

# Intro to Car Hacking: Whitehat Car-Napping or is it?

...



Jay Turla @shipcod3

**ROOTEON**

# whoami

- @shipcod3
- Security Ops Manager (Philippines) at Bugcrowd
- ROOTCON Goon / CFP Review Board
- That pinoy on virtual cons w/ roosters in the background
- One of the main organizers of the Car Hacking Village in ROOTCON and PH → #CarHackVillagePH (<https://www.carhackingvillage.com/about>)
- msf contributor (auxiliary & exploit modules)
- Speaker at local & international conferences: ROOTCON, HITCON, DEFCON 26 Packet Hacking Village, OWASP Seaside, DragonCon, SarCon, Bsides Myanmar, Bugcrowd LevelUp, Nullcon, PEHCON and TCON



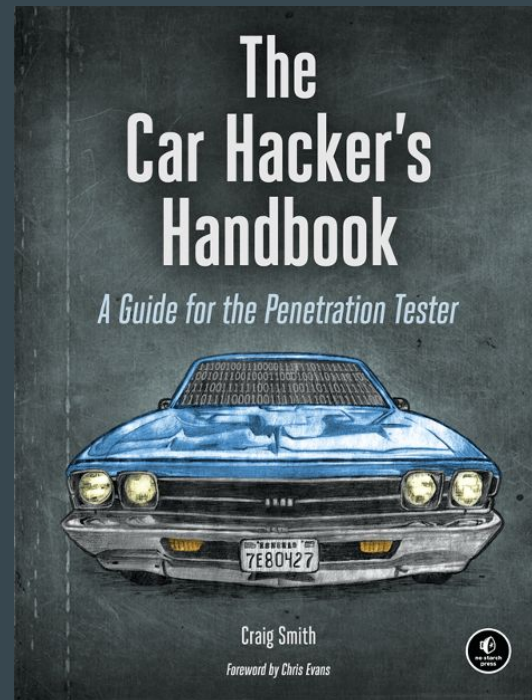
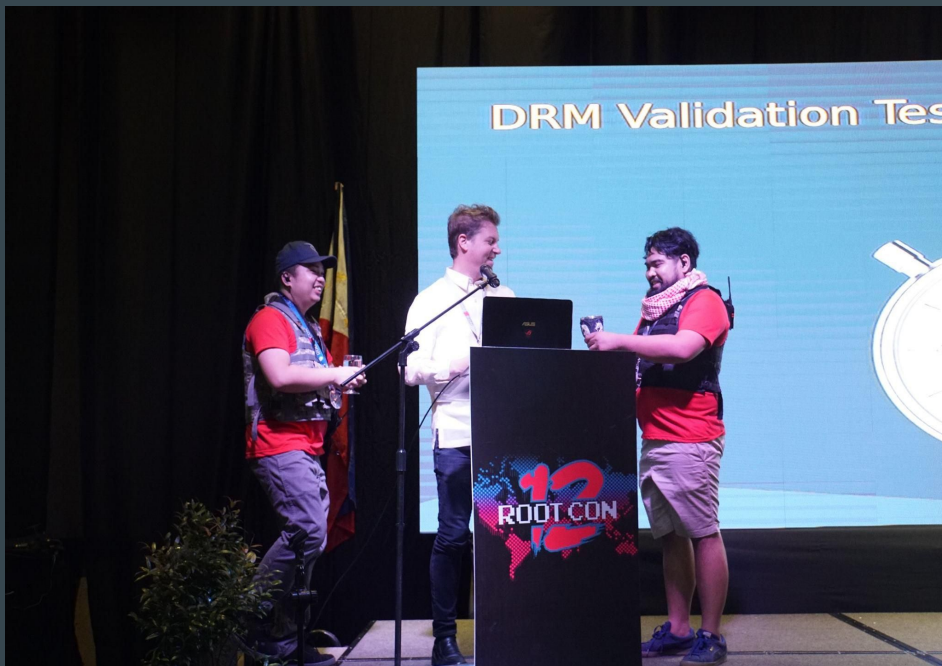
# Greetz to Arun Mane and Nikhil Bogam for the Anomaly Detection System



Taken during ROOTCON 13 at the 1st Car Hacking Village PH

# Best Car Hacking Book Ever

- Online version: <http://opengarages.org/handbook/ebook/>
- Written by Craig Smith



# Is there money in Car Hacking?



**BUG BASH**  
OCTOBER 12TH 2019

bugcrowd

Rank  
**1** Private user  
Points 120

Rank  
**2** irotem  
Points 114

Rank  
**3** iangcarroll  
Points 105

Rank	Researcher	Points
4	peri	95
5	uDISAn	76
6	fronders	60
7	anvo1	60
8	Specters	44
9	BusesCanFly	43
10	Lennert	43

Bug bash is closed

Submissions are still being validated.

0 : 00 : 00 : 00  
days hours minutes seconds

Bounties paid

**\$224000**

# Sample Attacks Carried out on Attack Surfaces

- CAN Injection attacks on the Controller  
Area Network: OBD-2, sensors, unprotected components, etc
- Radio Attacks (keyfob replay attacks and rolljam)

```
jjt@ubuntu:~/pentot/hackrf-tests$ cat hackrf_listen.sh  
hackrf_transfer -r 390_data.raw -f 39000000000 #listen  
jjt@ubuntu:~/pentot/hackrf-tests$ cat hackrf_transmit.sh  
hackrf_transfer -t 390_data.raw -f 39000000000 #transmit  
jjt@ubuntu:~/pentot/hackrf-tests$ █
```

- Hardware attacks (requires physical access)
- IoT Hacking on infotainment systems, fms, telematics
- Web and mobile attacks on certain endpoints
- DoS (denial of service attacks) and Memory Corruption: Bluetooth, WiFi, Services



# Unauthorized CAN Access



Reference:

<https://www.blackhat.com/docs/eu-16/materials/eu-16-Sintsov-Pen-Testing-Vehicles-With-Cantoolz.pdf>

# Did You Know? Airplanes and Broom Broom Have CAN Too

To reduce the number of interconnecting wires from control panels in the flight deck to system computers in the avionics compartment, Airbus deployed CAN bus. - **A380**

## Application





# Automotive Communication Bus Overview

Bus	LIN	CAN	FlexRay
Speed	40 kbit/s	1 Mbit/s	10 Mbit/s
Cost	\$	\$\$	\$\$\$
Wires	1	2	2 or 4
Typical Applications	Body Electronics (Mirrors, Power Seats, Accessories)	Powertrain (Engine, Transmission, ABS)	High-Performance Powertrain, Safety (Drive-by-wire, active suspension, adaptive cruise control)

Reference:

<https://www.ni.com/en-us/innovations/white-papers/06/flexray-automotive-communication-bus-overview.html>

# My Favorite <3 (My tools)



nano-can

CANtact

STM32 Can Sniffer  
by TechMaker

ValueCAN 4

# CarHacking.Tools by jgamblin

- collection of scripts to help jump start car research and hacking
- All the scripts are designed to run on Ubuntu
- Install via Virtual Machine:

<https://carhacking.tools/install/beta/CarHackingToolsCHVBeta.ova>

- Or can be installed via the repo:

```
git clone https://github.com/jgamblin/carhackingtools  
cd CarHackingTools  
sudo chmod +x *.sh  
./toolinstall.sh
```



# Try to learn how to cansend, cangen, candump, etc

```
jjt@ubuntu:~$ cansend
Usage: cansend - simple command line tool to send CAN-frames via CAN_RAW sockets.
Usage: cansend <device> <can_frame>.
<can_frame>:
  <can_id>#{R|data}           for CAN 2.0 frames
  <can_id>##<flags>{data}     for CAN FD frames

<can_id>:
  can have 3 (SFF) or 8 (EFF) hex chars
{data}:
  has 0..8 (0..64 CAN FD) ASCII hex-values (optionally separated by '.')
<flags>:
  a single ASCII Hex value (0 .. F) which defines canfd_frame.flags

Examples:
  5A1#11.2233.44556677.88 / 123#DEADBEEF / 5AA# / 123##1 / 213##311
  1F334455#1122334455667788 / 123#R for remote transmission request.

jjt@ubuntu:~$ █
```

# Just a good tool you want to try

```
jjt@ubuntu:~/pentot/caringcaribou/tool$ python3 cc.py uds discovery
```

```
-----  
CARING CARIBOU v0.3  
-----
```

```
Loaded module 'uds'
```

```
Sending Diagnostic Session Control to 0x0710  
  Verifying potential response from 0x0710  
    Resending 0x710... Success  
Found diagnostics server listening at 0x0710, response at 0x077a  
Sending Diagnostic Session Control to 0x07df  
  Verifying potential response from 0x07df  
    Resending 0x7df... Success  
Found diagnostics server listening at 0x07df, response at 0x077a  
Sending Diagnostic Session Control to 0x07e0  
  Verifying potential response from 0x07e0  
    Resending 0x7e0... Success  
Found diagnostics server listening at 0x07e0, response at 0x077a  
Sending Diagnostic Session Control to 0x07ff
```

```
Identified diagnostics:
```

```
+-----+-----+  
| CLIENT ID | SERVER ID |  
+-----+-----+  
| 0x00000710 | 0x0000077a |  
| 0x000007df | 0x0000077a |  
| 0x000007e0 | 0x0000077a |  
+-----+-----+
```

# Scripting and Automation (Warning the script below can cause DoS)

```
#!/bin/bash

while :
do
    cansend vcan0 000#000000000000000000000000
done
```

# Scripting and Automation: Meet pyvit

```
from pyvit.hw.socketcan import SocketCanDev
from pyvit import can

dev = SocketCanDev('vcan0')
dev.start()

frame = can.Frame(0x7F1)
frame.data = [0x27, 0x5F]
dev.send(frame)

frame.data = [0x27, 0x60, 0xFE, 0x1C]
dev.send(frame)
```

If you spot me in a conference: if interested, ask me one





# Upload code using Arduino IDE

- Sample CAN Sniffer:

<https://github.com/mintynet/nano-can/tree/master/can-receive-all> (CAN Receive All)

- My other sketches:

<https://github.com/ROOTCONLabs/tools/tree/master/car-hacking/sketches>

```
mazda_rpm_fuzzer
// #####
// # Mazda 2 Instrument Cluster that allows you to go vroom vroom #
// # RPM PoC Fuzzer for n00bz #
// # author: @shipcod3 #
// # ROOTCON Car Hacking Village #
// # greetz to sempix, mogul, mintynet and eman0n for the support #
// #####
// special love to @carhackingvillage and carfucar

#include <mcp_can.h>
#include <SPI.h>

#define FUNCTIONAL_ID 0x202 /* RPM ID*/

// CAN TX Variables
unsigned long prevTx = 0;
unsigned int invlTx = 1000;

// CAN RX Variables
unsigned long rxID;
byte dlc;
byte rxBuf[8];

// CAN Interrupt and Chip Select Pins
#define CAN0_INT 2 /* Set INT to pin 2 (This rarely changes) */
MCP_CAN CAN0(CAN0_INT); /* Set CS to pin 10 (Old shields use pin 10) */
byte counter = 0;

void setup(){

  Serial.begin(115200);
  while(!Serial);

  // Initialize MCP2515 running at 8MHz with a baudrate of 125kb/s and the masks and filters disabled.
  if(CAN0.begin(MCP_STDEXT, CAN_500KBPS, MCP_8MHZ) == CAN_OK)
    Serial.println("\nMCP2515 Initialized Successfully!");
  else{
    Serial.println("Error Initializing MCP2515... Permanent failure! Check your code & connections");
    while(1);
  }

  CAN0.setMode(MCP_NORMAL); // Set operation mode to normal so the MCP2515 sends acks to received data.
```

# Common Enterprise Tools on the Block: Vehicle Spy and CANoe

The screenshot shows the 'New Spy Setup - Vehicle Spy 3' window. The main area displays a table of CAN bus data with columns for Count, Time, Tx, Er, Description, ArbitHeader, Len, DataBytes, Network, and Node. The 'ActualEngine\_PerTorque' signal is highlighted in green. Below the table is a 'Signal Plot' section showing a graph of 'ActualEngine\_PerTorque' over time, with a Y-axis ranging from 0 to 200 and an X-axis from 18.0 to 36.0. The plot shows a fluctuating signal between approximately 10 and 15.

Count	Time	Tx	Er	Description	ArbitHeader	Len	DataBytes	Network	Node
999	999.000 ms			AmbientConditions	x18FEF500	8	C3 FF FF FF FF FF FF FF	HS CAN	
102	102.000 ms			AuxDiscreteIOState	x18FED9EF	8	0F FF FF FF FF FF FF FF	HS CAN	
102	102.000 ms			Brakes	x18FEFA47	8	FF FF FF FC FF FF FF FF	HS CAN	
999	999.000 ms			CM1	x18E00000	8	00 FF FF FF 3F FF FF FF	HS CAN	
36	36.000 ms			CruiseControlVehSpeed	x18FEF100	8	FF 50 00 50 00 00 00 C0	HS CAN	
60	60.000 ms			DM1	x18FECA40	8	72 02 02 01 00 02 00 01	HS CAN	
99	99.000 ms			DashDisplay	x18FEFC47	8	FF C8 FF FF FF FF FF FF	HS CAN	
498	498.000 ms			EAC1	x18F0062F	8	FF CC FC FC FF FF FF FF	HS CAN	
99	99.000 ms			EBC1	x18F00100	8	FF FF FF CF 00 FF FF FF	HS CAN	
18	18.000 ms			EEC1	xCF00400	8	F0 7F 7F 7F 7F 00 FF 7F	HS CAN	

The screenshot shows the 'Vector CANoe - OfflineAnalysis.cfg' window. The main area displays a trace of CAN bus data with columns for Time, Chn, ID, Name, Event Type, Dir, DLC, and Data. The trace shows a series of CAN frames with IDs ranging from 268 to 400. The 'Data' column shows hexadecimal values for each frame. The 'Trace' section at the bottom shows the current frame being analyzed.

Time	Chn	ID	Name	Event Type	Dir	DLC	Data
13:46:2000	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:46:2996	CAN1	704	CAN Frame	Tx	8	0F	5F 00 04 10 01 04 00 00
13:47:2000	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:47:2996	CAN1	704	CAN Frame	Tx	8	00	5E 00 03 10 01 04 00 00
13:48:1000	CAN1	704	CAN Frame	Tx	8	01	01 03 03 10 01 04 00 00
13:48:2000	CAN1	268	CAN Frame	Tx	8	00	50 00 02 10 01 04 00 00
13:48:2992	CAN1	704	CAN Frame	Tx	8	00	50 00 02 10 01 04 00 00
13:48:3000	CAN1	268	CAN Frame	Tx	8	00	50 00 02 10 01 04 00 00
13:48:3996	CAN1	704	CAN Frame	Tx	8	00	50 00 01 10 01 04 00 00
13:49:2000	CAN1	268	CAN Frame	Tx	8	00	50 00 01 10 01 04 00 00
13:49:2996	CAN1	704	CAN Frame	Tx	8	00	50 00 01 10 01 04 00 00
13:50:0726	CAN1	210	CAN Frame	Tx	2	00	00 00
13:50:0996	CAN1	704	CAN Frame	Tx	8	01	01 03 01 10 01 04 00 00
13:50:1726	CAN1	200	CAN Frame	Tx	1	00	00 00
13:50:2204	CAN1	266	CAN Frame	Tx	8	00	10 11 12 03 14 00 00 00
13:50:2704	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:50:3204	CAN1	400	CAN Frame	Tx	8	00	00 00 00 FF FF FF FF FF
13:50:3700	CAN1	704	CAN Frame	Tx	8	00	50 00 0A 10 01 04 00 00
13:50:4200	CAN1	401	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:51:2000	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:51:2996	CAN1	704	CAN Frame	Tx	8	00	64 00 09 10 01 04 00 00
13:52:0996	CAN1	704	CAN Frame	Tx	8	01	01 03 09 10 01 04 00 00
13:52:2000	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:52:2992	CAN1	704	CAN Frame	Tx	8	00	63 00 08 10 01 04 00 00
13:53:2000	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:53:2996	CAN1	704	CAN Frame	Tx	8	00	62 00 07 10 01 04 00 00
13:54:0555	CAN1	703	CAN Frame	Tx	8	00	00 CD CD CD CD CD CD CD
13:54:1051	CAN1	704	CAN Frame	Tx	8	01	01 03 07 10 01 04 00 00
13:54:1539	CAN1	704	CAN Frame	Tx	8	FF	01 07 01 07 00 00 00 00
13:54:2039	CAN1	268	CAN Frame	Tx	8	00	00 00 00 00 00 00 00 00
13:54:2495	CAN1	703	CAN Frame	Tx	8	FE	CD CD CD CD CD CD CD
13:54:2983	CAN1	704	CAN Frame	Tx	8	FF	01 07 01 07 00 00 00 00

# DEMO Time



# References and Credits

- Car Hacker's Handbook
- <https://github.com/mintynet/nano-can> (nano-can)
- Getting Started with Car Hacking: <https://www.carhackingvillage.com/getting-started>
- CAN Bus Basics With Hands On Fuzzing (Ian Tabor): <https://www.youtube.com/watch?v=6mxQFCHwpRI>
- Shoutz: Car Hacking Village, Craig Smith, Ian Tabor, carfucar, fronders, techmakerua, ASRG, ASRG-SIN, specters, Jami, and semprix

