Logboat – A Simulation Framework Enabling CAN Security Assessments

Sven Plaga
Fraunhofer AISEC
Garching b. Munich, Germany
sven.plaga@aisec.fraunhofer.de

Stefan Tatschner
Fraunhofer AISEC
Garching b. Munich, Germany
stefan.tatschner@aisec.fraunhofer.de

Thomas Newe
Electronic and Computer Engineering
University of Limerick, Ireland
thomas.newe@ul.ie

***Abstract*—Traditionally, fieldbus networks are operated in closed environments, where all communication nodes are assumed to be trustworthy. Therefore, the corresponding standards do not consider any security requirements. New technology trends, such as the upcoming Internet of Things (IoT), demand an interconnection between all components of an industrial infrastructure. As a consequence of this, there is a need for tools enabling security assessments and the simulation of protocol-based security improvements. In this paper we introduce Logboat, a flexible Python and Linux based simulation framework for security assessments on Controller Area Network (CAN) networks and make some architectural and technology-selection proposals. Subsequently, the different modules of Logboat and their capabilities are explained and a use case scenario is presented. The paper concludes with an outlook on upcoming research activities on CAN bus security where the presented framework can be of help.**

I. Introduction

As illustrated in Figure 1, present industrial automation systems are composed of several levels aligned in a hierarchically structure.

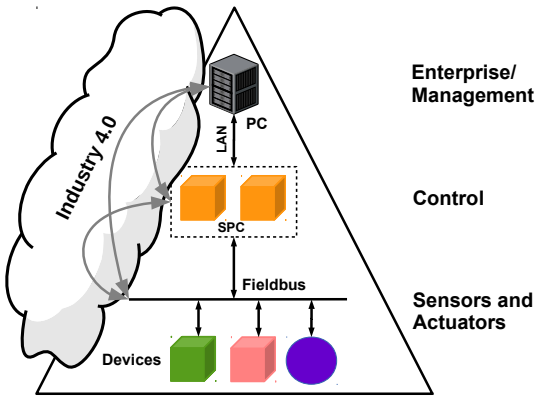

Figure 1. Traditional Automation Topology and Industry 4.0

The Enterprise/Management level consists of servers running administrative applications which are also partially connected to office networks. These systems have an Internet Protocol (IP) based interconnection. IP protocols are also used to obtain data from the underlying control layer where the shop-floor automation is controlled by specialized embedded devices like Stored-Program-Control (SPC) units. Using a fielbus protocol, the SPC units are connected to the lowermost layer, where they get sensor readings and communicate commands to the respective actuators.

The upcoming paradigm of ubiquitous Internet of Things, however, requires a radical breakup of the traditional hierarchical structures, replacing them by a new intermeshed structure. Offering new opportunities, like shorter time-to-market or low-priced small production quantities, the new structures will enable new business areas. Often seen as the next industrial revolution, this trend is also often referred to as Industry 4.0 [1].

Flattening the traditional industrial topology, there is little space to enforce proven security paradigms like compartmentalisation or perimeter oriented protection. As the new applications will require direct, unhindered access to all levels of a shop-floor, it is likely that also low-level fieldbus nodes will be exposed to Wide Area Networks (WANs).

Fieldbus protocols, however, have not been designed to fulfill any security objectives. As there are no authenticity schemes, bus nodes blindly trust every communication peer. Furthermore, the transmissions are not encrypted. Consequently, attackers might be able to fake nodes, perform man-in-the-middle (MITM) attacks or eavesdrop the communication.

Vulnerabilities are also found in the protocol stacks of the firmware layer. Here, programmers often do not expect to be confronted with malicious attacks, like communication nodes exploiting missing length checks or unhandled protocol anomalies. These might be used by an attacker for remote code execution or Denial of Service (DOS).

Having safety and real-time constraints as the only relevant requirements, a broad range of commercial off-the-shelf (COTS) diagnosis and simulation tools are neither enabled for fieldbus security assessments nor do they offer any cryptographic extensions.

II. Security Assessments for CAN

The Controller Area Network (CAN) [2] was originally developed for applications within the automotive domain and later on adopted for other domains, e.g. aerospace, avionics, or industrial shop-floors. Being a proven and widely used protocol, an upgraded version was released in 2012 with the CAN with Flexible Data-Rate (CAN FD) standard [3].

Considering the application areas, its importance, and future prospects, CAN is seen as a good starting point for researching security of established fieldbus protocols.

As many security research publications pointed out [4], fieldbus security assessments require a protocol-specific set of building blocks. Ideally, these are implemented as modules and are part of an easy to use and open framework. By combining these blocks, further functionality is achieved.

For an optimal framework the following building blocks are proposed:

- **Capturing:** Acquire data from the bus and provide some functionality to store it in a structured format.
- **Injection:** Act as a communication peer and write data on the bus.
- **Filtering:** Filter the captured messages in order to find and extract specific information.
- **Protocol Dissection:** Recognize, follow and dissect higher level transport protocols.
- **Cryptography:** In order to test novel cryptographic protocol extensions [5], cryptographic primitives need to be available. For key-exchange operations, asymmetric ciphers must be present as well as symmetric ciphers for data stream encryption. To ensure integrity and authenticity, cryptographic signature schemes are needed to be available.
- **Fuzzing:** For testing the impact of malformed datagrams, a fuzzing module provides arbitrary datagrams. These can be used by other modules of the framework.
- **Virtualization:** Besides connecting a pysical bus, the modeling of more complex communication scenarios requires virtualized nodes and virtual bus segments. For the implementation of real use cases, the virtualized nodes must be able to communicate with nodes on a physical bus.

The evaluation of market-leading COTS CAN toolkits found that these lack the required high degree of flexibility for the outlined building blocks. This is due to their proprietary character as they are often based on Microsoft Windows. Taking the huge programming effort into consideration, re-implementing missing base functions of the investigated COTS tools has been assumed to be unreasonable.

Consequently, the field of open-source software was investigated next. The results showed that the Linux Kernel offers a stable support for CAN and also CAN FD. Among several kernel modules, the SocketCAN project, initially developed as a PhD project [6], was selected as a baseline for the desired framework. Having SocketCAN and the reliable Linux kernel at the basis, the framework implementation could focus on the basic functionalities. Contrary to proprietary COTS tools, the open-source model of the Linux kernel allows the implementation of driver adjustments whenever needed.

For the framework, the popular Python programming language, in its reference implementation CPython, has been selected. Providing a huge set of default libraries, Python enables a rapid prototyping approach. Timing critical components, however, can be written in C or Assembler (ASM) as an extension to the Python interpreter.

III. Basic Concept

SocketCAN is part of the Linux kernel since April 16, 2008 (Linux v2.6.25) and provides a set of open-source CAN drivers and a networking stack. The Linux kernel includes device drivers for all major CAN chipsets used in various architectures and bus types. The Application Programming Interface (API) of SocketCAN is hardware independent, socket-based and offers support for multiple users [7].

Figure 2 shows the architecture of the SocketCAN kernel module. It adds a new protocol family `PF_CAN` to the Linux network stack including the `CAN_RAW` and `CAN_BCM` protocols. Additionally, it provides virtual network devices which can be used for testing or virtualization purposes. Since the abstraction level of SocketCAN is high, virtual network devices can be replaced with real CAN devices transparently.

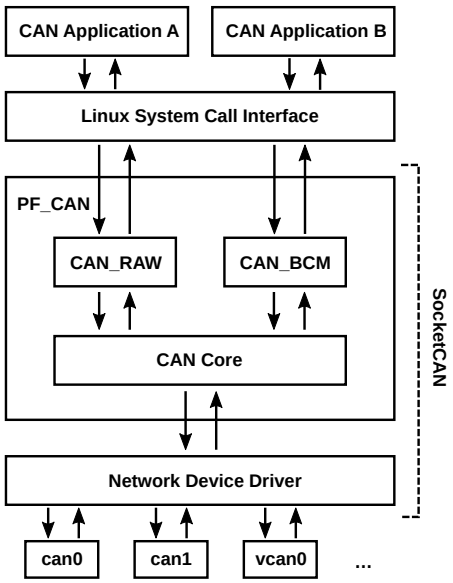

Figure 2. Architecture of the SocketCAN Stack [8]

SocketCAN integrates itself seamlessly into the Linux networking stack and adds an abstraction layer with a lot of functionality. Additionally, it provides a powerful and well known socket programming interface. Sophisticated queuing of CAN frames is handled by the underlying network stack, which also provides an API for device drivers and transport protocol modules.

Since the Linux kernel is implemented in C, the SocketCAN API is only available at the C level. Due to its basic and low level design, the C programming language is inappropriate for a testing or evaluation platform. The proposed solution is an abstraction layer built on top of SocketCAN and written in the Python programming language. The purpose of this additional abstraction layer is wrapping the whole functionality of SocketCAN and providing a scripting interface that exposes SocketCAN and makes it available to the Python ecosystem. The Python ecosystem supplies versatile libraries and tools for scientific data analysis, which can be adapted to use Logboat in order to dissect CAN data.

IV. Implementation

Basically, Logboat is designed as a comprehensive Python interface to SocketCAN. In order to achieve efficient resource utilization, no complex computations or scheduling schemes are implemented in Python. The main goal is to design a simple but powerful pythonic API, which is easy to understand and extendable by well established Python libraries.

A. High Level Description

The Linux kernel deals with CAN frames in a high level manner. SocketCAN processes CAN packets as special data types that include a particular CAN Identifier (ID) and the payload data. Low level components, e.g. routing or packet scheduling, are processed by the SocketCAN kernel module. Further CAN protocol parts, such as calculating Cyclic Redundancy Checks (CRCs) or responding with acknowledgement bits, are entirely handled by the underlying CAN device.

Figure 3 outlines the architecture of Logboat. The Python interpreter runs as a userspace process and compiles the Python code of a Logboat based application transparently into bytecode. Subsequently, the generated bytecode is interpreted by CPython in an evaluation loop.

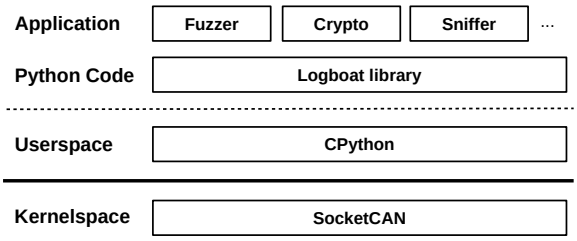

Figure 3. Architecture of Logboat; located above SocketCAN

Since the Python reference implementation CPython is written in C, it is is able to fully utilize the Linux System Call Interface (SCI). A system call is a controlled entry point into the Linux kernel which allows a particular process to request that the kernel executes some action on the process's behalf. To make use of a certain system call from a high level programming language, the Python data structures have to be converted to the corresponding C data types. These data types are represented by special Python bytestrings which are dependent on the used Central Processing Unit (CPU) architecture.

B. Python Classes

Logboat adds an object oriented abstraction layer on top of the SocketCAN data structures. Figure 4 shows three Python classes which are mainly used to communicate with the CAN bus through the Logboat software stack:

- **Bus**: Represents a bus connection and provides methods to communicate with the CAN bus.
- **Message**: A high level representation of a CAN message that includes methods to serialize itself into a bytestring.
- **Filter**: Optional receive filters which are used to apply further configuration to the particular connection in order to reduce ressource allocation.

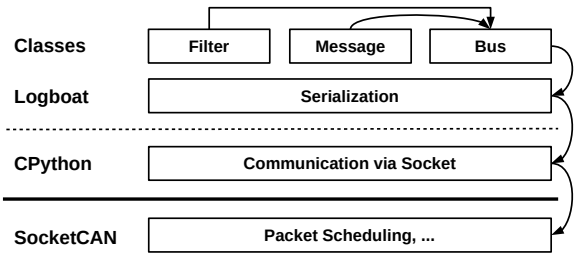

Figure 4. Packet Flow within the Logboat Stack

Sending CAN frames involves several different steps. First, an appropriate Message object with the desired CAN ID and payload has to be allocated. Second, in order to establish a connection a Bus object must be instantiated. The Bus object interacts with the SCI by creating an appropriate `CAN_RAW` socket, applying several socket options and optional receive filters. Finally, the Message is passed to the Bus object that executes the `send(2)` system call. Any further steps are then handled by the software stack and the operating system automatically.

Passing Messages to Bus objects forwards them to subjacent layers of the software stack. The Logboat library serializes the Message objects into bytestrings. They are written to the socket created by the Bus class. Finally, the operating system's network stack schedules the packets appropriately and forwards them to the CAN device. Receiving CAN messages requires the execution of the steps described above in reverse by replacing the `send(2)` with the `recv(2)` system call.

Additionally, Logboat applications can be triggered by certain events in response to CAN messages. This feature is provided by the underlying SocketCAN module by the so called `CAN_BCM` protocol (see Figure 2). `CAN_BCM` sockets transmit data to userspace, depending on configured events, like flipping a bit in a received message.

V. Use Case Scenario

An example application for Logboat is discussed in the remainder of this section. The presented use case is a penetration test scenario, where an electronic Device under Test (DUT) is attacked on its different system layers. By attacking hardware, software and communication subsystems, vulnerabilities are detected and investigated.

Unresolved, these can be found and exploited by attackers to escalate their privileges, gain unauthorized access or extract critical data. Therefore, penetration tests are important for manufacturers who want a security-level analysis of their product but also users could be interested in the results.

Figure 5 shows Logboat as part of a gray box testbed, where the manufacturer of the DUT provided some background information on the CAN messages, speeding up the overall test procedure. The setup

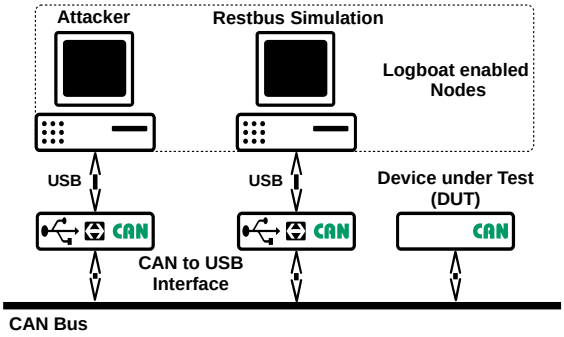

Figure 5. Logboat Penetration Test Scenario

consists of two nodes running Logboat and the DUT itself.

The first Logboat node implements the simulation of messages of physically non-present CAN nodes. Having these messages present on the bus is required for the DUT to operate in its normal state. The simulation is implemented using event triggered `CAN_BCM` sockets which are accessed through the Logboat framework.

The second Logboat node represents the attacker who has gained access to the bus. In this configuration, Logboat is used for eavesdropping and the injection of falsified CAN messages. Evaluating the DUT's reaction to undefined frames, Logboat makes use of fuzzing engines like boofuzz [9].

Testing a DUT without the need of assembling a complete orginal CAN infrastructure, is one benefit of Logboat. Another one is the flexibility of the Logboat framework also enabling the creation of complex attack scenarios.

VI. Outlook

The Logboat framework has reached a stable development state. A future use case scenario is a comprehensive research on cryptographic CAN bus security schemes.

In the current use case scenarios, real-time constrains were not considered. As real-time requirements cannot be ruled out for future applications, another considered enhancement is the usage of kernel based real-time extensions by the Logboat framework. Here, the Real-Time Linux (RTL) project introduced by the Linux Foundation in October 2015 [10] seems to be a promising approach.

Logboat is layered on top of the Linux kernel and tightly coupled to its network stack. Therefore, it is expected that a RTL enabled system will receive increased CAN performance and responsiveness. Thanks to the flexible layered Logboat architecture, low implementation and integration efforts are expected.

VII. Conclusion

The presented paper started motivating the need for security assessments for well established fieldbus protocols. Subsequently, the requirements for a CAN fieldbus security assessment framework have been proposed. Aligned on these initial requirements, an open-source based architecture and implementation of the proposed framework has been presented. The paper concludes with a practical use-case scenario and an outlook on possible further development objectives and enhancements.

VIII. Project Funding

The work on Logboat is part of the German national security reference project IUNO (http://www.iuno-projekt.de). The project is funded by the Federal Ministry of Education and Research (BMBF) and aims to provide building-blocks for security in the emerging field of Industry 4.0.

References

[1] R. Drath and A. Horch, "Industrie 4.0: hit or hype", in *IEEE Industrial Electronics Magazine*, June 2014.

[2] *CAN Specification*. [Online]. Available: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf (visited on 03/18/2016).

[3] *CAN FD Specification*. [Online]. Available: https://web.archive.org/web/20130929023243/http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf (visited on 03/18/2016).

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, *et al.*, "Experimental security analysis of a modern automobile", in *IEEE Symposium on Security and Privacy*, Oakland, California, USA, 2010.

[5] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, *et al.*, "Hardware engines for bus encryption: a survey of existing techniques", in *IEEE Computer Society*, Munich, Germany, 2005.

[6] O. Hartkopp, "Programmierschnittstellen für eingebettete Netzwerke in Mehrbenutzerbetriebssystemen am Beispiel des Controller Area Network", PhD thesis, 2011. [Online]. Available: http://edoc.bibliothek.uni-halle.de/servlets/MCRFileNodeServlet/HALCoRe_derivate_00004667/Dissertation-Hartkopp-Onlineversion.pdf (visited on 03/23/2016).

[7] O. Hartkopp, W. Grandegger, *et al.*, *Readme file for the controller area network protocol family (aka socketcan)*, 2007. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/can.txt (visited on 03/07/2015).

[8] M. Kleine-Budde, "Socketcan - the official can api of the linux kernel", in *IEEE International Conference of Communications*, Algonquin College, Canada, 2012.

[9] J. Pereyda, *The Boofuzz Engine*. [Online]. Available: https://github.com/jtpereyda/boofuzz (visited on 03/15/2016).

[10] T. L. Foundation. (Oct. 5, 2015). Project to advance real-time linux, [Online]. Available: http://www.linuxfoundation.org/news-media/announcements/2015/10/linux-foundation-announces-project-advance-real-time-linux (visited on 03/14/2016).