specification based: Comply with the rules
- Frequency based: Comply with the frequency

Now, can we evade them?

What if we manipulate/substitute a real frame?

- Specification based: Comply with the rules
- Frequency based: Comply with the frequency

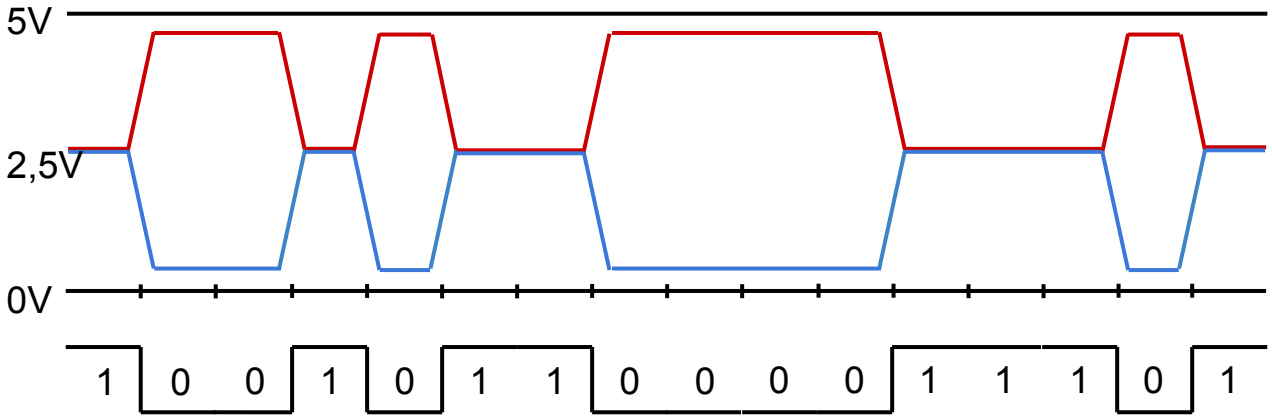
Now, can we evade them?

What if we manipulate/substitute a real frame?

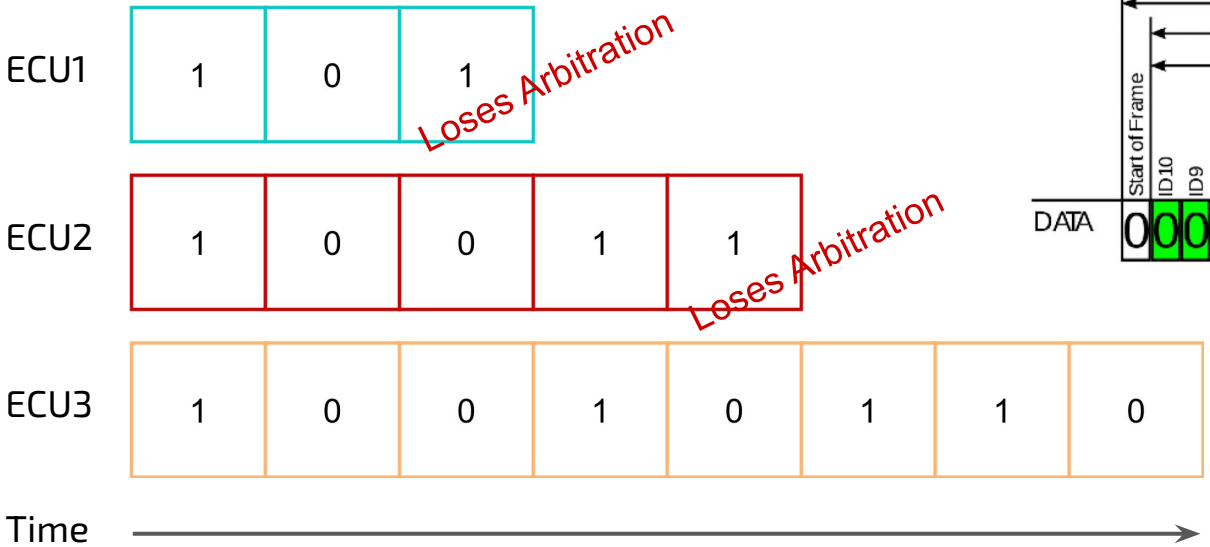
- Specification based: Comply with the rules
- Frequency based: Comply with the frequency

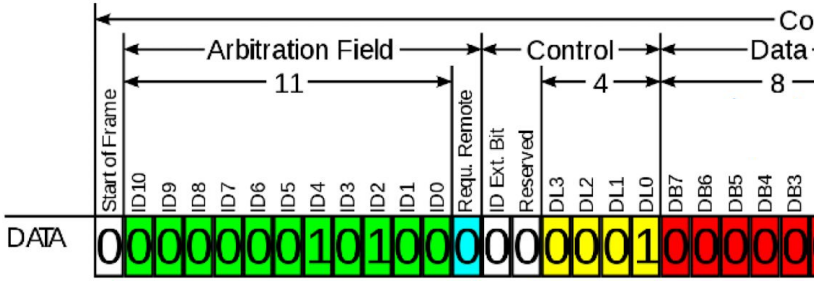
But… how? (CAN is broken pt.2)

What if CAN **embeds** a way to let us do it?



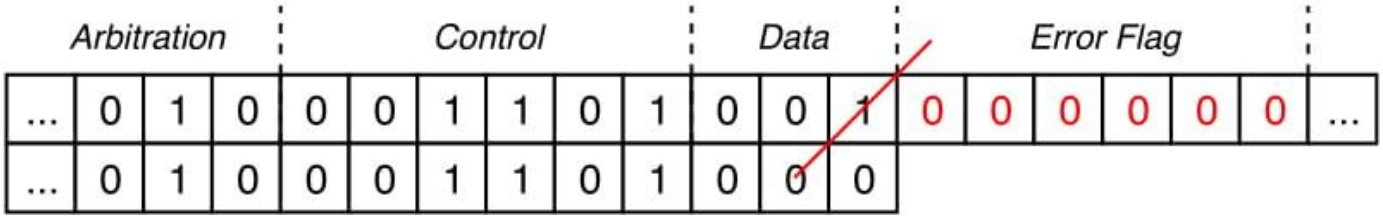
CAN is broken pt.2





CAN is broken pt.2

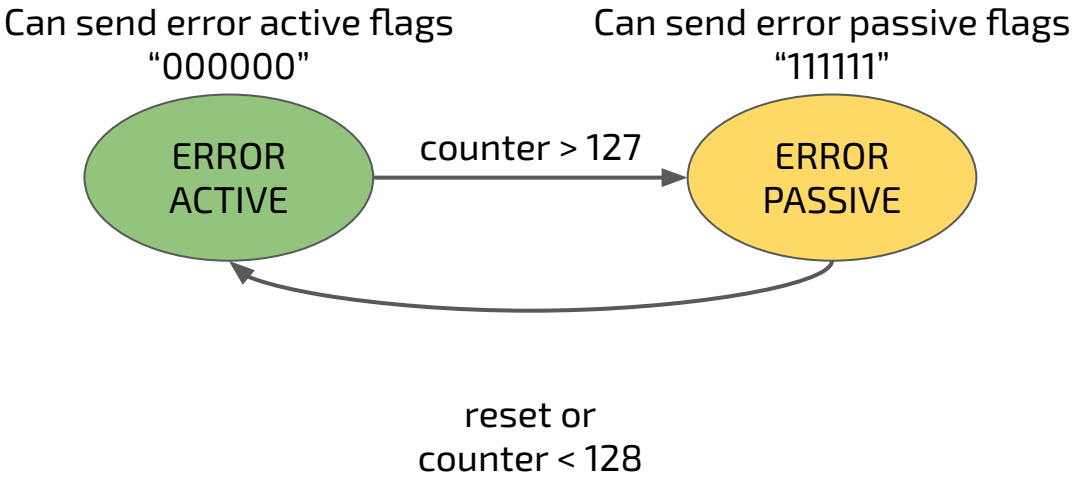
What if CAN **embeds** a way to let us do it?



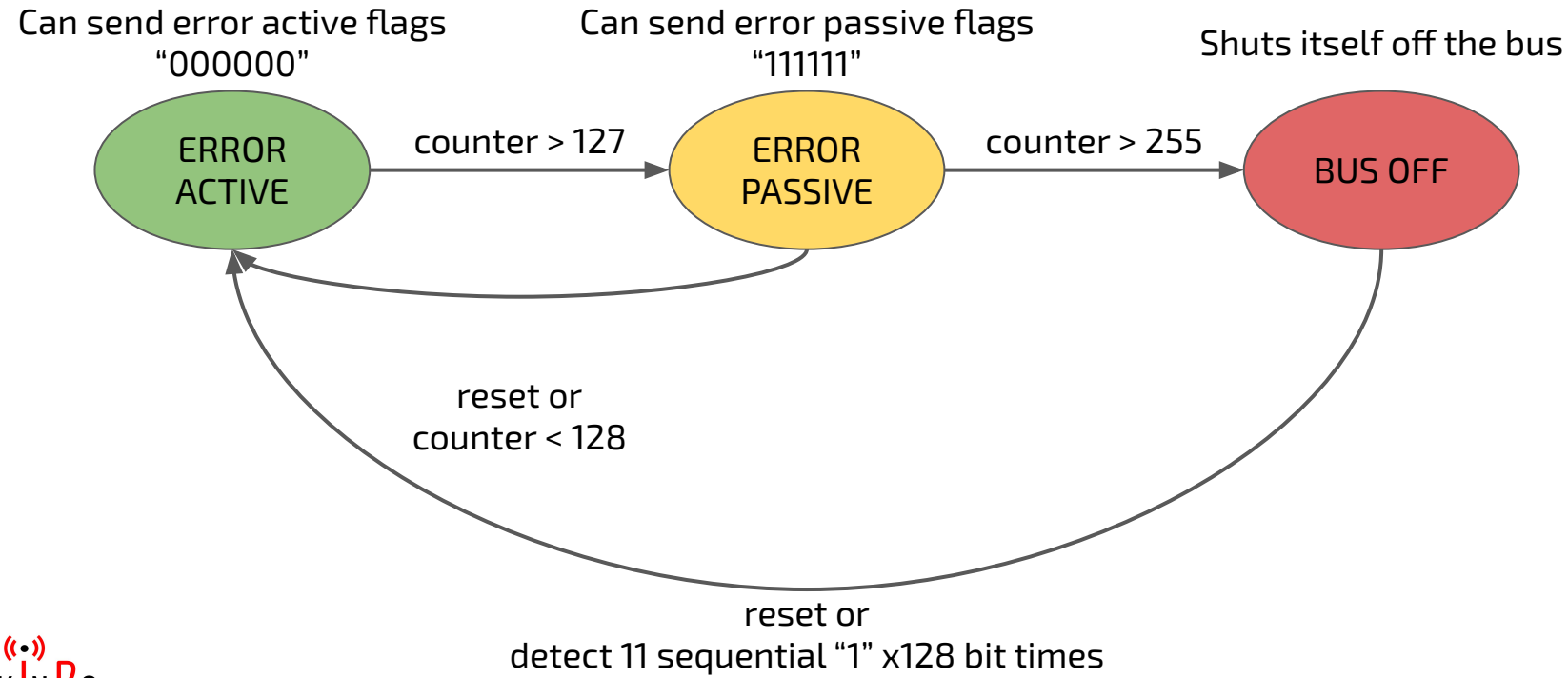
CAN is broken pt.2

Can send error active flags
“000000”

CAN is broken pt.2



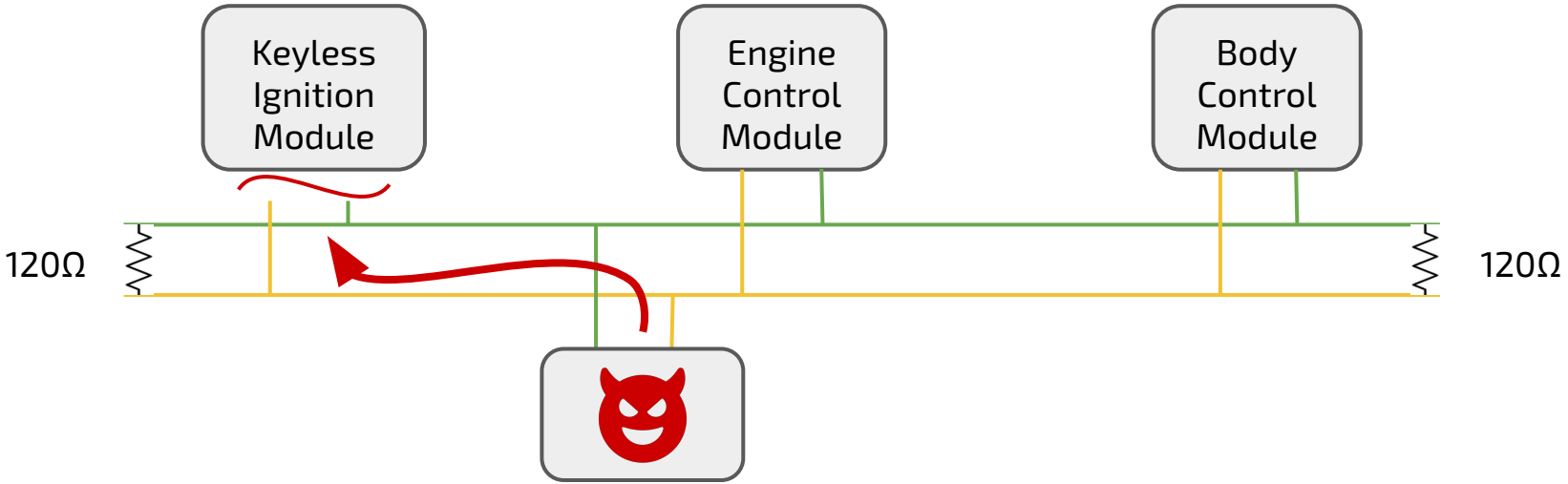
CAN is broken pt.2



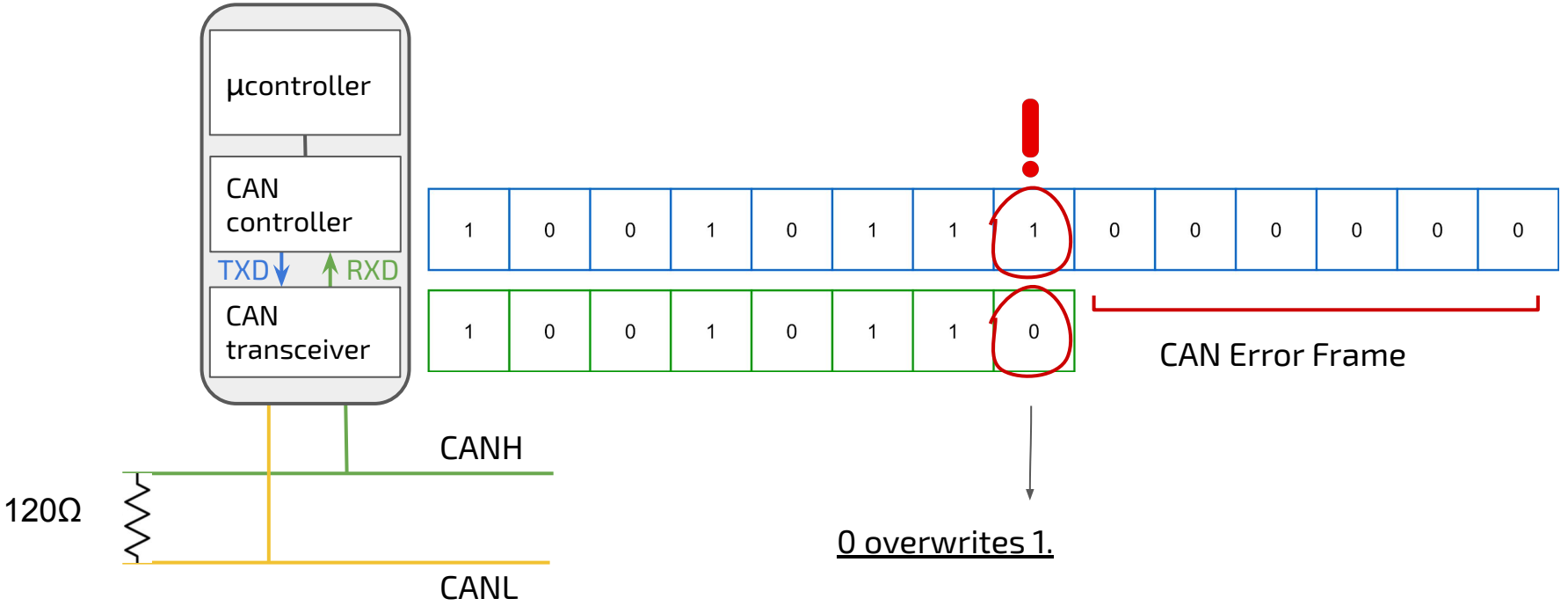
Let's put 2 and 2 together

How do we substitute a frame?

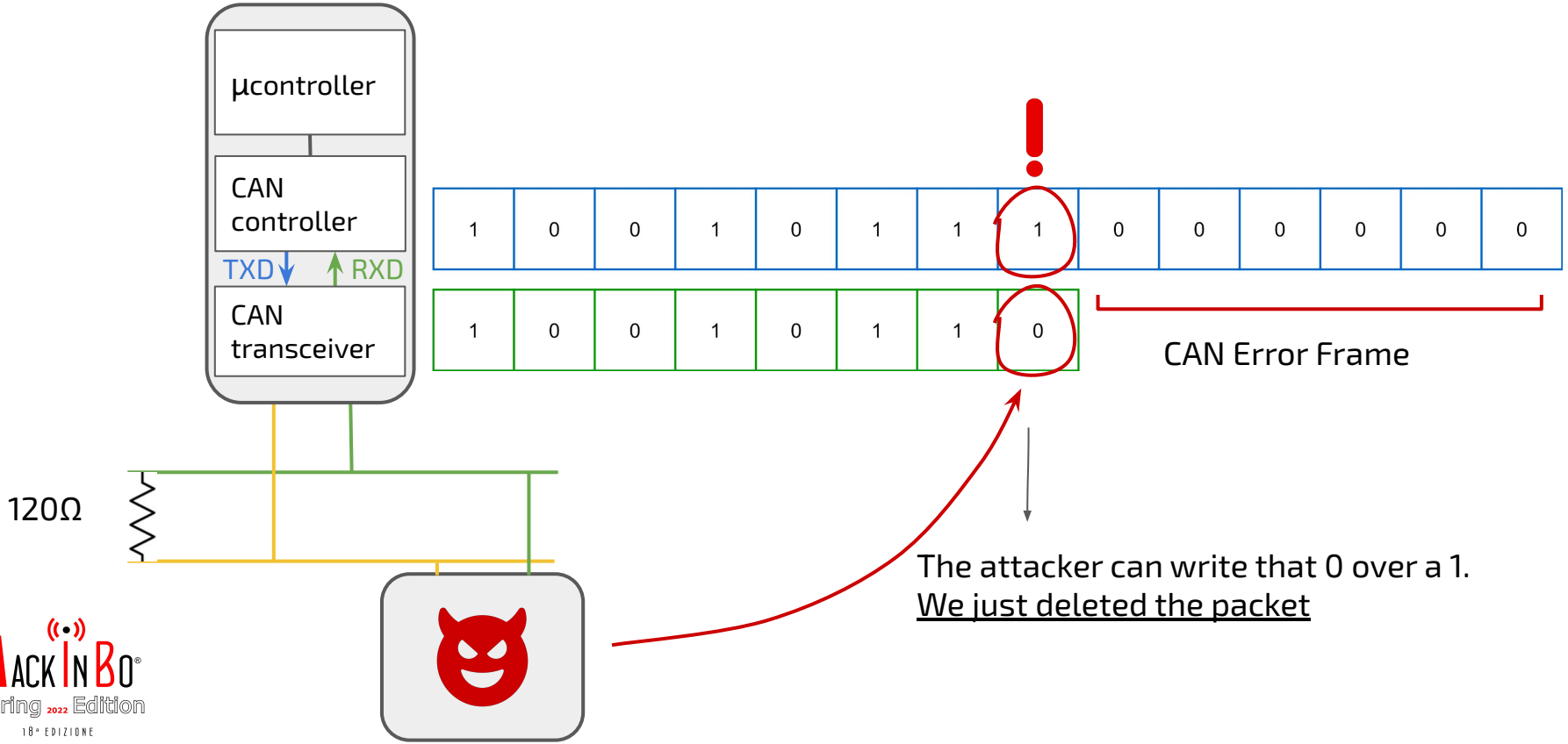
Let's put 2 and 2 together



Let's put 2 and 2 together



Let's put 2 and 2 together



Steps

1) Discover the ID of the victim

e.g., Reverse engineer the CAN IDs of an identical vehicle

2) Detect the ID of the victim on the bus

3) Find a “1” (recessive) bit in the packet

4) Overwrite it with a 0

5) Repeat 32 consecutive times

Steps

1) Discover the ID of the victim

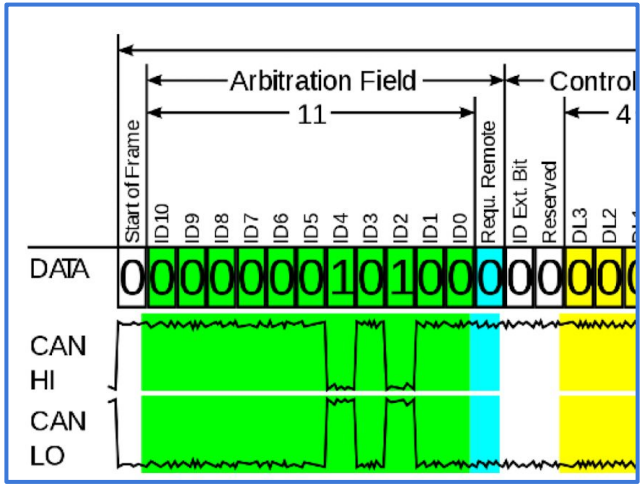
2) Detect the ID of the victim on the bus

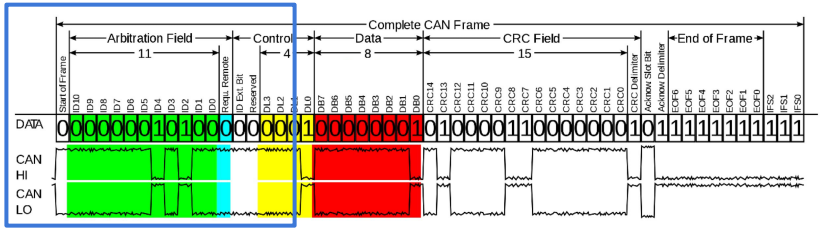
3) Find a “1” (recessive) bit in the packet

4) Overwrite it with a 0

5) Repeat 32 consecutive times

e.g., read all IDs passing on the bus

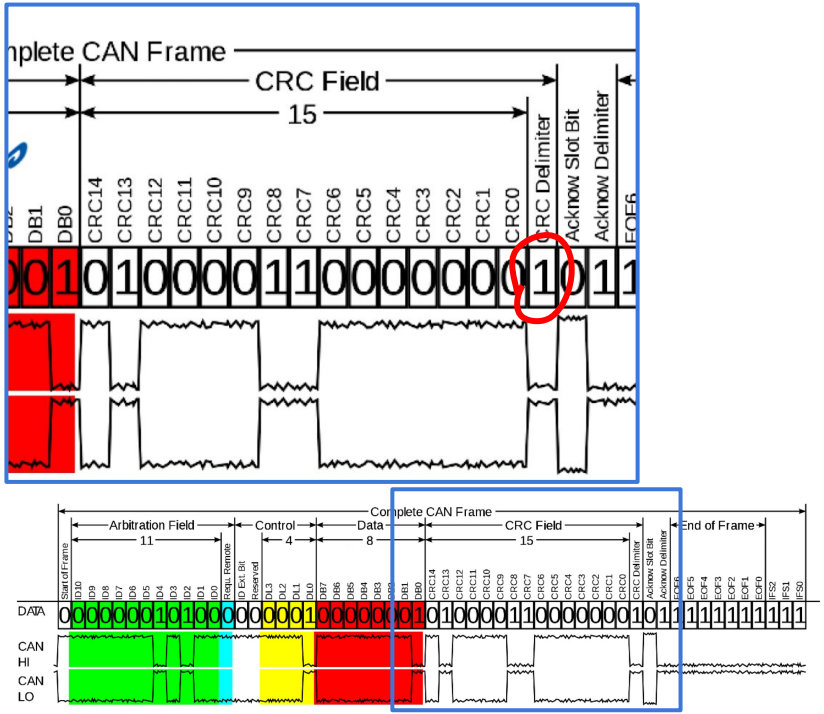




Steps

1) Discover the ID of the victim
2) Detect the ID of the victim on the bus
3) Find a “1” (recessive) bit in the packet
4) Overwrite it with a 0
5) Repeat 32 consecutive times

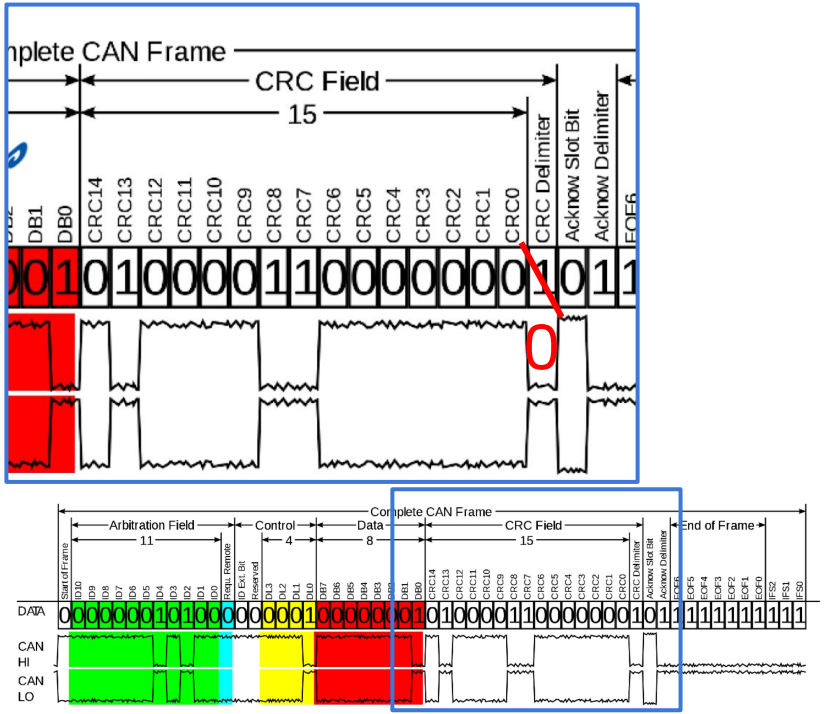
CRC delimiter is “1” by design



Steps

1) Discover the ID of the victim
2) Detect the ID of the victim on the bus
3) Find a “1” (recessive) bit in the packet
4) Overwrite it with a 0
5) Repeat 32 consecutive times

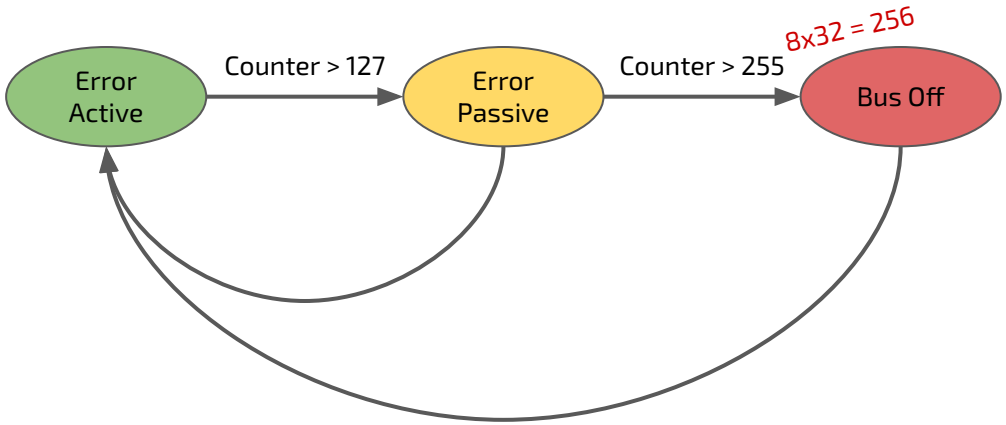
This triggers an error generated by the victim



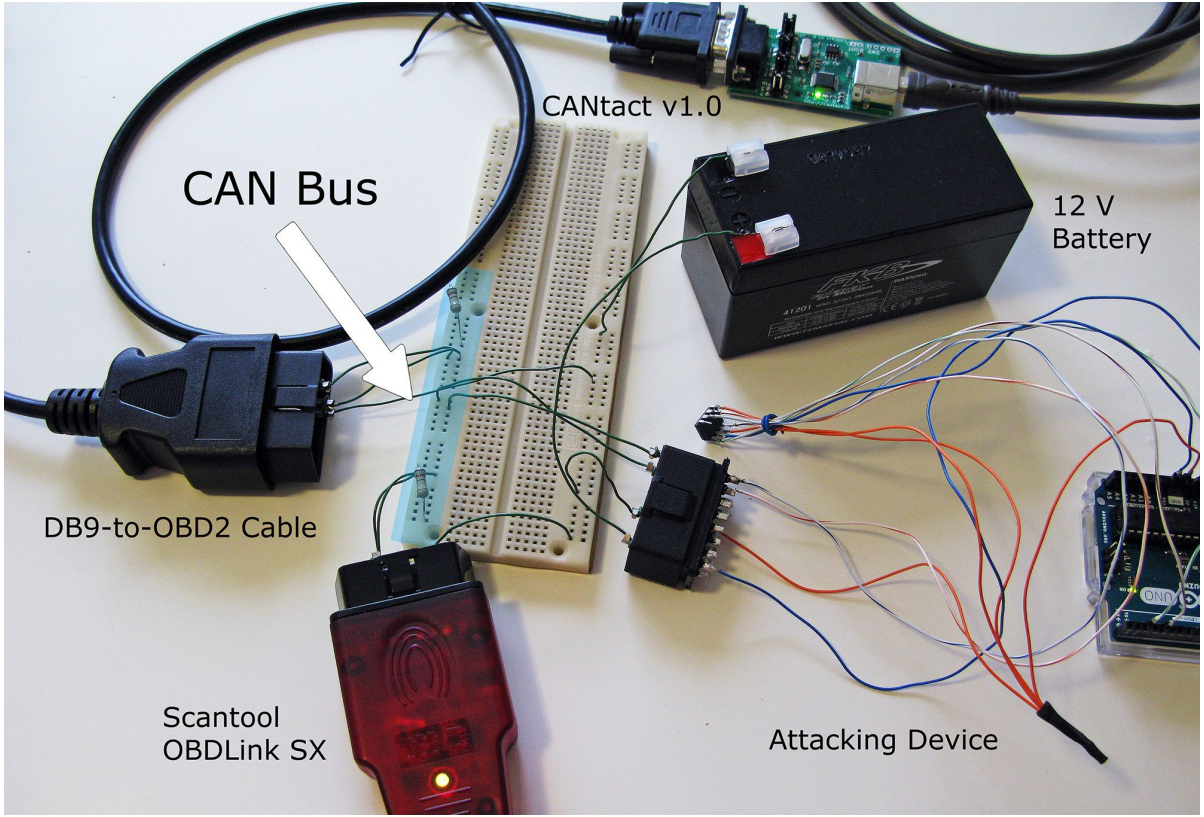
Steps

1) Discover the ID of the victim
2) Detect the ID of the victim on the bus
3) Find a “1” (recessive) bit in the packet
4) Overwrite it with a 0
5) Repeat 32 consecutive times

This kind of error adds +8 to the counter of the victim



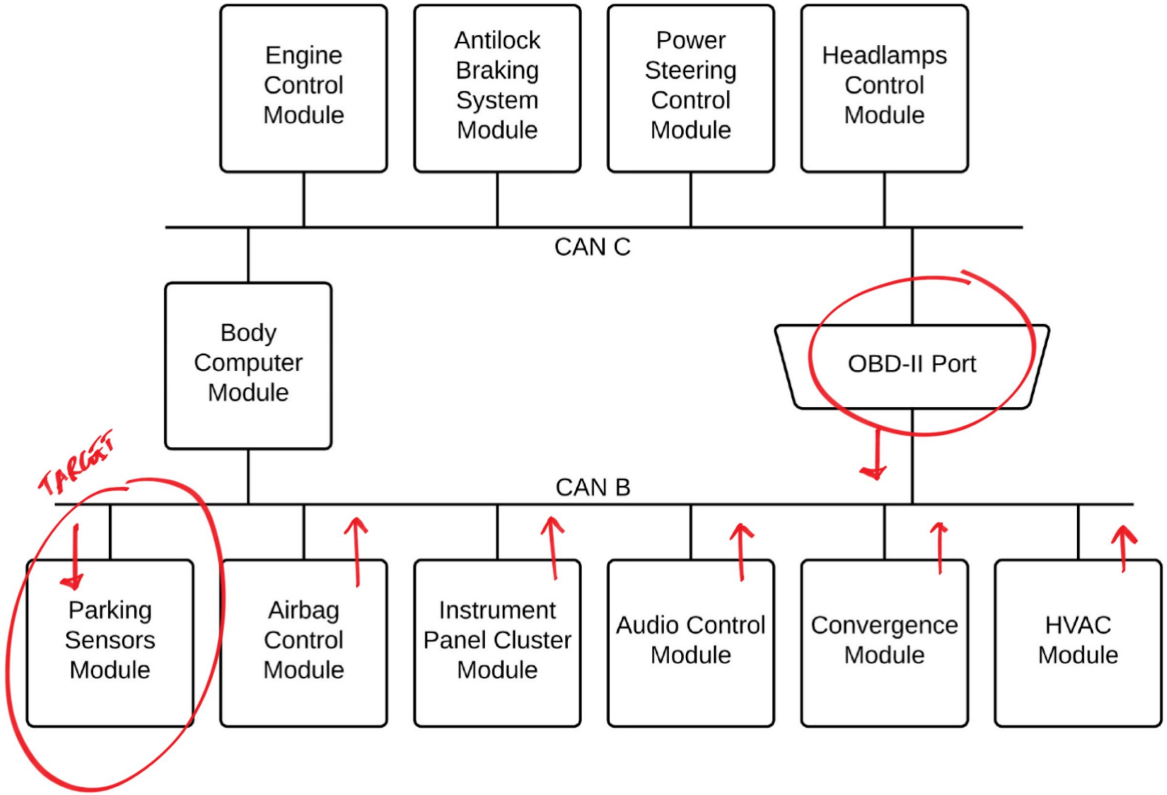
Proof of Concept Implementation



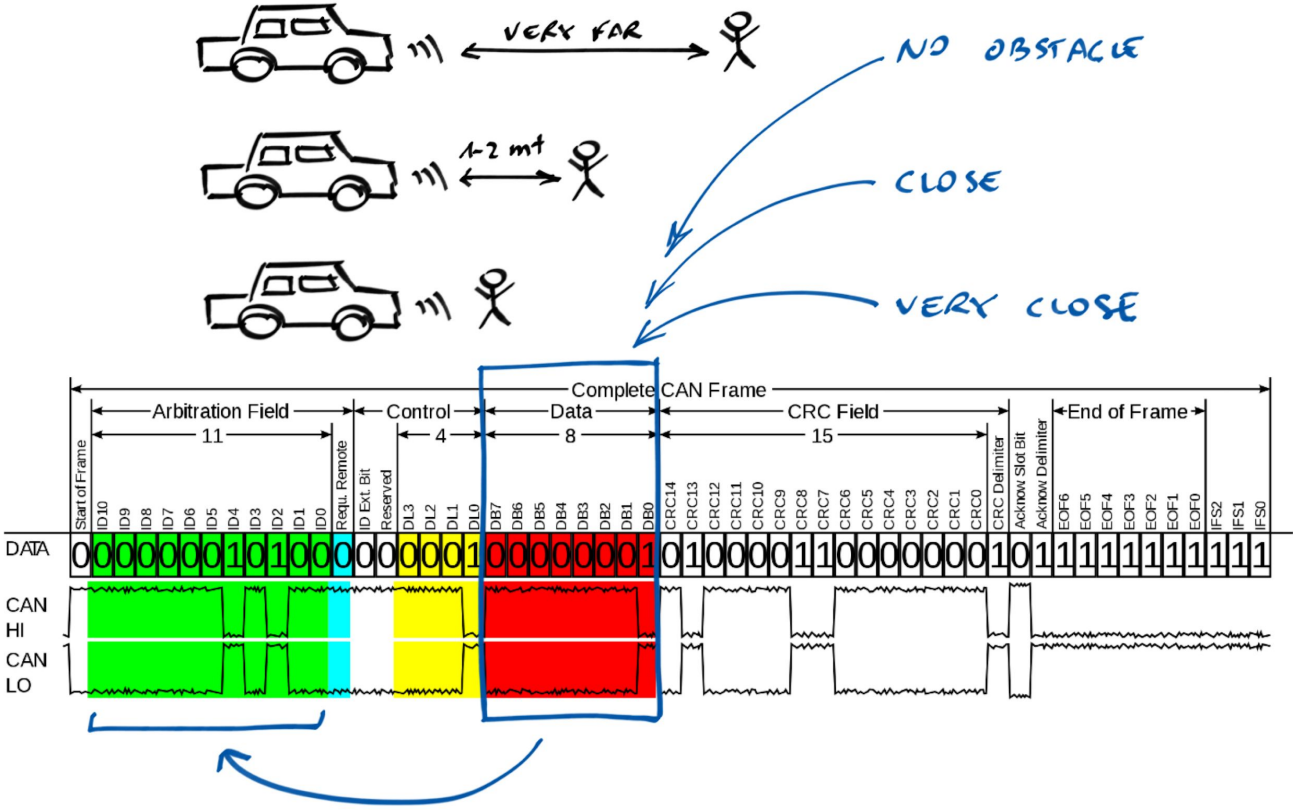
Proof of Concept Implementation

Palanca, A., Evenchick, E., Maggi, F., & Zanero, S. (2017, July). A stealth, selective, link-layer denial-of-service attack against automotive networks. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 185-206). Springer, Cham.

Alfa Giulietta Exploited



Alfa Giulietta Exploited



Alfa Giulietta Exploited

https://is.gd/candos

Can we prevent it?

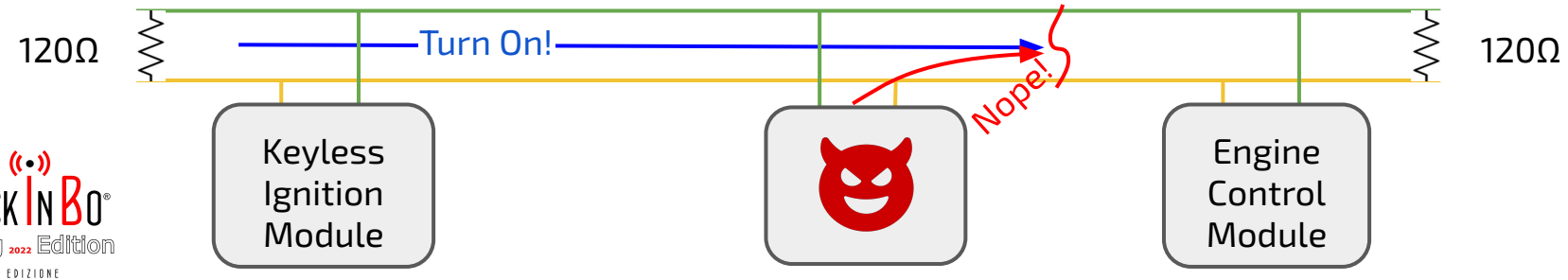
Not really…

- Based on the protocol specs
- Can't really retrieve logs to distinguish between real failures and attacks

Attack scenarios

Denial of Service for the sake of Denial of Service

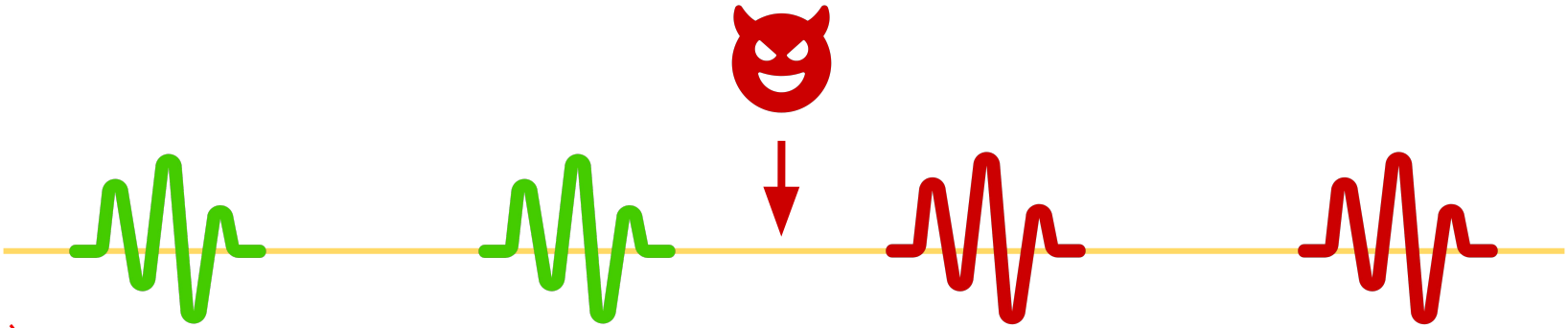
e.g. Ransomware



Attack scenarios

Detection avoidance for spoofing attacks

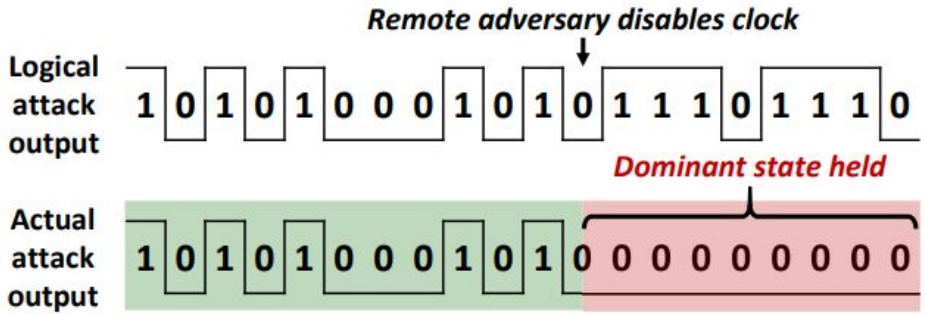
- Shut down the victim ECU
- Send spoofed data

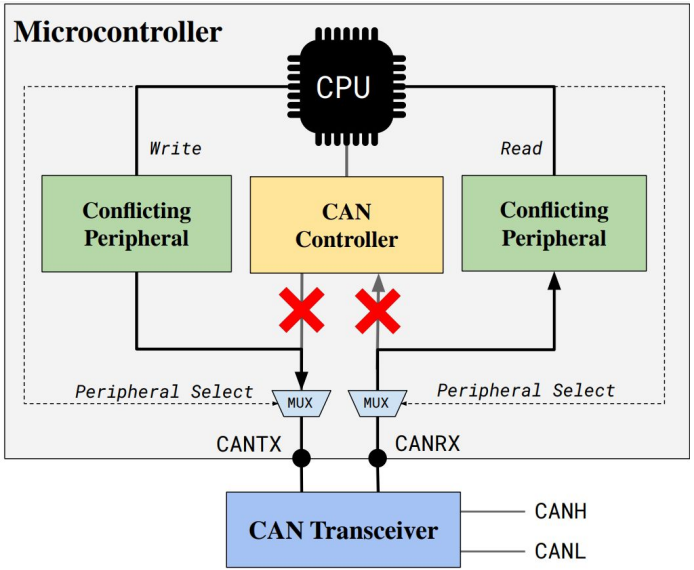


Can we do this from remote?

Once, we thought that no, we require a physical connection

Now we have more than one way to do so…



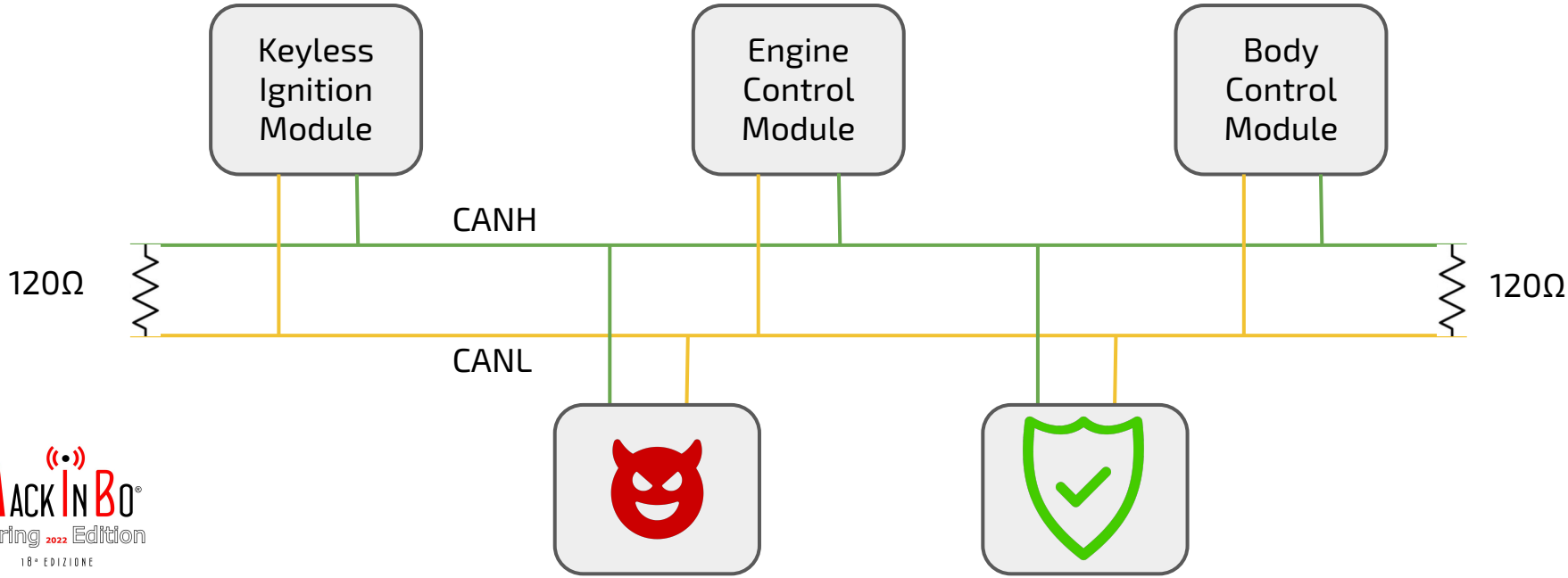


Kulandaivel, Sekar, et al. "Cannon: Reliable and stealthy remote shutdown attacks via unaltered automotive microcontrollers." 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021.

Finally, can we do something about it?

We can read data from the bus

We can detect the attacker once he tries to spoof data after the DoS



Let's see what the protocol says

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 (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.

We only need part of these

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 (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.

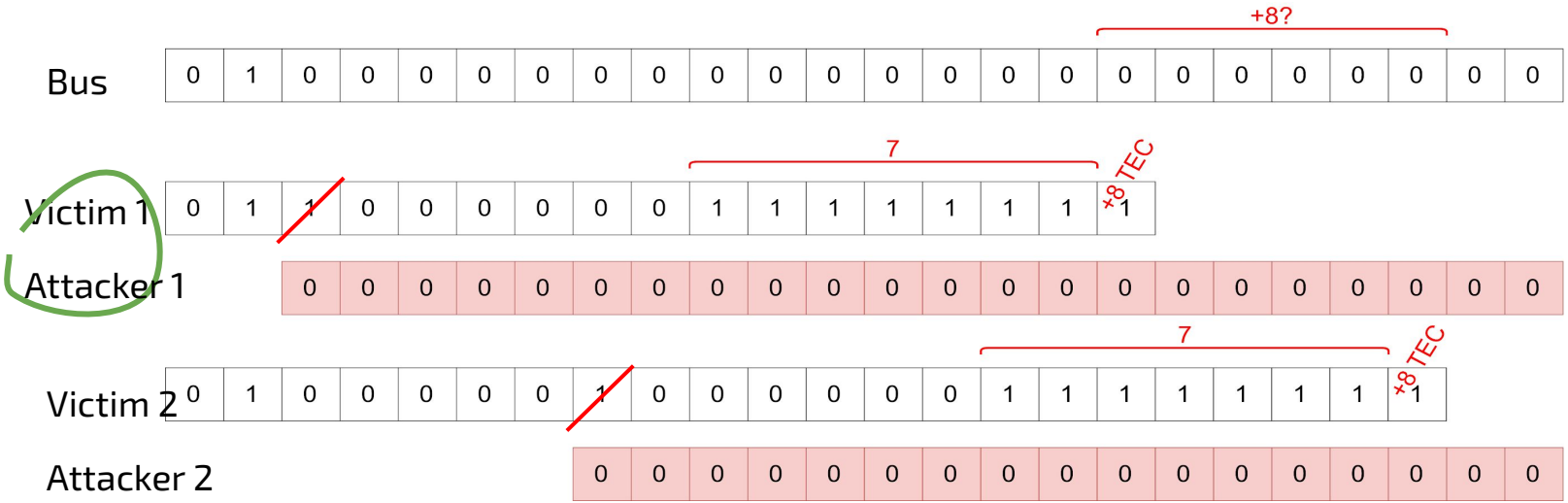
10. A node is 'bus off' when the TRANSMIT ERROR COUNT is greater than or equal to 256.

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.

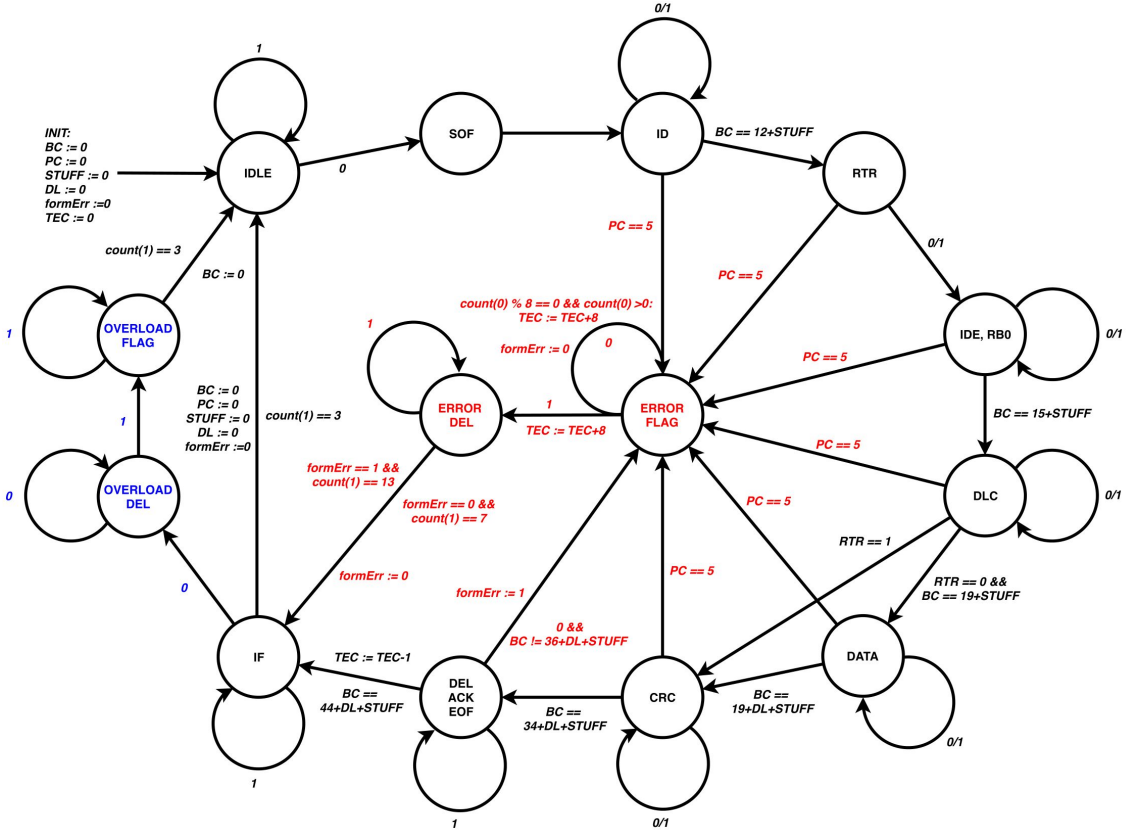
12. An node which is 'bus off' is permitted to become 'error active' (no longer 'bus off') with its error counters both set to 0 after 128 occurrence of 11 consecutive 'recessive' bits have been monitored on the bus.

Rule 6

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



Our new CAN Controller

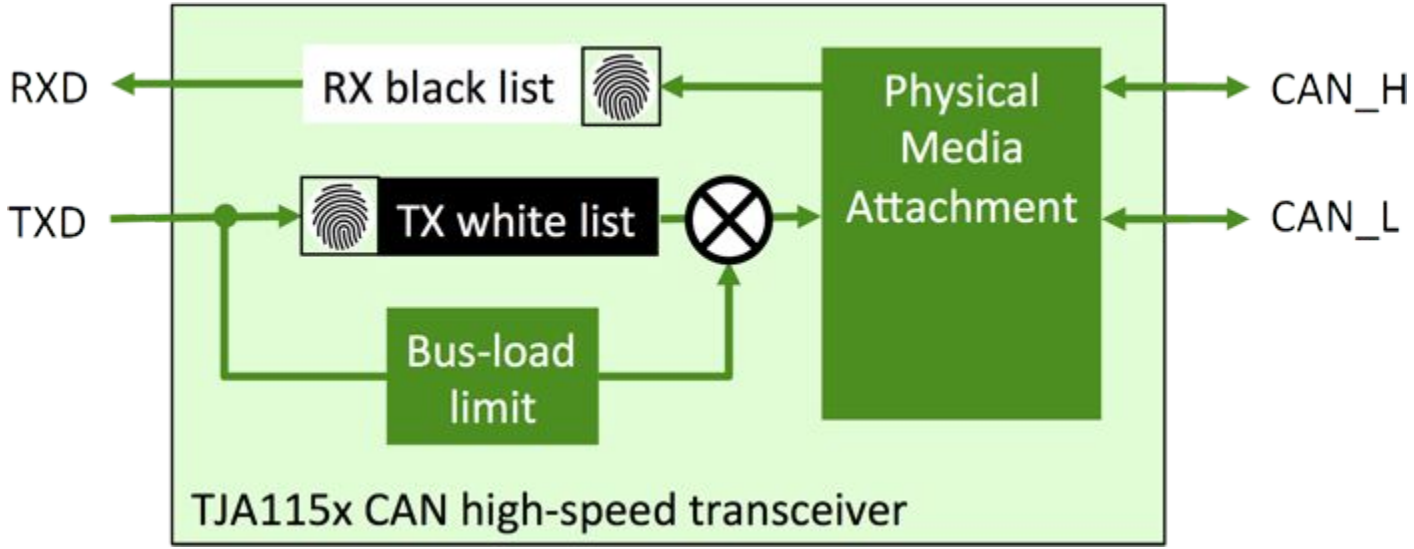


The whole process

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.
6) React?

Other solutions?

NXP's transceiver



Other solutions?

Not for CAN, not for DoS.

Thanks!

For any questions:

< stefano.longari@polimi.it >

@ascarecrowhat