POSTER: PenJ1939: An Interactive Framework for Design and Dissemination of Exploits for Commercial Vehicles

Subhojeet Mukherjee
Colorado State University
Subhojeet.Mukherjee@colostate.edu

Noah Cain
Colorado State University
noahcain@rams.colostate.edu

Jacob Walker
Colorado State University
jksctwkr@rams.colostate.edu

David White
Colorado State University
dwhite54@colostate.edu

Indrajit Ray
Colorado State University
Indrajit.Ray@colostate.edu

Indrakshi Ray
Colorado State University
Indrakshi.Ray@colostate.edu

ABSTRACT

Vehicle security has been receiving a lot of attention from both the black hat and white hat community of late. Research in this area has already led to the fabrication of different attacks, of which some have been shown to have potentially grave consequences. Vehicle vendors and original equipment manufacturers (OEM)s are thus presented with the additional responsibility of ensuring in-vehicular communication level security. In this poster paper, we present a framework, which allows any individual to write, test, and store exploit scripts which could then be run by any interested party on in-vehicular networks of commercial vehicles like trucks and buses.

KEYWORDS

CAN; J1939; Exploit; Script; Development; Interactive; Download

1 INTRODUCTION

Back in the 70's vehicles were driven purely by physical and mechanical interactions. Today, much of the human-mechanical interaction is mediated through embedded devices also referred to as Electronic Control Units (ECU)s. These devices are often intelligent and can ensure smooth driving, safety and comfort. The computerization of vehicles has, however, led to the advent of newer exploits which target these embedded devices and the underlying network they communicate with. Hackers and security professionals have shown that embedded networks in passenger cars can be compromised to cause large scale damage [2]. These networks primarily rely on the Controller Area Network (CAN) protocol for message exchange. While CAN ensures reliable message delivery across ECUs, it does not specify how messages on the network are utilized by ECUs. While passenger vehicles mostly use proprietary specifications to make such decisions, commercial vehicles use a common set of standards (SAE J1939 [1]) specified by SAE International. This makes commercial vehicles increasingly more susceptible to attacks which target the ubiquitously used SAE J1939 protocol stack.

CCS'17, Oct. 30–Nov. 3, 2017, Dallas, TX, USA.
 ISBN 978-1-4503-4946-8/17/10.
DOI: https://doi.org/10.1145/3133956.3138844

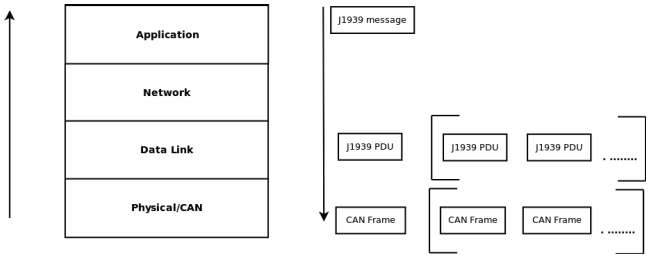

Figure 1: SAE J1939 Protocol Stack

Recently, security researchers [5, 6] have shown that attacks can be launched at different layers of the SAE J1939 protocol stack.

Since commercial vehicles expose almost the same set of attack surfaces as passenger vehicles, these attacks can be realized using similar tool-sets and can have severe consequences. In this poster paper, we present a framework which can be used to develop, test (on a built-in testbed setup) and store exploits written by any individual with J1939/CAN or relevant security related experience. These scripts can then be downloaded by a plethora of interested individuals including industry professionals, mechanics and garage technicians, and other tech-savvy users. Eventually, we aim to create an easily accessible framework which can benefit the heavy vehicle security community by making exploits accessible readily and expediently, thereby accelerating the process of penetration testing.

2 BACKGROUND

CAN [4] is an arbitration-based protocol that facilitates highly reliable communication over a multi-master broadcast serial bus. The SAE J1939 [1] protocol runs on top of CAN, i.e. it utilizes the physical communication standards exposed by the CAN protocol. The J1939 protocol stack is organized based on the seven layer OSI networking model. Currently, SAE standards are specified for 4 of the 7 layers[1]. These layers are shown in Fig. 1. A J1939 message is composed by an ECU at the *Application* layer and transmitted as a sequence of bits at the *Physical*/CAN layer after being bundled into fixed size *Protocol Data Units* (PDU)s. A typical J1939 PDU consists of a 29 bit *Identifier Field* and a 64 bit *Data Field*. A single J1939 message is uniquely identified using a *Parameter Group Number* or PGN. For example, messages related to torque or speed control

[1] http://www.sae.org/standardsdev/groundvehicle/j1939a.htm

correspond to PGN 0 (0000_{16}). Information required to generate a PGN for a message is embedded in the *Identifier* field. Vehicle specific parameters are embedded within the *Data* field. When an ECU receives a J1939 message, it first obtains the PGN from the identifier field. It then refers to the SAE standards [1] to obtain a set of parameter identifiers known as *Suspect Parameter Numbers* (SPNs) assigned to each PGN. Each SPN is assigned to a set of attributes (*starting position* , *length*, *resolution*, *offset* and *name*) that can be used to interpret the contents of the *Data* Field. As an hypothetical example, in the following bit sequence (representing the contents on the *Data* field), $00011100\ldots_2$, SPN 789, *starting at bit position* 2 from left and extending for 4 bit of *length*, could signify percentage of torque applied when the decimal equivalent of the sequence of 4 bits is multiplied by the resolution 0.125% and added to the offset 0 i.e $0011_2{}^*0.125_{10} \rightarrow 3_{10} * 0.125_{10}$.

3 PENJ1939 FEATURES

PenJ1939 is generally designed as an interactive framework which can be used by professionals to design and access existing attacks on J1939 based networks. However, designing attacks is often accompanied by other helpful and often necessary activities like monitoring bus traffic, interpreting J1939 messages, etc. With PenJ1939, we make an effort to integrate such features into one framework so as to ease the process of development and testing of attacks. Mentioned below are the salient features of PenJ1939:-

- **Attack Scripting**: Currently PenJ1939 allows attacks to be scripted in the Python scripting language. We provide a development interface, which could be used to write python code. As scripts could sometimes involve threaded executions, we allow users to split the scripting interface and develop/execute multiple programs in parallel. Users are also allowed to *upload* previously written and tested scripts on the PenJ1939 database.
- **Script Testing**: Each test script is executed on embedded controllers connected to a physical testbed. Test outputs and associated errors or warnings are presented to the user in order to aid in development. To test their scripts, users are required to obtain access to dedicated node controllers which act as gateways to the CAN network.
- **Library Access**: Script developers are also granted access to modules developed previously. This is done to avoid redundancy and speed up the process of exploit writing. Currently, all modules included as a part of a script are stored in the *Module* database.
- **Traffic Sniffing**: We provide restricted access (via a *TestBed Manger*) to the PenJ1939 experiment testbed. Script writers or users can test scripts by running their code and observing relevant outputs in hexadecimal format. Obtained outputs can also be interpreted at runtime using an inbuilt J1939 interpreter. This allows users to see actual vehicle parameters, some of which might be intended targets of the attack.
- **Script Verification**: Scripts being written on or uploaded to the PenJ1939 framework are passed through a final verification phase before being committed to the database. Albeit some scripts might not be supported for execution on our testbed. We still archive such scripts but notify end-users about the status. Verifying a script only ensures it executes without errors or warnings. Verification does not ensure the eventual correctness of the attack, i.e. whether it was successful in achieving the final goal.
- **Script Annotation**: Before exploits are committed to the PenJ1939 database, the exploit developers are encouraged to annotate their scripts with metadata to be used later as search fields. Currently, we support the following fields: a simple *Documentation* of what the exploit does, five default *Tags*: *J1939 layer*, *type of attack*, *affected ECUs*, *PGNs used* and *SPNs used* and *Pre-Requisites* like *ARM operating systems for node controllers*, *necessary ECUs connected to the bus*, *bus baud-rate* etc.
- **Regex-based Filtering**: PenJ1939 exposes a number of regular expression-based filtering interfaces. Script writers can filter traffic sniffed off the J1939 network. PenJ1939 supports message filtering based on *PGNs*, *SPNs*, *SPN interpreted values* and regex-based filtering on raw hexadecimal expressions for both the *ID* and *Data* fields. Modules and scripts can be filtered on the contents of their documentation, or associated tags and pre-requisites using regular expressions.
- **Downloading Scripts**: End-users can download previously uploaded scripts by either browsing the script directories or executing search regular expression-based queries. Before downloading a script, the user can verify the script's execution to see if it achieves the desired results, and the user can also access information about the developer of the script.

4 ARCHITECTURE AND COMPONENT INTERACTION

Fig. 2 shows the interactions between the architectural components of PenJ1939. The thick solid boxes represent web-pages accessible to the user. In order to write or download exploits, users need to be logged into the PenJ1939 system. The *Login-Manager* is responsible for verifying user credentials and/or signing up new users. Users can also take a brief tutorial of the system before creating user accounts. Both scripts written and uploaded on the system are sent to the *TestBed Manager* for final verification. The status of the script is accordingly updated to "Yes" if it had an error free execution or "N/A" if the testbed platform does not have adequate resources to run the script. Failed scripts are rejected and the user is notified accordingly. Once a script is ready to be uploaded to the *DB-Manager*, the *Scripting-Manager* asks the user to annotate scripts and modules by populating metadata search fields. The *DB-Manager* is also responsible for executing regular expression queries on search fields and return relevant scripts and modules. Users can select modules from the module browser, and read the module's documentation before using such modules. Once scripts are ready to be tested, users can request nodes. The *TestBed-Manager* is responsible for returning node handles, if a free node is found. The testbed is modeled on a previously published work from our group [3]. Currently, the *ECU Layer* has 3 ECUs (Engine Controller, Retarded, Brake Controller) attached to it. The *Sensor and Simulator Layer* is currently disabled. The testbed thus generates traffic as obtainable from a standalone truck. Two BeagleBone black devices, acting as *Node Controllers* are attached to the CAN network. Users can reserve these nodes to execute their scripts via the *TestBed-Manager*.

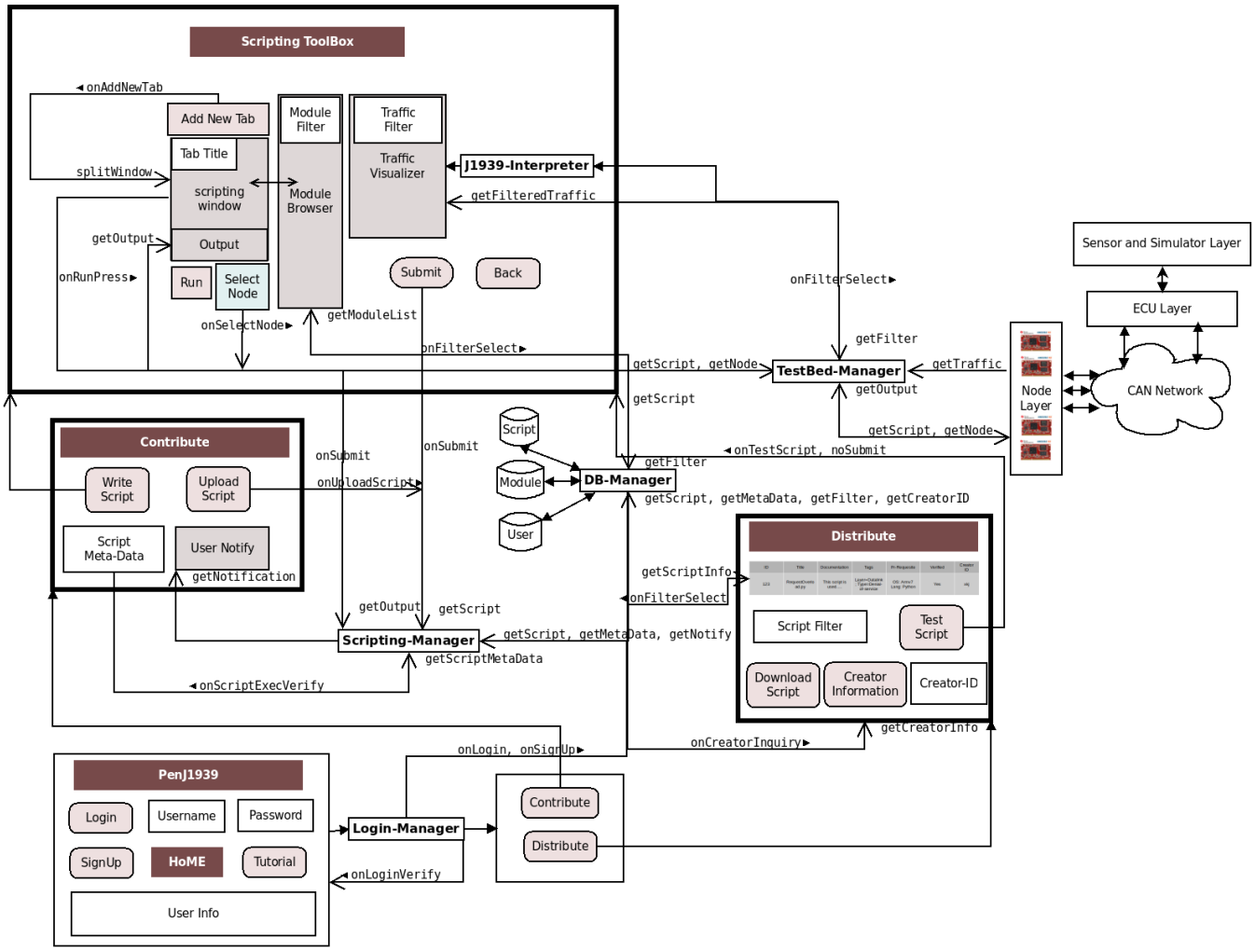

Figure 2: PenJ1939 Components and their Interactions

5 CURRENT IMPLEMENTATION AND FUTURE WORK

We have started the implementation process for PenJ1939. As mentioned earlier the testbed was a part of previous research and hence was set up at the beginning. All managers (Fig. 2) and associated databases have been set up. A J1939 decoder was designed as a part of our previous work [3]. Core functional modules of that the decoder were integrated into the PenJ1939 system in order to design the *J1939-Interpreter*. We are currently in the process of designing the web-based front-ends.

In future, we aim to improve some security critical aspects of our system. In particular, we believe that by integrating proper authentication and authorization mechanisms it may be possible for security professionals and researchers to alert specific OEMs about security issues in their J1939 implementations. We also aim to add additional features like authorized editing of code and patch management as a vision for of making PenJ1939 more usable.

ACKNOWLEDGEMENT

The work was supported in part by NSF under award numbers CNS 1619641 and CNS 1715458.

REFERENCES

[1] 2013. Serial Control and Communications Heavy Duty Vehicle Network - Top Level Document. (2013). http://standards.sae.org/j1939_201308

[2] C. Miller and C. Valasek. 2014. A Survey of Remote Automotive Attack Surfaces. *Black Hat USA* 2014 (2014).

[3] J. Daily, R. Gamble, S. Moffitt, C. Raines, P. Harris, J. Miran, I. Ray, S. Mukherjee, H. Shirazi, and J. Johnson. 2016. Towards a Cyber Assurance Testbed for Heavy Vehicle Electronic Controls. *SAE International Journal of Commercial Vehicles* 9, 2 (2016), 339–349.

[4] R. Bosch. 1991. CAN specification version 2.0. *Rober Bosch GmbH, Postfach* 300240 (1991).

[5] S. Mukherjee, H. Shirazi, I. Ray, J. Daily and R. Gamble. 2016. Practical DoS Attacks on Embedded Networks in Commercial Vehicles. In *Proceedings of 12th International Conference on Information Systems Security*. 23–42.

[6] Y. Burakova, B. Hass, L. Millar, A. Weimerskirch. 2016. Truck Hacking: An Experimental Analysis of the SAE J1939 Standard. In *Proceedings of 10th USENIX Conference on Offensive Technologies*. 211–220.