**Digital Forensics in Vehicles**

**Problem:**

Usual applications for digital forensics revolve around everyday computers. Personal computers, hard drives, cloud storage, mobile phones, and daily interactions over a network make up the majority of each consumer’s digital footprint. As the Internet of Things grows however, other everyday objects grow more significant as pieces of evidence, especially when there are no other popular electronics around.

One such source is the automobile. Largely unnoticed until recently, cars and onboard instruments have advanced enough to blur the line between being a mode of transportation and being a device. While some are apprehensive about the implications vehicle-based computers have on personal security and safety, the amount of information available on vehicles today can prove to be incredibly valuable in a complex court dispute. Vehicle accidents, crimes, and other violations are rarely neat and simple for several reasons. Accidents without warning and are hard to avoid, leaving few witnesses and dependable information. In the event that there is no camera footage or any other record of the incident, it may be difficult to assign fault to the drivers involved. The data inside the vehicle itself, on the other hand, offers concrete evidence. There are already tools that exist today to perform some of these actions, such as diagnostic tools used by car repair centers to check statistics like fuel economy

My task was to create a tool that can retrieve data from vehicles specifically with digital forensics in mind. The data should include information that would help identify and track the history of a vehicle, its drivers, and any suspects involved in a case. The tool would be able to communicate with various makes, models, and years of vehicles and generate a report on the information that it was able to retrieve from the vehicle. For this tool, I used Python to interpret serial streams, communicate with the vehicle, and generate the report. In terms of hardware, I had access to three different vehicles over the course of the project and used a commercially available cable to connect my computer to each one.

**Background and Research Performed:**

The initial idea for the project came from a previous car hacking internship project called “VROOM II” that I had left unfinished. VROOM II mainly focused on manipulation of individual instruments like lights, locks, and brakes, but basic similarities exist with this project. The method for connection with each vehicle will be through the Onboard Diagnostic port, or OBD II port. This specific port is extremely useful as it is connected to nearly every telematics system in a car. Moreover, a 1996 standardization law required all cars imported to or manufactured in the United States after that year to have the OBD II port within two feet of the steering wheel. The OBD II cable interprets the serial signals into a stream of hex values.

There were two main targets on the vehicles themselves. The first was the system of electronic control units around the vehicle which would provide everything from live instrument data to saved values. The second, still technically accessed through an electronic control unit, was the event data recorder. The main purpose for accessing the event data recorder was to retrieve longer-term data such as vehicular accident histories. These “black boxes,” according to an article by Robson Forensic, can appear on either airbag control units, the engine control model, or the powertrain control model. In general, they are useful because they record crash event data. The limiting issue though, is that vehicle manufacturers design so much of their vehicles to be proprietary that there are next to no tools available commercially. Manufactures grip tightly on their access methods, and the only crash retrieval system is created by Bosch, mostly for police and professional crash reconstructionists. Still, even the Bosch tool is severely limited as it is only compatible with the newer models of just a few car makes.

Luckily, communication with the vehicles is still feasible without a black box tool. In order to establish a connection with any diagnostic port, it is first necessary to understand the different protocols used. Over the In the VROOM II project, the main protocol used was the standard ISO 15765-4 Controller Area Network (CAN) Protocol. This protocol communicates through Parameter IDs (PIDs) in the form of hex values. CAN has been the industry standard for the past decade and is the most common protocol in the US. For this project, two out of three vehicles used the CAN protocol. The third vehicle used an older ISO 9141-2 protocol which similarly communicates through PIDs, but, as stated by the Volkswagen datasheet, the main differences are in the baud rate and an initialization handshake.

**Resources:**

Articles:

National Highway Traffic Safety Administration. “USDOT Releases 2016 Fatal Traffic Crash Data.”

National Highway Traffic Safety Administration. *Event Data Recorders*.

United States Department of Justice, Federal Bureau of Investigation. *Crime in the United States, 2016*.

The Car Hacker’s Handbook by Craig Smith

Vin Cloning: How Thieves can Steal Your Car’s Identity from ABC News

Tools/Code Base:

VROOM II Project by Ziwei Peng and Connor Eckert

ELM327 OBD to RS232 Interpreter Datasheet

Volkswagen’s K-Line Communication Description

ISO 15765-4 CAN Datasheet

OBD II PIDs, from obdcon.sourceforge.net

CoolTerm Serial Terminal by Roger Meier

**Assumptions and Design Decisions:**

There were a few design choices for the project. The vehicles I would be using should diverse in make, model, and year (ultimately depending on availability). Additionally, the vehicles could not be too old. Specifically, the model should be after 1996, when OBD ports became mandatory in the US, and preferably closer to 2010, when EDRs were mandated by the National Highway Traffic Safety Administration. Due to availability, none of the vehicles had been previously damaged or modified. This preserves the original factory state of the vehicle.

**Status of the Project:**

The vehicles I used were a 2002 Toyota Camry Le, a 2010 Honda Civic Sedan, and a 2011 Subaru Outback 2.5i. The two CAN protocol vehicles were the Honda Civic and the Subaru Outback. For each vehicle, I gathered raw CAN bus streams, responses for VIN numbers, and various other PID available codes. In addition to the standard PID codes, I queried each vehicle for manufacturer-specific codes.

To communicate with the OBD cable, I created a program through the PySerial library. Each vehicle had specific baud rates, which were determined through mostly through trial and error through initial connection attempts in CoolTerm. I found that the CAN vehicles used an 115200 baud rate while the ISO 9141-2 vehicle needed an initial 5 baud init before a transition to the higher baud rate. Upon sending a request and receiving a Hex capture, it would be necessary to decipher the response as they were specific for each vehicle. The four values checked across all vehicles were the Vehicle ID Number (VIN), the Calibration ID, the Calibration Verification Number, and the Engine RPM.

Subaru:

|  |  |  |
| --- | --- | --- |
| Item | Hex Capture | Value |
| VIN | 0: 49 02 01 **34 53 34**  1: **42 52 42 43 43 38 42**  2: **33 33 34 35 39 37 30** | 4S4BRBCC8B3345970 |
| RPM | 41 0C **0B A9** | 2985 / 4 = 746 |
| Calibration ID | 0: 49 04 01 **59** **36 46**  0: 49 04 01 **45 45 35**  1: **49 39 31 30 54** 00 00  1: **33 45 31 36 30** 00 00  2: 00 00 00 00 00 00 00  2: 00 00 00 00 00 00 0 | Y6F3E160  EE5I910T  \*Note 0,0/1,1/2,2 stacking |
| Calibration Verification Number | 0: 49 06 01 **34 36 43**  1: **41 44 42 35 31** 00 | 46CADB51 |

Honda:

|  |  |  |
| --- | --- | --- |
| Item | Hex Capture | Value |
| VIN | 18 DA F1 11 10 14 49 02 01 **32 48 47**  18 DA F1 11 21 **46 41 31 46 38 36 41**  18 DA F1 11 **22 48 35 34 39 36 34 36** | 2HGFA1F86AH549646 |
| RPM | 18 DA F1 1D 04 41 0C **0B 18** 55 55 55 | 2840 / 4 = 710 |
| Calibration ID | 18 DA F1 1D 10 13 49 04 01 **33 37 38**  18 DA F1 11 10 13 49 04 01 33 37 38  18 DA F1 1D 21 **30 36 2D 52 4E 41 2D**  18 DA F1 1D 22 **41 36 34 30** 00 00 55  18 DA F1 11 21 30 35 2D 52 4E 41 2D  18 DA F1 11 22 41 37 33 30 00 00 55 | 37806-RNA-A640 |
| Calibration Verification Number | 18 DA F1 11 10 13 49 06 01 **35 44 33**  18 DA F1 1D 21 **32 35 34 42 41 20** 00 | 5D3254BA |

Toyota:

|  |  |  |
| --- | --- | --- |
| Item | Hex Capture | Value |
| VIN | NO DATA | NO DATA AVAILABLE |
| RPMs | 41 0C 0B AB | 2987 / 4 = 747 |
| Calibration ID | 49 04 01 **33 33 33 30**  49 04 02 **32 30 30 30**  49 04 03 00 00 00 00  49 04 04 00 00 00 00  49 04 05 **35 33 33 30**  49 04 06 **32 30 30 30**  49 04 07 00 00 00 00  49 04 08 00 00 00 00 | 33302000  53302000 |
| Calibration Verification Number | 49 06 01 **36 43 34 44**  49 06 02 **37 32 31 32**  49 06 03 00 00 00 00  49 06 04 00 00 00 00 | 6C4D7212 |

The significance of the VIN lies in the fact that it is a unique identification code for the vehicle. In application, only one vehicle with the VIN should exist, and all other occurrences signify fraudulence. The PID is 09 02, which is displayed as a header. For example, the Subaru shows 49 02 signifying the PID, with a 01 to mark the beginning of the VIN. Depending on the car, transmission which take multiple lines may append the header to the beginning of each. Once the end of the value is reached, the ECU will pad the remaining space with zeroes or some sort of stop bit (likely 0x55).

The headers for the RPM, Calibration ID, and Calibration Verification Number are similar, with their PIDs being 01 0C, 09 04, and 09 06. While the purpose of the RPM was to confirm live feedback from the vehicles, the other two values provide more forensics significance. The Calibration ID functions to identify the software calibration of the vehicle and is unique to the model. The Calibration Verification Number verifies the integrity of the vehicle software using a checksum. The combination of these two values can identify any major changes made to the vehicle. This meets the first goal of the tool.

The second goal of the tool was to be able to access saved crash data from the event data recorder. Unfortunately, I was unable to interface with the EDR through the OBD II port, but I was able to find responses from nonstandard PIDs. As stated in the PID list from the ObdCon site, there are nine standard modes. Other PIDs vary as they are proprietary to the vehicle manufacturer and model. Two known modes which are popular with manufacturers are 21 and 22, but without the necessary identification of those codes, the hex outputs mean little. In an attempt to distinguish any possible unique responses, I implemented a dump of the raw hex stream from the vehicle CAN Bus in hopes of comparing samples with and without certain activated instruments to distinguish the header format, but found little progress.

**Practical Application:**

A 2015 article from ABC News described a new form of car theft. According to the article, of the 700,000 car thefts every year in the US, many are disassembled and sold for parts or restored by other parts, but a new method called VIN Cloning is trending and proving to be even more profitable. The “cloning” is entirely mechanical, with the process consisting of replacing the VIN plate under the windshield of the vehicle with a counterfeit one. VIN Cloning has several collateral effects. It masks vehicle theft, but it also allows for the illegal sale of stolen cars to people who would otherwise not be involved. Additionally, since at VIN numbers are unique, counterfeit plates often match real ones, creating a shared responsibility of the VIN record between two drivers. In the even that one has a poor driving history or several legal complications, the innocent driver will feel equal effects.

In order to identify the real vehicle, using the forensics tool to retrieve the saved VIN would be a quick solution. If saved data on the vehicle was tampered with or if parts were replaced, the Calibration Verification Number would change, consequently showing an invalid checksum. The victim could then use this information as evidence against the seller or other owner of the vehicle. While this would not necessarily help identify the thief, it would protect the victim from misdirected fault.

**Program Execution:**

1. Locate the OBD port on the target vehicle. Plug the OBD II cable in to both the port and the computer
2. Verify the protocol used by the vehicle. This may not be in the user manual, but the information should be available online through the manufacturer.
3. On Windows, open Device Manager to find which COM port the ODB II cable was assigned. On Linux, open a Terminal windows and issue the $ ls /dev/tty\* command to identify the correct port.
4. Change the values in CarTool.py to match the port and vehicle (the protocol).
5. Attempt to connect to the port over a terminal window by running Python, importing serial, then setting up a connection with various baud rates. A valid baud rate should return ASCII characters rather than garbled values:

con = serial.Serial(port)

con.baudrate = 115200

1. Set the monitorAll flag to True or False.
2. Run the file with the command:

$ python3 CarTool.py # on Linux

>> py -3 CarTool.py

Works Cited

Becker, Steven. “Interpreting Vehicle EDR (Black Box) Data & Recognizing Errors.” *Robson Forensic*, 11 November 2011, www.robsonforensic.com/articles/interpreting-vehicle-edr-black-box-data-recognizing-errors-expert-article. Accessed 30 April 2018.

Eckert, Connor, and Peng, Ziwei. “VROOM Project.” ARDEC Improvised Capabilities (2017). Print.

“ELM327 OBD to RS232 Interpreter.” *ELM Electronics*. ELM Electronics Inc., 2013, www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DSF.pdf. Accessed 20 April 2018.

“ISO 15765-4: Road Vehicles – Diagnostics on Controller Area Networks (CAN).” *ISO*. ISO Copyright Office, 2005,

“K-Line Communication Description.” *Volkswagen Group of America*, www.alflash.com.ua/OBDII/K-line\_communication\_description.pdf. Accessed 20 April 2018.

Meier, Roger. “CoolTerm.” Computer Software, Roger Meier’s Freeware, 1.5.0, 07 Jan. 2018, Web, www.freeware.the-meiers.org. Accessed 30 April 2018.

National Highway Traffic Safety Administration. *Event Data Recorders*. Retrieved 30 Apr. 2018 from https://www.law.cornell.edu/cfr/text/49/part-563.

National Highway Traffic Safety Administration. “USDOT Releases 2016 Fatal Traffic Crash Data,” *NHTSA Press Releases*, 06 Oct. 2017, www.nhtsa.gov/press-releases/usdot-releases-2016-fatal-traffic-crash-data. Accessed 30 April 2018.

“OBD-II PIDs.” *OBD-II Resource*. OBD-II Resource, 2008, www.obdcon.sourceforge.net/2010/06/obd-ii-pids/. Accessed 30 April 2018.

Smith, Craig. “The Car Hacker’s Handbook.” *Open Garages.* San Francisco, No Starch Press, 2016, www.opengarages.org/handbook/ebook/. Accessed 30 April 2018.

United States Department of Justice, Federal Bureau of Investigation. (September 2016). *Crime in the United States, 2016.* Retrieved 30 Apr. 2018 from www.ucr.fbi.gov/crime-in-the-u.s/2016/crime-in-the-u.s.-2016/topic-pages/tables/table-10.

Wagschal, Gerry, Gio Benitez, & Khoury, Lisa. “VIN Cloning: How Thieves can Steal Your Car’s Identity.” *ABC News*, 24 Jan. 2015, www.abcnews.go.com/US/vin-cloning-thieves-masking-car-thefts/story?id=28401709. Accessed 30 April 2018.