

Automated Cross-Platform Reverse Engineering of CAN Bus Commands From Mobile Apps

Authors: Haohuang Wen, Qingchuan Zhao, Qi Alfred Chen, And Zhiqiang Lin

Presented by: Jason Pratama

About The Study

- ▶ Presented in NDSS Symposium 2020 by Haohuang Wen, a Ph.D. student from Ohio State University (February 24, 2020) as a joint work between Ohio State University and University of California Irvine

<https://www.youtube.com/watch?v=Gd07JpS5uG4>

- ▶ Slides from the Authors

<https://www.ndss-symposium.org/wp-content/uploads/24231-slides.pdf>

Overview

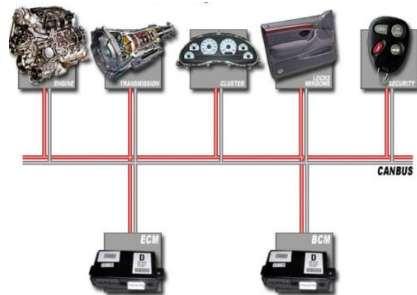
- ▶ This paper is about extracting CAN bus commands from the mobile apps using the **CANHunter** software
- ▶ The software utilizes **Backward Program Slicing** to recover execution path, **Dynamic Forced Execution** to recover syntactics, **UI correlation** and **Function Argument Association** to recover semantics.
- ▶ “CANHunter is able to uncover 182,619 unique CAN bus commands of 360 car models from 21 car makers, and recover the semantics of 86.1% of them.” [1]

[1] Paper Link: <https://www.ndss-symposium.org/wp-content/uploads/2020/02/24231-paper.pdf>

In-vehicle Network and CAN Bus

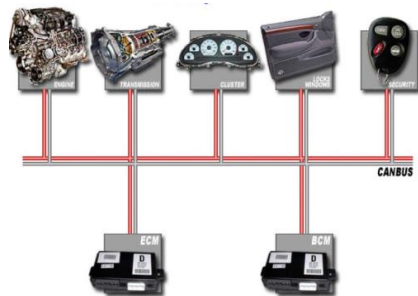


In-vehicle Network and CAN Bus



Control Area Network (CAN) bus.

In-vehicle Network and CAN Bus



Control Area Network (CAN) bus.

S F	Identifier	R	I	D	Data Field							C	A	E
		T	D	L	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	C	K
		R	E	C										

CAN bus command.

Applications of CAN Bus Commands

Driver Behavior Monitoring



An On Board Diagnostic (OBD-II) dongle, used by insurance company Progressive to monitor driver behavior

Applications of CAN Bus Commands

Driver Behavior Monitoring



An On Board Diagnostic (OBD-II) dongle, used by insurance company Progressive to monitor driver behavior

Vehicle Control



An In-Vehicle Infotainment (IVI) system.

Applications of CAN Bus Commands: recently on **Autonomous Driving**



Autoware



Applications of CAN Bus Commands: Security

Vehicle Hacking



The Jeep Cherokee hacking [[MV15](#)].

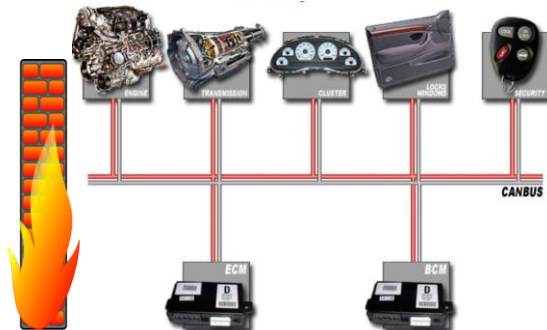
Applications of CAN Bus Commands: Security

Vehicle Hacking



The Jeep Cherokee hacking [[MV15](#)].

Vehicle Security Monitoring



CAN Bus Firewall [[HKD11](#)] [[MA11](#)].

Reverse Engineering of CAN Bus Commands

State-of-the-art

- 1 Fuzzing with random CAN bus commands [[KCR+10](#)] [[LCC+15](#)].
- 2 Manually triggering physical actions and observing the CAN bus [[car](#)] [[wir](#)].

Reverse Engineering of CAN Bus Commands

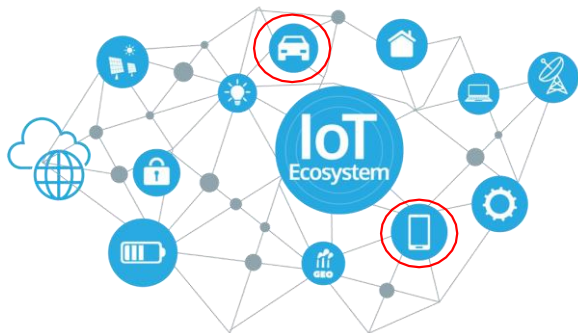
State-of-the-art

- 1 Fuzzing with random CAN bus commands [[KCR+10](#)] [[LCC+15](#)].
- 2 Manually triggering physical actions and observing the CAN bus [[car](#)] [[wir](#)].

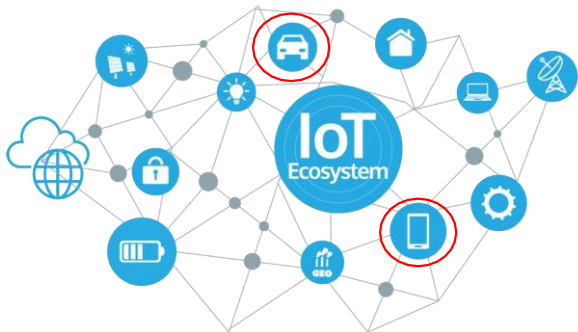
Shortcoming

- 1 **Limited scalability.** CAN bus commands are highly *customized* and *diversified*.
- 2 **Excessive cost.** Significant *manual effort* and real *automobiles* are required.

Our Observation

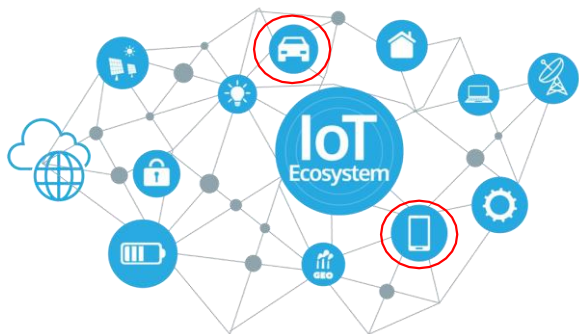


Our Observation



IVI App

Our Observation



IVI App

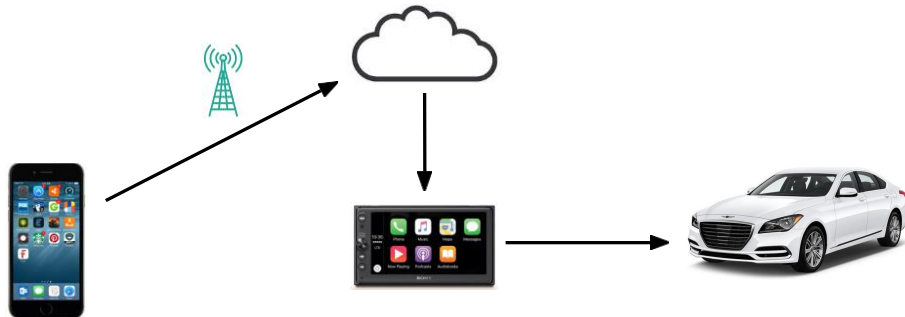


OBD-II Dongle App

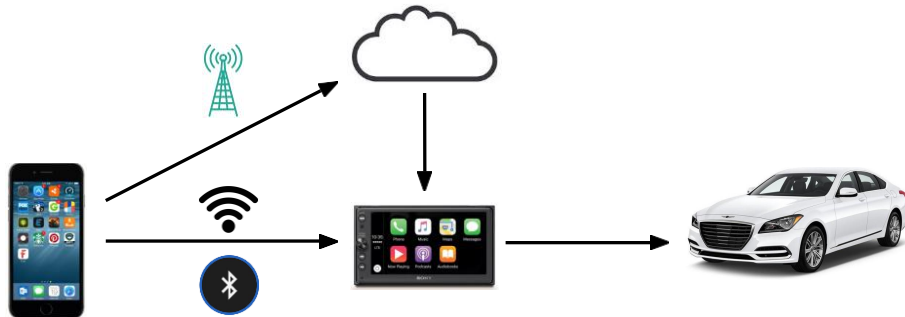
Our Observation



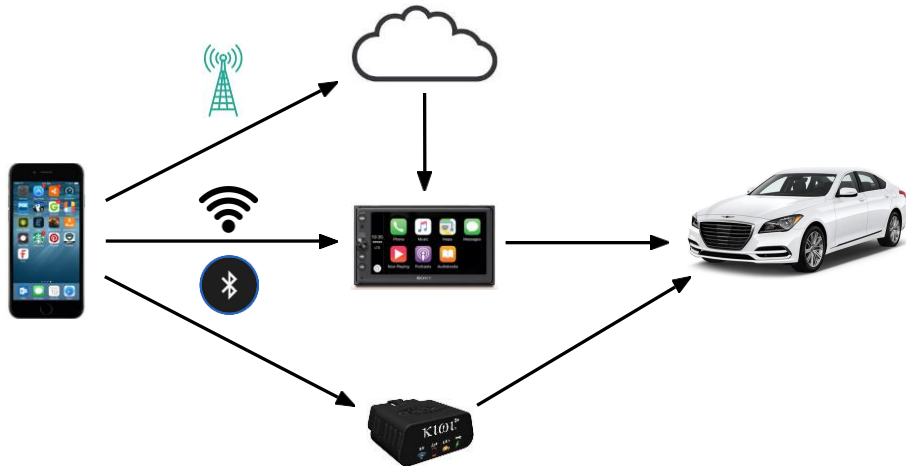
Our Observation



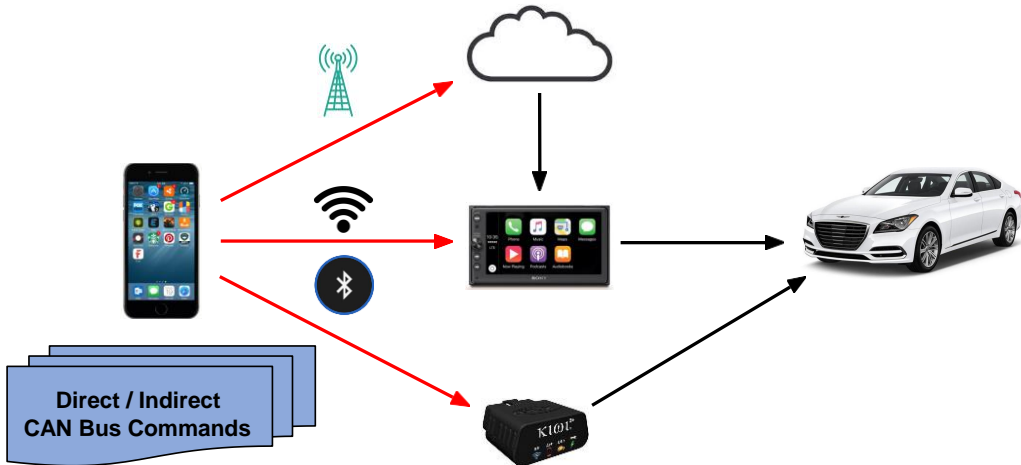
Our Observation



Our Observation



Our Observation



Our Contributions

- 1 Novel Approach.** We propose a cost-effective and automatic approach for reverse engineering CAN bus commands through analyzing mobile apps.
- 2 Effective Techniques.** We design a suite of effective techniques to uncover CAN bus command **syntactics** (structure and format) and **semantics** (meaning and functionality).
- 3 Implementation and Evaluation.** We implemented CANHUNTER on both Android and iOS platforms, and evaluated it with 236 car mobile apps. It discovered 182,619 unique CAN bus commands in which 86.1% of them are recovered with semantics.

Challenges and Insights

Challenges

- 1 Precisely identify CAN bus command execution path
- 2 Command syntactics recovery
- 3 Command semantics recovery

Challenges and Insights

Challenges

- 1 Precisely identify CAN bus command execution path
- 2 Command syntactics recovery
- 3 Command semantics recovery

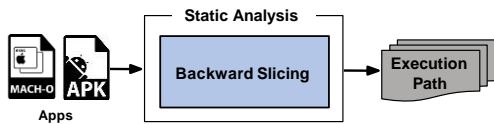
Solutions

- 1 Identify execution path with **backward program slicing**
- 2 Syntactics recovery with **dynamic forced execution**
- 3 Semantics recovery with **UI correlation** and **function argument association**

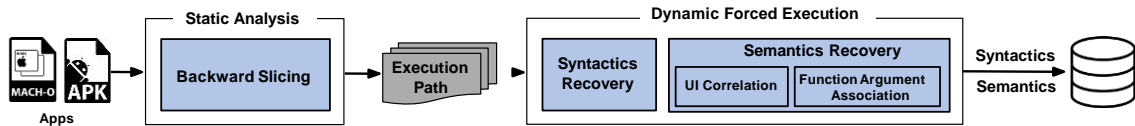
Overview of C A N H U N T E R



Overview of CANHUNTER



Overview of CANHUNTER



Backward Slicing

```
Screen_Info_Diag.viewDidLoad()  
13 v4 = UIButton()  
14 v4.setText("Engine Controls")  
...  
27 v4.addTarget(v4,"initECUs")  
    // register button trigger function
```

```
MD_AllECUsToyota.initECUs()  
4 v12.initWithRequestId("0x7E0","Engine Controls")  
5 v12.frageID = "0x7E0"  
...  
13 v22 = BaseFahrzeug.initWithName("Corolla VIII")  
14 v22.ECU = v12  
...  
25 v25 = v24.createWorkableECUKategorie(v22)
```

```
WorkableModell.createWorkableECUKategorie(a3)  
...  
12 v6 = a3  
13 v7 = v6.ECU.frageID  
...  
18 v8 = v7.substring(2,5)  
19 v9 = NSString.stringWithFormat:@"%03x",v8)  
...  
42 v5.writeValue(v9,v14,1) // Target API
```

The screenshot shows the Carly diagnostic application interface. At the top, there are navigation tabs: "Introduction", "Diagnostic", and the Carly logo. Below the tabs, a header reads "List of possibly built-in ECUs". The main content area displays a list of ECUs, each with a gear icon and a right-pointing arrow:

- 1. Motor Control**
 - Engine
 - Engine Controls
- 2. ABS / DSC / Brake**
 - ABS Brakes
 - Steering Assist
- 3. Airbag**
 - Airbag
 - Gurtstraffer

At the bottom of the list, there are two buttons: "Show Adapter" and "To Full-Version".

Backward Slicing

```
Screen_Info_Diag.viewDidLoad()  
13 v4 = UIButton()  
14 v4.setText("Engine Controls")  
...  
27 v4.addTarget(v4,"initECUs")  
    // register button trigger function
```

```
MD_AllECUsToyota.initECUs()  
4 v12.initWithRequestId("0x7E0","Engine Controls")  
5 v12.frangeID = "0x7E0"  
...  
13 v22 = BaseFahrzeug.initWithName("Corolla VIII")  
14 v22.ECU = v12  
...  
25 v25 = v24.createWorkableECUKategorie(v22)
```

```
WorkableModell.createWorkableECUKategorie(a3)  
...  
12 v6 = a3  
13 v7 = v6.ECU.frangeID  
...  
18 v8 = v7.substring(2,5)  
19 v9 = NSString.stringWithFormat:@"%03 00 02",v8)  
...  
42 v5.writeValue(v9,v14,1) // Target API
```

The screenshot shows the Carly diagnostic application interface. At the top, there are navigation tabs: "Introduction", "Diagnostic", and the Carly logo. Below the tabs, a header reads "List of possibly built-in ECUs". The main content area displays a list of ECUs, each with a gear icon and a title: "1. Motor Control", "Engine", "Engine Controls", "2. ABS / DSC / Brake", "ABS Brakes", "Steering Assist", "3. Airbag", "Airbag", "Gurtstraffer", and "Airbag". At the bottom, there are two buttons: "Show Adapter" and "To Full-Version".

Backward Slicing

```
Screen_Info_Diag.viewDidLoad()
13 v4 = UIButton()
14 v4.setText("Engine Controls")
...
27 v4.addTarget(v4,"initECUs")
    // register button trigger function
```

```
MD_AllECUsToyota.initECUs()
4 v12.initWithRequestId("0x7E0","Engine Controls")
5 v12.frameID = "0x7E0"
...
13 v22 = BaseFahrzeug.initWithName("Corolla VIII")
14 v22.ECU = v12
...
25 v25 = v24.createWorkableECUKategorie(v22)
```

```
WorkableModell.createWorkableECUKategorie(a3)
...
12 v6 = a3
13 v7 = v6.ECU.frameID
...
18 v8 = v7.substring(2,5)
19 v9 = NSString.stringWithFormat:@"%03 00 02",v8)
...
42 v5.writeValue(v9,v14,1) // Target API
```

Introduction Diagnostic Carly

List of possibly built-in ECUs

1. Motor Control

Engine >

Engine Controls >

2. ABS / DSC / Brake

ABS Brakes >

Steering Assist >

3. Airbag

Airbag >

Gurtstraffer >

Show Adapter To Full-Version

Syntactics Recovery

```
Screen_Info_Diag.viewDidLoad()  
13 v4 = UIButton()  
14 v4.setText("Engine Controls")  
...  
27 v4.addTarget(v4,"initECUs")  
    // register button trigger function
```

```
MD_AllECUsToyota.initECUs()  
4 v12.initWithRequestId("0x7E0","Engine Controls")  
5 v12.frageID = "0x7E0" // 0x7E0  
...  
13 v22 = BaseFahrzeug.initWithName("Corolla VIII")  
14 v22.ECU = v12  
...  
25 v25 = v24.createWorkableECUKategorie(v22)
```

```
WorkableModell.createWorkableECUKategorie(a3)  
...  
12 v6 = a3  
13 v7 = v6.ECU.frageID // 0x7E0  
...  
18 v8 = v7.substring(2,5)  
19 v9 = NSString.stringWithForamt("%@ 30 00 02",v8)  
...  
42 v5.writeValue(v9,v14,1) // Target API
```

The screenshot shows the Carly diagnostic application interface. At the top, there are navigation tabs: "Introduction", "Diagnostic", and the Carly logo. Below the tabs, a header reads "List of possibly built-in ECUs". The main content area displays a list of ECUs, each with an icon and a right-pointing arrow:

- 1. Motor Control** (gear icon)
- Engine
- Engine Controls
- 2. ABS / DSC / Brake** (orange ring icon)
- ABS Brakes
- Steering Assist
- 3. Airbag** (airbag icon)
- Airbag
- Gurtstraffer
- ... (partially visible)

At the bottom of the interface, there are two buttons: "Show Adapter" and "To Full-Version".

Syntactics Recovery

```

                Screen_Info_Diag.viewDidLoad()
13  v4 = UIButton()
14  v4.setText("Engine Controls")
...
27  v4.addTarget(v4,"initECUs")
    // register button trigger function

```

```

                MD_AllECUsToyota.initECUs()
4   v12.initWithRequestId("0x7E0","Engine Controls")
5   v12.frameID = "0x7E0" // 0x7E0
...
13  v22 = BaseFahrzeug.initWithName("Corolla VIII")
14  v22.ECU = v12
...
25  v25 = v24.createWorkableECUKategorie(v22)

```

```

                WorkableModell.createWorkableECUKategorie(a3)
...
12  v6 = a3
13  v7 = v6.ECU.frameID // 0x7E0
...
18  v8 = v7.substring(2,5) // 7E0
19  v9 = NSString.stringWithFormat:@"%030 00 02",v8)
    // 7E0 30 00 02" Command Syntactics
...
42  v5.writeValue(v9,v14,1) // Target API

```


Semantics Recovery

```
Screen_Info_Diag.viewDidLoad()  
13 v4 = UIButton()  
14 v4.setText("Engine Controls")  
...  
27 v4.addTarget(v4,"initECUs")  
    // register button trigger function
```

```
MD_AllECUsToyota.initECUs()  
4 v12.initWithRequestId("0x7E0","Engine Controls")  
5 v12.frageID = "0x7E0"  
...  
13 v22 = BaseFahrzeug.initWithName("Corolla VIII")  
14 v22.ECU = v12  
...  
25 v25 = v24.createWorkableECUKategorie(v22)
```

```
WorkableModell.createWorkableECUKategorie(a3)  
...  
12 v6 = a3  
13 v7 = v6.ECU.frageID  
...  
18 v8 = v7.substring(2,5)  
19 v9 = NSString.stringWithFormat:@"%03x",v8  
...  
42 v5.writeValue(v9,v14,1) // Target API
```

Introduction Diagnostic Carly

List of possibly built-in ECUs

1. Motor Control

Engine >

Engine Controls >

2. ABS / DSC / Brake

ABS Brakes >

Steering Assist >

3. Airbag

Airbag >

Gurtstraffer >

Show Adapter To Full-Version

Semantics Recovery

```

                Screen_Info_Diag.viewDidLoad()
13  v4 = UIButton()
14  v4.setText("Engine Controls")
...
27  v4.addTarget(v4,"initECUs")
    // register button trigger function

```

```

                MD_AllECUsToyota.initECUs()
4   v12.initWithRequestId("0x7E0","Engine Controls")
5   v12.frageID = "0x7E0"
...
13  v22 = BaseFahrzeug.initWithName("Corolla VIII")
14  v22.ECU = v12
...
25  v25 = v24.createWorkableECUKategorie(v22)

```

```

                WorkableModell.createWorkableECUKategorie(a3)
...
12  v6 = a3
13  v7 = v6.ECU.frageID
...
18  v8 = v7.substring(2,5)
19  v9 = NSString.stringWithForamt("%@ 30 00 02",v8)
...
42  v5.writeValue(v9,v14,1) // Target API

```

The screenshot shows the Carly diagnostic application interface. At the top, there are navigation tabs: "Introduction", "Diagnostic", and the Carly logo. Below this, a header reads "List of possibly built-in ECUs". The main content area is divided into three large blue sections, each representing a category of ECUs:

- 1. Motor Control**: This section includes "Engine" and "Engine Controls", both with right-pointing chevron arrows.
- 2. ABS / DSC / Brake**: This section includes "ABS Brakes" and "Steering Assist", both with right-pointing chevron arrows.
- 3. Airbag**: This section includes "Airbag" and "Gurtstraffer", both with right-pointing chevron arrows.

At the bottom of the interface, there are two buttons: "Show Adapter" and "To Full-Version".

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019

- 2

- 3

- 4

- 5

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019
- 2
- 3
- 4
- 5

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019

- 2

- 3

- 4

- 5

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019
- 2 182,619 CAN bus commands are discovered
- 3
- 4
- 5

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019
- 2 182,619 CAN bus commands are discovered
- 3 107 apps expose direct CAN bus commands
- 4
- 5

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019
- 2 182,619 CAN bus commands are discovered
- 3 107 apps expose direct CAN bus commands
- 4 109 apps expose indirect commands
- 5

Result Characteristics by App Categories

	# Total	# Dongle	# IVI
Android	122	74	48
iOS	114	72	42
Total (Android \cup iOS)	236	146	90
Overlapped apps (Android \cap iOS)	79	38	41

Table: Distribution of collected apps.

- 1 We crawled 236 vehicle apps in April 2019
- 2 182,619 CAN bus commands are discovered
- 3 107 apps expose direct CAN bus commands
- 4 109 apps expose indirect commands
- 5 20 apps are obfuscated

Result Characteristics by App Categories

Indirect (i.e., Interpreted) CAN Commands

- 1 IVI apps usually use *interpreted commands* for vehicle control
- 2 Interpreted commands are usually strings or numbers

App	Content	Sent to Cloud	Sent to Vehicle
AcuraLink	HORN_LIGHT, UNLOCK, LOCATION	✓	
Alpine	frontSpeakerPattern, rearSpeakerPattern		✓
Alpine Tunelt	RESUME, PHONE_DIAL_END, AUDIO_FOCUS	✓	
Audi MMI Connect	LOCK, UNLOCK, G_STAT, FIND_CAR	✓	
Carbin Control	Climate_Control_Temperature, Control_Fan_Speed		✓
Car-Net	Unlock:2, Lock:3, Flash:0, Hornlight:1		✓

Table: Interpreted commands from IVI apps.

Result Characteristics by App Categories

Indirect (i.e., Interpreted) CAN Commands

- 1 IVI apps usually use *interpreted commands* for vehicle control
- 2 Interpreted commands are usually strings or numbers

App	Content	Sent to Cloud	Sent to Vehicle
AcuraLink	HORN_LIGHT, UNLOCK, LOCATION	✓	
Alpine	frontSpeakerPattern, rearSpeakerPattern		✓
Alpine Tunelt	RESUME, PHONE_DIAL_END, AUDIO_FOCUS	✓	
Audi MMI Connect	LOCK, UNLOCK, G_STAT, FIND_CAR	✓	
Carbin Control	Climate_Control_Temperature, Control_Fan_Speed		✓
Car-Net	Unlock:2, Lock:3, Flash:0, Hornlight:1		✓

Table: Interpreted commands from IVI apps.

Result Characteristics by Car Models

We identify CAN bus commands from over 360 car models across 21 car makers

Car Maker	# Commands	Car Model
Audi	51,517	A3, A4, A5, A6, A7, A8, Q3, Q5, Q7, S3, S4
Volkswagon	44,504	Cabrio, Corrado, Caddy, Gol, Golf, Jetta,
Skoda	11,009	Citigo, Fabia, Rapid, Superb, Yeti
Toyota	9,030	Auris, Avensis, Camry, Corolla, Prius, RAV4
BMW	8,963	Series 1, 3, 5, M5, X5
Seat	8,277	Ibiza, Leon, Altea, Mii, Toledo, Arosa
Mercedes	7,247	Benz
Lexus	6,087	CT200, ES350, GS350, GX460, RX450, IS460

Table: Distribution of CAN Bus commands over part of car makers.

Result Characteristics by Semantics

- 1 157,296 (86.1%) CAN bus commands are recovered with semantics
- 2 The semantics can be categorized into *diagnosis* and *vehicle control*

Result Characteristics by Semantics

- 1 157,296 (86.1%) CAN bus commands are recovered with semantics
- 2 The semantics can be categorized into *diagnosis* and *vehicle control*

Semantics	# Commands	Category
Engine speed	460	Diagnosis
Coolant temperature	281	Diagnosis
Throttle angle	256	Diagnosis
Oil temperature	176	Diagnosis
Single door lock remote	60	Control
Blink on unlock key	42	Control
Sound on remote lock volume	40	Control
Auto unlock when moving	27	Control

Table: Distribution of CAN bus commands over part of semantics.

Correctness Evaluation

- 1 Over 70% of the command syntactics and semantics are validated
- 2 We tried the following three sources for validation:
 - 1 Public resource
 - 2 Cross validation
 - 3 Real car testing

Correctness Evaluation

Car Model	Syntac.	Semantics (Ground Truth)	Semantics (Our Result)	Matched
Toyota Prius	0x727	Transmission	Transmission	✓
	0x7A1	Steering Assist	Steering Assist	✓
	0x7A2	Park Assist	APGS	✓
	0x7E0	Engine Controls	ECT	✓
Audi A3	0x70C	SteeringWheel	Steering wheel	✓
	0x714	DashBoard	Instrument	✓
	0x7E1	TCMDQ	Transmission	✓
Seat Ibiza	0x713	Brake1ESP	ABS Brakes	✓
	0x714	KombiUDS	Instruments	✓
Honda Civic	0x158	Speed	EAT_TRANS_SPEED	✓
	0x17C	Engine RPM	ENG_STATUS	✓
	0x1A4	VSA_STATUS	VSA_WARN_STATUS_ABS	✓
	0x324	Water Tempreature	ENG_TEMP	✗
	0x305	SEATBELT_STATUS	SRS_EDR_DELTA_VMAX	✗
	0x35E	CAMERA_MESSAGES	FCM_WARN_STATUS	✗

Table: Part of the commands validated with **public resources**.

Correctness Evaluation

Car Model	Syntac.	Semantics (Ground Truth)	Semantics (Our Result)	Matched
Toyota Prius	0x727	Transmission	Transmission	✓
	0x7A1	Steering Assist	Steering Assist	✓
	0x7A2	Park Assist	APGS	✓
	0x7E0	Engine Controls	ECT	✓
Audi A3	0x70C	SteeringWheel	Steering wheel	✓
	0x714	DashBoard	Instrument	✓
	0x7E1	TCMDQ	Transmission	✓
Seat Ibiza	0x713	Brake1ESP	ABS Brakes	✓
	0x714	KombiUDS	Instruments	✓
Honda Civic	0x158	Speed	EAT_TRANS_SPEED	✓
	0x17C	Engine RPM	ENG_STATUS	✓
	0x1A4	VSA_STATUS	VSA_WARN_STATUS_ABS	✓
	0x324	Water Tempreature	ENG_TEMP	✗
	0x305	SEATBELT_STATUS	SRS_EDR_DELTA_VMAX	✗
	0x35E	CAMERA_MESSAGES	FCM_WARN_STATUS	✗

Table: Part of the commands validated with **public resources**.

Correctness Evaluation

App	Android		iOS		Overlapped	
	# Syn.	# Sem.	# Syn.	# Sem.	# Syn.	# Sem.
BlueDriver	304	304	304	304	304	304
Carista	105,198	105,198	105,198	105,198	105,198	105,198
Carly for BMW	14,377	14,377	16,427	16,427	13,480	13,480
Carly for Mercedes	7,921	6,528	1,698	1,698	1,393	1,393
Carly for Toyota	5,305	5,266	39	39	39	39
Carly for VAG	16,402	7,283	18,627	10,429	7,283	7,283
CarVantage	41	41	41	41	41	41
Engie	144	144	68	68	68	68
inCarDoc	160	160	160	160	160	160
Kiwi OBD	220	220	6	6	6	6

Table: Part of the **cross-platform validation** (commands across different platforms) results.

Correctness Evaluation

App	Android		iOS		Overlapped	
	# Syn.	# Sem.	# Syn.	# Sem.	# Syn.	# Sem.
BlueDriver	304	304	304	304	304	304
Carista	105,198	105,198	105,198	105,198	105,198	105,198
Carly for BMW	14,377	14,377	16,427	16,427	13,480	13,480
Carly for Mercedes	7,921	6,528	1,698	1,698	1,393	1,393
Carly for Toyota	5,305	5,266	39	39	39	39
Carly for VAG	16,402	7,283	18,627	10,429	7,283	7,283
CarVantage	41	41	41	41	41	41
Engie	144	144	68	68	68	68
inCarDoc	160	160	160	160	160	160
Kiwi OBD	220	220	6	6	6	6

Table: Part of the **cross-platform validation** (commands across different platforms) results.

Correctness Evaluation

App	Android		iOS		Overlapped	
	# Syn.	# Sem.	# Syn.	# Sem.	# Syn.	# Sem.
BlueDriver	304	304	304	304	304	304
Carista	105,198	105,198	105,198	105,198	105,198	105,198
Carly for BMW	14,377	14,377	16,427	16,427	13,480	13,480
Carly for Mercedes	7,921	6,528	1,698	1,698	1,393	1,393
Carly for Toyota	5,305	5,266	39	39	39	39
Carly for VAG	16,402	7,283	18,627	10,429	7,283	7,283
CarVantage	41	41	41	41	41	41
Engie	144	144	68	68	68	68
inCarDoc	160	160	160	160	160	160
Kiwi OBD	220	220	6	6	6	6

Table: Part of the **cross-platform validation** (commands across different platforms) results.

Correctness Evaluation

Car model	# Overlapped		App1	App2
	Android	iOS		
Audi A4	52	52	Carista	Carly for VAG
Audi A6	22	22	Carista	Carly for VAG
Seat Leon	19	19	Carista	Carly for VAG
Skoda Fabia	0	24	Carista	Carly for VAG
VW Caddy	0	12	Carista	Carly for VAG
VW Polo	52	52	Carista	Carly for VAG
VW Tiguan	8	0	Carista	Carly for VAG
Skoda Superb	0	20	Carista	Carly for VAG
Porsche Cayenne	0	72	Carly for VAG	Carly for Partners
Toyota Prius	39	39	Carly for Toyota	Carista
BMW 550i	8	8	Carly for BMW	Carista

Table: Part of the **in-platform validation** (commands within the same platforms) results

Correctness Evaluation

Car model	# Overlapped		App1	App2
	Android	iOS		
Audi A4	52	52	Carista	Carly for VAG
Audi A6	22	22	Carista	Carly for VAG
Seat Leon	19	19	Carista	Carly for VAG
Skoda Fabia	0	24	Carista	Carly for VAG
VW Caddy	0	12	Carista	Carly for VAG
VW Polo	52	52	Carista	Carly for VAG
VW Tiguan	8	0	Carista	Carly for VAG
Skoda Superb	0	20	Carista	Carly for VAG
Porsche Cayenne	0	72	Carly for VAG	Carly for Partners
Toyota Prius	39	39	Carly for Toyota	Carista
BMW 550i	8	8	Carly for BMW	Carista

Table: Part of the **in-platform validation** (commands within the same platforms) results

Correctness Evaluation

Car model	# Overlapped		App1	App2
	Android	iOS		
Audi A4	52	52	Carista	Carly for VAG
Audi A6	22	22	Carista	Carly for VAG
Seat Leon	19	19	Carista	Carly for VAG
Skoda Fabia	0	24	Carista	Carly for VAG
VW Caddy	0	12	Carista	Carly for VAG
VW Polo	52	52	Carista	Carly for VAG
VW Tiguan	8	0	Carista	Carly for VAG
Skoda Superb	0	20	Carista	Carly for VAG
Porsche Cayenne	0	72	Carly for VAG	Carly for Partners
Toyota Prius	39	39	Carly for Toyota	Carista
BMW 550i	8	8	Carly for BMW	Carista

Table: Part of the **in-platform validation** (commands within the same platforms) results

Correctness Evaluation



A Toyota RAV4.



A Toyota Corolla.

Correctness Evaluation

Command (RAV4)	Command (Corolla)	Semantics
750 ... 14 1A 26	750 ... 1A 65 02	Wireless door locking
750 ... 14 92 26	750 ... 92 65 02	Blink turn signals
750 ... 14 9A 06	750 ... 9A 45 02	Panic Function on remote
750 ... 14 9A 25	750 ... 9A 61 02	Relock automatically
750 ... 14 9A 26	750 ... 8A 65 02	Beep when locking
750 ... 11 00 60	750 ... 14 06 00	Unlock via physical key
750 ... 11 80 20	750 ... 11 C0 20	Unlock when shifting into gear
7C0 ... 3B A2 40	7C0 ... 3B A2 40	Display unit (MPG)
7C0 ... 3B 74 A0	7C0 ... 3B A7 C0	Seat belt warning (driver)
7CC ... 00 01 00	7CC ... 3B 82 00	Fan Speed

Table: Part of commands validated with **real-car testing**.

Correctness Evaluation

Command (RAV4)	Command (Corolla)	Semantics
750 ... 14 1A 26	750 ... 1A 65 02	Wireless door locking
750 ... 14 92 26	750 ... 92 65 02	Blink turn signals
750 ... 14 9A 06	750 ... 9A 45 02	Panic Function on remote
750 ... 14 9A 25	750 ... 9A 61 02	Relock automatically
750 ... 14 9A 26	750 ... 8A 65 02	Beep when locking
750 ... 11 00 60	750 ... 14 06 00	Unlock via physical key
750 ... 11 80 20	750 ... 11 C0 20	Unlock when shifting into gear
7C0 ... 3B A2 40	7C0 ... 3B A2 40	Display unit (MPG)
7C0 ... 3B 74 A0	7C0 ... 3B A7 C0	Seat belt warning (driver)
7CC ... 00 01 00	7CC ... 3B 82 00	Fan Speed

Table: Part of commands validated with **real-car testing**.

Related Work

1 CAN and Vehicle Security.

- ▶ Vehicle attack [[MV14](#)] [[CMK+11](#)] [[MRHM16](#)] [[MV15](#)] [[Sta13](#)] [[MV13](#)] and CAN reverse engineering [[KCR+10](#)].
- ▶ Defenses of CAN bus. Anomaly detection [[CS16](#)] [[MGF10](#)] [[NLJ08](#)], forensics measures [[HKD11](#)] and delayed data authentication [[NLJ08](#)].

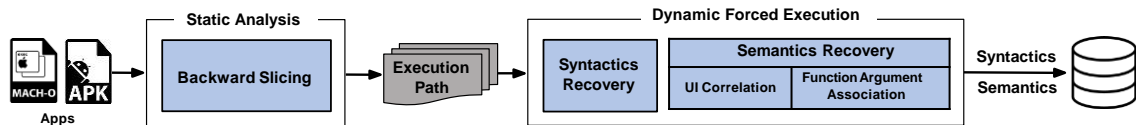
2 Protocol Reverse Engineering. Polyglot [[CYLS07](#)], AutoFormat [[LJXZ08](#)], Discoverer [[CKW07](#)], Tupni [[CPC+08](#)], and ReFormat [[WJC+09](#)].

3 Forced execution. J-Force [[KKK+17](#)] for JavaScript applications, X-Force [[PDZ+14](#)] and Limbo [[WC07](#)] for binaries, and Dexism [[EJS18](#)].

Future Work

- 1 Handling obfuscation.** The current implementation of CANHUNTER is not resilient to anti-analysis techniques such as control flow obfuscation. Deobfuscation techniques can be applied to address this limitation.
- 2 Investigating other vehicle commands.** CANHUNTER reported a great number of AT commands for vehicle diagnosis, and also interpreted commands for vehicle control. These commands are worth of security attention.
- 3 Reverse engineering of other IoT protocols.** CANHUNTER has the potential to be extended to reverse engineer the syntactics and semantics of other IoT protocols.

CANHUNTER



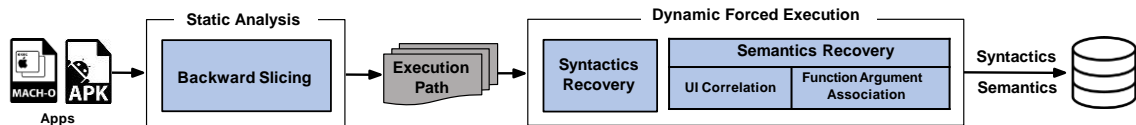
CANHUNTER

- ▶ An automatic and cost-effective approach of reverse engineering CAN bus commands from mobile apps
- ▶ Recover both the syntactics and semantics of CAN bus commands

Implementation and Evaluation

- ▶ We implemented CANHUNTER on both Android and iOS platforms
- ▶ We evaluated CANHUNTER on 236 apps in which 182,619 commands are discovered with 86% recovered with semantics

CANHUNTER



CANHUNTER









- ▶ An automatic and cost-effective approach of reverse engineering CAN bus commands from mobile apps
- ▶ Recover both the syntactics and semantics of CAN bus commands

Implementation and Evaluation

- ▶ We implemented CANHUNTER on both Android and iOS platforms
- ▶ We evaluated CANHUNTER on 236 apps in which 182,619 commands are discovered with 86% recovered with semantics

The source code and dataset is available at <https://github.com/OSUsecLab/CANHunter>.

References I

-  *How to Hack a Car - A Quick Crash Course*, <https://medium.freecodecamp.org/hacking-cars-a-guide-tutorial-on-how-to-hack-a-car-5eafcfbbb7ec>.
-  Weidong Cui, Jayanthkumar Kannan, and Helen J Wang, *Discoverer: Automatic protocol reverse engineering from network traces.*, USENIX Security Symposium, 2007, pp. 1-14.
-  Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al., *Comprehensive Experimental Analyses of Automotive Attack Surfaces*, USENIX Security Symposium, 2011.
-  Weidong Cui, Marcus Peinado, Karl Chen, Helen J Wang, and Luis Irun-Briz, *Tupni: Automatic Reverse Engineering of Input Formats*, ACM conference on Computer and Communications Security (CCS), 2008.
-  Kyong-Tak Cho and Kang G Shin, *Fingerprinting Electronic Control Units for Vehicle Intrusion Detection*, USENIX Security Symposium, 2016.
-  Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song, *Polyglot: Automatic extraction of protocol message format using dynamic binary analysis*, Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 317-329.
-  Mohamed Elsabagh, Ryan Johnson, and Angelos Stavrou, *Resilient and scalable cloned app detection using forced execution and compression trees*, 2018 IEEE Conference on Dependable and Secure Computing (DSC), IEEE, 2018, pp. 1-8.
-  Tobias Hoppe, Stefan Kiltz, and Jana Dittmann, *Security threats to automotive can networks—practical examples and selected short-term countermeasures*, Reliability Engineering & System Safety **96** (2011), no. 1, 11-25.

References II

-  Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al., *Experimental Security Analysis of a Modern Automobile*, IEEE Symposium on Security and Privacy (S&P), 2010.
-  Kyungtae Kim, I Luk Kim, Chung Hwan Kim, Yonghwi Kwon, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu, *J-force: Forced execution on javascript*, Proceedings of the 26th international conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017, pp. 897-906.
-  Hyeryun Lee, Kyunghye Choi, Kihyun Chung, Jaein Kim, and Kangbin Yim, *Fuzzing can packets into automobiles*, 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, IEEE, 2015, pp. 817-821.
-  Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang, *Automatic protocol format reverse engineering through context-aware monitored execution*, Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08) (San Diego, CA), February 2008.
-  Michael Mu̇ter and Naim Asaj, *Entropy-based Anomaly Detection for In-vehicle Networks*, IEEE Intelligent Vehicles Symposium (IV), 2011.
-  Michael Mu̇ter, Andr'e Groll, and Felix C Freiling, *A structured approach to anomaly detection for in-vehicle networks*, Information Assurance and Security (IAS), 2010 Sixth International Conference on, IEEE, 2010, pp. 92-98.
-  Sahar Mazloom, Mohammad Rezaeirad, Aaron Hunter, and Damon McCoy, *A Security Analysis of an In-Vehicle Infotainment and App Platform*, Usenix Workshop on Offensive Technologies (WOOT), 2016.
-  Charlie Miller and Chris Valasek, *Adventures in automotive networks and control units*, Def Con **21** (2013), 260-264.

References III

-  _____, *A survey of remote automotive attack surfaces*, black hat USA **2014** (2014), 94.
-  _____, *Remote exploitation of an unaltered passenger vehicle*, Black Hat USA **2015** (2015), 91.
-  Dennis K Nilsson, Ulf E Larson, and Erland Jonsson, *Efficient in-vehicle delayed data authentication based on compound message authentication codes*, Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th, IEEE, 2008, pp. 1-5.
-  Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu, Zhiqiang Lin, and Zhendong Su, *X-force: Force-executing binary programs for security applications.*, USENIX Security Symposium, 2014, pp. 829-844.
-  Jason Staggs, *How to hack your mini cooper: reverse engineering can messages on passenger automobiles*, Institute for Information Security (2013).
-  Jeffrey Wilhelm and Tzi-cker Chiueh, *A forced sampled execution approach to kernel rootkit identification*, International Workshop on Recent Advances in Intrusion Detection, Springer, 2007, pp. 219-235.
-  *Wireshark: The World's Most Popular Network Protocol Analyzer*, <http://www.wireshark.org/>.
-  Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, and Mike Grace, *ReFormat: Automatic Reverse Engineering of Encrypted Messages*, European Symposium on Research in Computer Security (ESORICS), 2009.

Discussion Points

- ▶ Are the results valid/accurate?
- ▶ Will the attack work on other cars/apps using different communication?
- ▶ Is there any way to defend against this kind of attack?

Are the results valid/accurate?

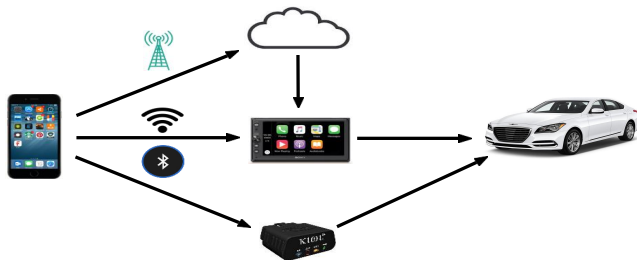
- ▶ “CANHunter is able to uncover 182,619 unique CAN bus commands of 360 car models from 21 car makers, and recover the semantics of 86.1% of them.” [1]
- ▶ “We have also evaluated their correctness (both syntactics and semantics) using public resources, cross-platform and cross-app validation, and also real car testing, with which over 70% of all the uncovered commands are validated.” [1]
- ▶ “We observe no inconsistency in cross-platform and cross-app validation. While there are 3 semantic inconsistency among 241 manually validated CAN bus commands from public resources and real-car testing, we find that these three cases are actually caused by mistakes from app developers.” [1]

Are the results valid/accurate?

Car Model	Syntac.	Semantics (Ground Truth)	Semantics (Our Result)	Matched
Toyota Prius	0x727	Transmission	Transmission	✓
	0x7A1	Steering Assist	Steering Assist	✓
	0x7A2	Park Assist	APGS	✓
	0x7E0	Engine Controls	ECT	✓
Audi A3	0x70C	SteeringWheel	Steering wheel	✓
	0x714	DashBoard	Instrument	✓
	0x7E1	TCMDQ	Transmission	✓
Seat Ibiza	0x713	Brake1ESP	ABS Brakes	✓
	0x714	KombiUDS	Instruments	✓
Honda Civic	0x158	Speed	EAT_TRANS_SPEED	✓
	0x17C	Engine RPM	ENG_STATUS	✓
	0x1A4	VSA_STATUS	VSA_WARN_STATUS_ABS	✓
	0x324	Water Tmperature	ENG_TEMP	✗
	0x305	SEATBELT_STATUS	SRS_EDR_DELTA_VMAX	✗
	0x35E	CAMERA_MESSAGES	FCM_WARN_STATUS	✗

Will the attack work on other cars/apps using different communication?

- ▶ In the presentation, the author mentions that he has validated the CAN bus commands using real cars via the OBD-II dongle (Toyota RAV4 and Corolla) [1]



Is there any way to defend against this kind of attack?

- ▶ The attack take advantage of the communication between mobile app and IVI/ODB dongle.
- ▶ Communication paths? (ODB dongle, wifi/Bluetooth IVI, cloud)
- ▶ Direct/indirect commands?
- ▶ Encryption?
- ▶ Obfuscation techniques?

Links

- ▶ NDS Website:

<https://www.ndss-symposium.org/ndss-paper/automated-cross-platform-reverse-engineering-of-can-bus-commands-from-mobile-apps/>

- ▶ Presentation by Haohuang Wen:

<https://www.youtube.com/watch?v=Gd07JpS5uG4>

- ▶ Paper Link:

<https://www.ndss-symposium.org/wp-content/uploads/2020/02/24231-paper.pdf>

- ▶ Original Slides:

<https://www.ndss-symposium.org/wp-content/uploads/24231-slides.pdf>

- ▶ CANHunter Source Code:

<https://github.com/OSUSecLab/CANHunter>