ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645
Indian Journal of Science and Technology, *Vol 12(41), DOI: 10.17485/ijst/2019/v12i41/145568, November 2019*

Towards the Prevention of Car Hacking: A Threat to Automation Industry

Pooja Sharma[1], Vaibhav Jha[1], Vasudha Arora[1] and Prateek Jain[2],*

[1]Department of Computer Science & Engineering,
Manav Rachna International Institute of Research & Studies, Faridabad 121003, India,
pooja02998@gmail.com, vaibhavkumar2012@gmail.com
[2]Accendere CL Educate Ltd, New Delhi 110044, India; prateek.jain@accendere.co.in

Abstract

Background/objectives: Connectivity provides a safer environment, but it also acts as a backbone to provide attack surface to hackers. There are millions of cars on the road today, and so many are expected to be in future; there might be a risk to the passengers, vehicle drivers, etc. **Methods/statistical analysis:** This study discusses the issue of car hacking which is one of the real threats to automobile as well as automation, and how we can prevent it by studying the details about the controller area network (CAN) bus architecture so that the auto manufacturer gives more emphasis to developing a secure vehicular information system. **Findings:** Hackers gain access to the car system via the internet, Bluetooth, etc. As much as a car is automated, it is much more vulnerable to cyber-attack. When a car is connected to the internet, it provides access to the vehicle's delicate CAN bus. Hackers can hijack non-safety and safety-critical functions such as steering, accelerator, brake and clutches by sending commands. **Improvements/applications:** This study gives a general overview of how we can validate the security features of the vehicle so that we can secure our vehicle from black hat hackers, resulting in saving millions of people who could be a victim of such menacing cyber-attacks.

Keywords: Car Hacking, CAN Bus, Cyber-attacks, OBD Hacking

1. Introduction

In 2010 researchers at the Center for Automotive Embedded Systems Security (CAESS), California, detected that gaining a connection with ODB-II port of the car can easily disable the breaks and switch on/off the engine. They embeded a malicious code in the car's telematics unit and were able to break its network security.[1]

In 2013 cyber security researchers Charlie Miller and Chris Valasek have shown to *The Forbes* how they could access vehicle controls through a laptop computer via the ODB port.[2] In 2014 Mathew Solnik, an information security researcher, misguided the car's engine, brakes and security systems from his laptop by wirelessly connecting to the ODB port in the controller area network (CAN) bus system.

In 2015 host Lesley Stahl, in a demonstration by the U.S. military's Defense Advanced Research Projects Agency (DARPA), drove a car remotely using his laptop.

In 2017, William Hatzer and Arjun Kumar at Rapid7 claimed that Hyundai Blue Link app can be a reason of the "MAN IN THE MIDDLE ATTACK". Hackers can easily have access to the personal information of the user. In today's world, much of the objects that we use in day-to-day life and at homes are increasingly becoming controllable by the remote. Due to technology there is a need to automate everything and to influence and automate the object's behavior that once required local and manual input. Thus, automation has become the necessity and an important issue to tackle with.[3]

Vehicle is one of the most typical productions of industries. A vital and necessary consideration of a car is safety. In the past, car designers did not need to think about a problem that a car could be possibly attacked and controlled by hackers. But with a significant development in recent years, IT crimes have become a serious problem that cannot be ignored. The deficiency of safety on

*Author for correspondence

electronic and information system of cars should get more attention.

Considering the modern vehicles, it is quite easy to immediately picture a scenario where a car is controlled using a smartphone. Moreover, this leads to a rise in autonomous vehicles as well as self-driving cars, and this represents the next logical step and is a reality for current scenario. Due to a rise in the complexity of the electronic circuit of the vehicles, there is a need to understand these electronic control units (ECUs) as well as their importance in monitoring the various subsystems of a car. In addition, modern vehicles are able to communicate with other devices using wireless interfaces, potentially exposing the internal network of the car to vulnerabilities. It is our belief that the current state-of-the-art internal communication systems used in modern cars are not ready to handle threats from external attackers.[3]

Currently, ECUs are widely used in cars for controlling and achieving most functions of cars. A vehicle may have dozens to hundreds of ECUs to work with. In this case, CAN plays a role that connects ECUs together. The hardware of CAN is called the CAN bus.[4]

One feature of CAN is that it follows a massage-based protocol to transfer information. In a real car, the contents of CAN messages depend on the car's designer, but the form of these messages certainly obeys a particular standard (ISO 11898). Because of this, it is not difficult to analyze these messages merely by reading them. Besides, the message form of CAN data frame which is used for sending status information or instructions does not include any field for identifying the sender of messages.[4]

2. Different Ways to Unlock a Car

2.1 Using an Arduino-based RF Transceiver

The first attack we performed was done by a radio device which costed just 2000 INR with a radio receiver, a small control board, but is capable of spying and extracting continues code values used by keyless entry systems (Figure 1).[5,6]

We included code values in the signal which is sent every time when a driver presses the key buttons, which is then used together to emulate a key that is unique for every vehicle. Then we performed reverse engineering into one component inside a car's network and were able to extract a cryptographic key. Then we combined the two secret keys, which enabled us to clone the key fob and access the car.

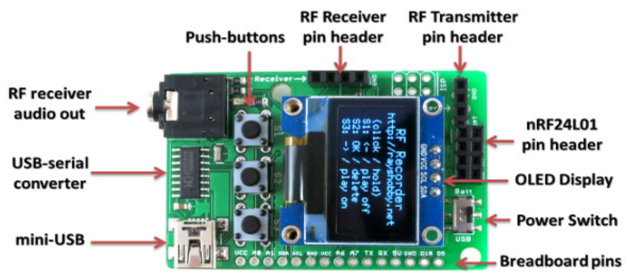


Figure 1. Arduino-based RF transceiver.[6]

2.2 Hijack with HiTag2 and a Radio Device in 60 Seconds

In the second method, we used a cryptographic scheme called HiTag2 which is old but still used in millions of vehicles, including Lancia, Opel, Renault, Ford, Alfa Romeo, Chevrolet and Peugeot.

To perform this attack, a hacker needs a tiny radio setup which is similar to the one used in the previous hack. Using a radio device, we were able to read and intercept the strings of the coded signals from the car's key fob.

We discovered that flaws in the HiTag2 scheme with the help of rolling codes would allow cracking the cryptographic key in a second. So these two methods were just for unlocking the car, making it accessible for hackers or thieves to steal it. But if we use a digital system instead of rolling codes, it would be more secure. To hack a car, unlocking it is the first step of every hacker, so that they can tamper the CAN bus system and the OBD port.

3. Tampering the CAN Bus

Two security researchers Javier Vazquez-Vidal and Alberto Garcia Illera have developed CAN Hack, a tiny device, which is even smaller than our mobiles, to hack cars. The device costs 1500 INR, but is able to give away the entire control of any car to an attacker from headlights and windows to its steering angles[7] and brakes (Figure 2).[8]

By injecting a malicious code into the CAN ports makes it possible for an attacker to send wireless commands remotely from a computer. It can take just 5 minutes or less for coming into the action and then walk away. Whether it takes 1 minute or 1 year, a hacker could wait and then trigger it to do whatever one has programmed it to do. Once hackers have the control of this network, they can control locks, lights, steering and even breaks (Figure 3).[9]

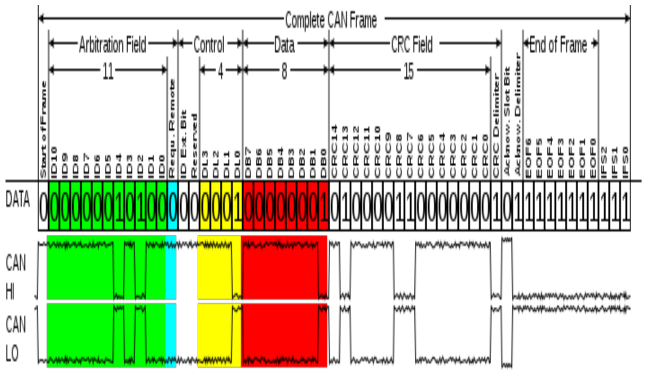


Figure 2. CAN bus system.[8]

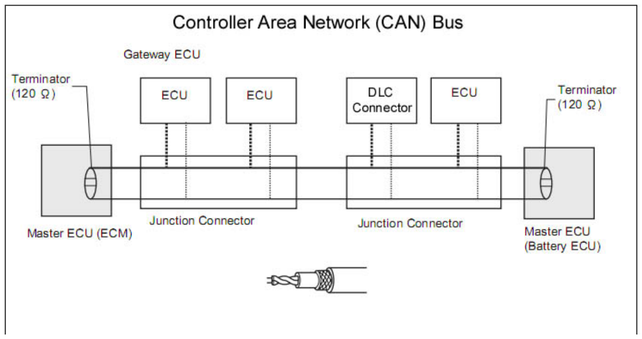


Figure 3. CAN bus architecture.[9]

3.1. CAN Bus Architecture

CAN bus is called the heart of any modern vehicle's interconnected systems. The CAN bus is a single, centralised network bus on which all of a vehicle's data traffic is broadcast. Every command from the operator is being carried by the CAN bus system such as "apply the brakes" or "roll down the windows" to readouts from sensors reporting engine temperature or tire pressure. The emergence of the CAN[10] bus brought improvements in efficiency and a reduction in complications, thus reducing wiring costs too (Figure 4).

But with the car hacking toolkit (CHT), hackers have already tested on different vehicles and successfully did tricks, which include setting off alarms, affecting the steering, applying brakes, and switching off headlights. We performed this with the help of Bluetooth, but we could also do the same with the help of Raspberry Pi or a WiFi router, enabling the CHT to control the car from a far distance.

4. Understanding the OBD Port

All the vehicles come equipped with an OBD (On Board Diagnostic) port, which allows the external devices to interface with a car's computer system. We generally find

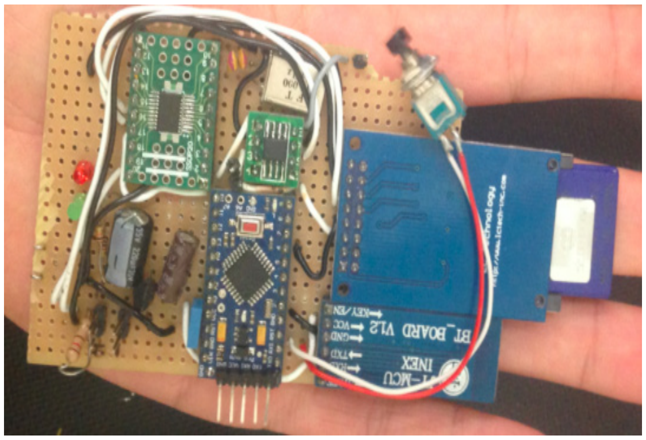


Figure 4. Car hacking toolkit.

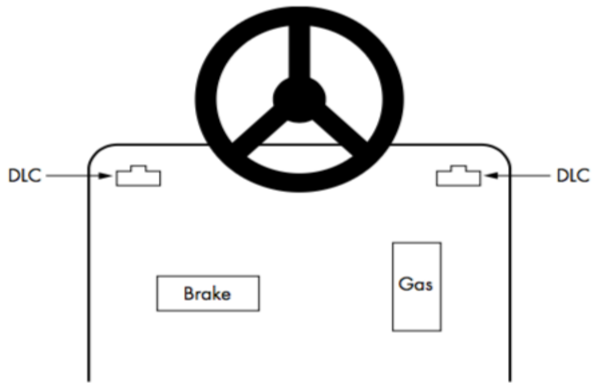


Figure 5. OBD layout.

this connector under[11] the steering column just above the break and accelerator panel or hidden elsewhere on the dashboard (Figure 5).

5. Layman Procedure

First of all, as soon as we gain access to an OBD board, we are able to extract every information of the car. We can use that information to understand the architecture and behaviour of that car.

But changes could only be done when a hacker or attacker has access to the CAN bus architecture. For communicating with the CAN bus, we require various drivers and software. The best technique would be to amalgamate the CAN tools along with their various interfaces to form a customary interface so that we could easily share and communicate between different tools (Figure 6).[12]

Sockets CAN, an open source driver of CAN and official API of Linux kernel, makes it possible to make tools to support CAN. Socket CAN applications use the standard C socket which comes along with a custom

Figure 6. Tampering with the CAN system.[12]

network protocol family, PF_CAN. With the help of this functionality, kernel handles CAN device drivers to communicate with existing networking hardware, thus providing user-space utilities and a common interface.[13]

We used this git command to install CAN utils in our package manager.

$ sudo apt-get install can-utils
$ git clone https://github.com/linux-can/can-utils

6. Data Recorder Logging

All vehicles that came after 2015 are equipped with a kind of black box called event data recorder (EDR), but it can record only a finite portion of information that a black box on an aircraft could do. Information stored on an EDR is as follows[14]:

6.1. Airbag Deployment

Generally airbags open when a car gets hit on its bonnet, but here with the amalgamation of codes we can open it anytime.

6.2. Steering Angles

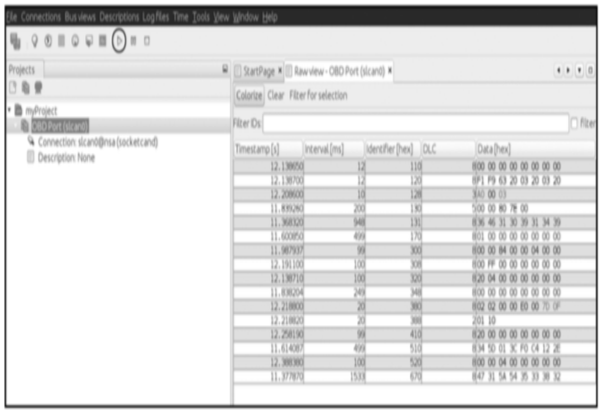
Turning the steering into wrong angles might lead to an accident.

6.3. Vehicle Speed

Engine speed could be tampered using a reverse CAN; thus, acceleration could be suddenly boosted, leading to a major accident.

6.4. Brake Status

Brakes could be applied anytime by the attacker, which might result in a tragedy.


File Connections Busviews Descriptions Logfiles Time Tools View Window Help
Projects
myProject
OBD Port (slcan0)
Connection: slcan0@nsa (socketcand)
Description: None
StartPage
Raw view - OBD Port (slcan0)
Colorize
Clear
Filter for selection
Filter IDs
filter

Timestamp [s]	Interval [ms]	Identifier [hex]	DLC	Data [hex]
12.138650	12	110	8	00 00 00 00 00 00 00 00
12.138700	12	120	8	F1 F9 63 20 03 20 03 20
12.208600	10	128	3	A0 00 03
11.839260	200	130	5	00 00 80 7E 00
11.368320	948	131	8	36 46 31 30 39 31 34 39
11.600850	499	170	8	01 00 00 00 00 00 00 00
11.987937	99	300	8	00 00 84 00 00 04 00 00
12.191100	100	308	8	00 FF 00 00 00 00 00 00
12.138710	100	320	8	20 04 00 00 00 00 00 00
11.838204	249	348	8	00 00 00 00 00 00 00 00
12.218800	20	380	8	02 02 00 00 E0 00 7D 0F
12.218820	20	388	2	01 10
12.258190	99	410	8	20 00 00 00 00 00 00 00
11.614087	499	510	8	34 5D 01 3C F0 C4 12 2E
12.388380	100	520	8	00 00 04 00 00 00 00 00
11.377870	1533	670	8	47 31 5A 54 35 33 38 32

Figure 7. Packets received from the vehicle.

6.5. Ignition Cycles

Ignition could get disrupted while driving, causing a sudden stoppage of the car.

7. Communicating with the Wireshark for Reversing CAN Bus

To keep a watch on the activity of CAN, we need a device called OBD-II that could monitor and generate CAN packets. This device will cost around 2000 INR. Open source hardware and software are ideal to use as it is compatible with the majority of software tools. We used Wireshark to capture and alter the packets, and candump from the can-utils suite (Figure 7).[15,16]

Every vehicle has a unique CAN system; therefore, common packet investigation won't work for CAN. As there's so much disturbance on CAN, it's very difficult to sort in an order of every packet.

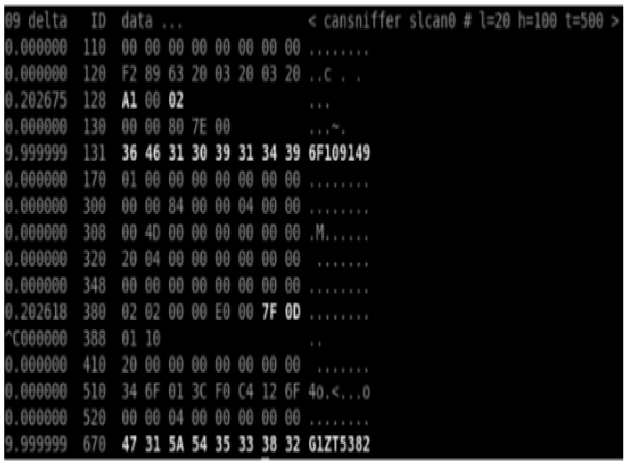
7.1. Wireshark

For networking, we used Wireshark with SocketCAN to capture CAN packets. Both canX and vcanX devices could be listened with Wireshark. If you need to use a slcanX device with Wireshark, one should change the name from slcanX to canX.

If interface renaming doesn't work, then one has to transfer CAN packets from an interface that Wireshark can't read; a single CAN could bridge the two interfaces. To do so, we used the mentioned commands (Figure 8):

$ candump -b vcan0 slcan0

Raw hex bytes are shown because the data section isn't decoded. This happens because Wireshark's decoder is



```
09 delta   ID  data ...                < cansniffer slcan0 # l=20 h=100 t=500 >
0.000000  110  00 00 00 00 00 00 00 00 ........
0.000000  120  F2 89 63 20 03 20 03 20 ..c . .
0.202675  128  A1 00 02                ...
0.000000  130  00 00 80 7E 00          ...~.
9.999999  131  36 46 31 30 39 31 34 39 6F109149
0.000000  170  01 00 00 00 00 00 00 00 ........
0.000000  300  00 00 84 00 00 04 00 00 ........
0.000000  308  00 4D 00 00 00 00 00 00 .M......
0.000000  320  20 04 00 00 00 00 00 00  .......
0.000000  348  00 00 00 00 00 00 00 00 ........
0.202618  380  02 02 00 00 E0 00 7F 0D ........
^C000000  388  01 10                   ..
0.000000  410  20 00 00 00 00 00 00 00  .......
0.000000  510  34 6F 01 3C F0 C4 12 6F 4o.<...o
0.000000  520  00 00 04 00 00 00 00 00 ........
9.999999  670  47 31 5A 54 35 33 38 32 G1ZT5382
```

Figure 8. CAN packets in Wireshark.

not able to deal with ISO-TP or UDS packets but can only handle the basic CAN header.

7.2. Writing to the CAN Bus

Then we write back to the CAN bus the below-mentioned code, which handles the steering wheel angle.

$ openxc-control write –name steering_wheel_angle_value 41.0
$ openxc-control write –bus 2–id 41 –data 0x1234

It is basically called raw CAN hacking. However, one can write an app or embedded graphical interface so that the vehicle could read and react, thus making it the quickest route to own a car for free.

7.3. Hacking OpenXC

After our work of reversing CAN signals, one can frame their own OpenXC firmware. As OpenXC is an API for the car, its work is to read as well as translate information from a car's internal network so that the data could become approachable from most Android apps using the OpenXC library. Compiling[17] our own firmware becomes easy which indicates now we could read or write whatever we want and even write code for the "unsupported" signals. To start an engine, we can create a signal for that and then add it to our own firmware in order to provide a layman interface to give ignition to the car. So, this is the power of open source. Consider a signal that renders speed of the engine. Giving 8-8 will set a basic configuration to return the speed signal of engine. Then we sent RPM data with a 4-byte-long instruction ID 0x1110 starting at the fourth byte.

```
{ “name” : “Intersquad”,
   “buses”: {
     “hs”: {
         “controller”: 1,
        “speed”: 600000
      }
   },
   “instruction”: {
     “0x110”: {
        “name”: “Acceleration”,
        “bus”, “hs”,
        “signal”: {

          “signal_of_engine_speed “: {
             “name”: “engine_speed”,
             “bit_position”: 4,
             “bit_size”: 18

} }
} }
}
```

With the help of OpenXC, our modifications of CAN system are stored in JSON. JSON is used for storing and exchanging data. First of all, we increased acceleration of the car using the above code, thus modifying the bus by framing a JSON with a text editor. In the code, we framed a signal of JSON for a high-speed bus running at 600 kilobytes per second.

JSON can read human-readable text for transmitting data consisting of array data types and attribute value pairs. As soon as we have the JSON, we compiled the above code into a CPP format which again could be compiled into the firmware:

$openxc-generate-firmware-code –message-set/run-bench.json > signal.cpp

With the help of these commands, we recompiled the firmware. If somehow things go wrong and we can't gain access to the CAN bus system, then ECU hacking comes into the picture.[17,18]

8. Conclusion

Cyber security is now the need of hour. Smart cars are the most vulnerable and open to any sort of exploits. One can imagine the situation of being hacked while driving. Even the airbags, brakes and accelerators may not be in one's control on wheel. So, manufacturers need to lay much importance on the CAN bus system by making it more hardware-secured and using secret codes. By finding all possible ways of attack a hacker can perform on the car, we can patch that vulnerability and could save people.

References

1. Currie R. Developments in car hacking. SANS Institute; 2015.
2. Smith C. The car hacker's handbook: a guide for the penetration tester. No Starch Press; 2016.
3. Jafarnejad S. A car hacking experiment: when connectivity meets vulnerability. In: IEEE globecom workshop; 2015. P. 1–6.
4. Zhang Y. Controlling a car through obd injection. In: IEEE 3rd international CONFERENCE on cyber security and cloud computing; 2016. P. 26–9.
5. Martinelli F. Car hacking identification through fuzzy logic algorithms. In: IEEE international conference on fuzzy systems; 2017. P. 1–7.
6. Samara G, Al-Salihy AHW, Sures R. Security analysis of vehicular ad hoc networks. In: Second international conference on network applications, protocols and services; 2010.
7. Van Osch, Michiel, Smolka SA. Finite-state analysis of the CAN bus protocol. In: Proceedings sixth IEEE international symposium on high assurance systems engineering. Special topic: impact of networking; 2001. P. 42–52.
8. Rouf I, Miller R, Mustafa H, Taylor T, Oh S, Xu W, et al. Security and privacy vulnerabilities of in-car wireless networks: a tire pressure monitoring system case study. USENIX Security; 2010. P. 323–38.
9. Kaspersky lab daily. [cited 2019 May 22]. https://en.wikipedia.org/wiki/Kaspersky_Lab.
10. Brief history of car hacking. [cited 2017 Aug 30]. https://smart.gi-de.com/2017/08/brief-history-car-hacking-2010-present/.
11. Studnia I, Nicomette V, Alata E, Deswarte Y, Kaâniche M, Laarouchi Y. Survey on security threats and protection mechanisms in embedded automotive networks. In: 43rd annual IEEE/IFIP conference on dependable systems and networks workshop; 2013. P. 1–12.
12. Ring M. Survey on vehicular attacks-building a vulnerability database. In: IEEE international conference on vehicular electronics and safety; 2015. P. 208–12.
13. Hackingnews. [cited 2017 Feb 03]. https://latesthackingnews.com/2017/02/03/metasploit-now-supports-hacking-cars/.
14. Cui X, Li J. Tools and practices. Secure and trustworthy transportation cyber-physical systems. Singapore: Springer; 2017. P. 143–59.
15. Kroker A, Kroker M. Hacking the future: stories for the flesh-eating 90s. New World Perspectives; 1996. P. 1–146.
16. Miller C, Valasek C. Remote exploitation of an unaltered passenger vehicle. Black Hat USA; 2015. P. 1–91.
17. Cui W. Automatic reverse engineering of input formats. In: Proceedings of the 15th ACM conference on computer and communications security; 2008. P. 391–402.
18. Malekian R. Design and implementation of a wireless OBD II fleet management system. IEEE Sens J. 2016, 17(94):1154–64.