A Survey on Penetration Test Tools for Controller Area Networks*

Full Paper

Enrico Pozzobon
Technical University of Applied Sciences Regensburg
Regensburg, Germany
enrico.pozzobon@othr.de

Nils Weiss
Technical University of Applied Sciences Regensburg
Regensburg, Germany
nils2.weiss@othr.de

Sebastian Renner
Technical University of Applied Sciences Regensburg
Regensburg, Germany
sebastian1.renner@othr.de

Rudolf Hackenberg
Technical University of Applied Sciences Regensburg
Regensburg, Germany
rudolf.hackenberg@othr.de

ABSTRACT

Controller Area Networks (CANs) are still most used network technology in todays connected cars. Today and in the near future, penetration tests in the area of automotive security will still require tools for CAN media access. More and more open source automotive penetration tools and frameworks are presented by researchers on various conferences. All with different properties in terms of usability, features and supported use-cases. If one wants to start with security investigations in automotive networks, he has to find a proper tool for his investigations by try and error of available solutions. This paper compares current available CAN media access solutions and gives an advices on competitive hard- and software tools for automotive penetration testing.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

CAN, Penetration Testing, Benchmarks

ACM Reference Format:
Enrico Pozzobon, Nils Weiss, Sebastian Renner, and Rudolf Hackenberg. 2018. A Survey on Penetration Test Tools for Controller Area Networks: Full Paper. In *Proceedings of ACM Computer Science in Cars Symposium (ACM-CSCS'18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Since the automotive industry is moving towards autonomous driving, more and more cars are or will soon be connected to a backend system in the Internet, the security of automotive systems becomes a crucial factor in the process of developing self-driving cars. With additional remotely accessible interfaces, which are needed to enable the communication between vehicles and the Internet, the attack surface of a modern car significantly increases. Furthermore, this development can expose protocols like CAN, which usually only have been used in the car's internal network, to remote ports. Since for example the CAN protocol was developed decades ago, it does not introduce any kind of mechanisms to secure the communication. The combination between the introduction of new connected interfaces, which are necessary for the use of self-driving features and the use of legacy and potentially unsecured protocols creates a new high-impact risk in the context of attacks on the car's IT security. Therefore the evaluation of this risk is an important factor in the process of security testing. To ensure reliable and sufficient testing, professional tools specifically developed to support automotive protocols are needed.

This paper introduces the most commonly used frameworks, soft- and hardware available in the field of automotive security testing. Additionally, a pattern on how to cluster them into subgroups will be shown. Moreover, different criteria, valid to conduct an extensive comparison between the chosen tools will be described, followed by a survey on the the actual tools. After explaining the test process for each subject under test, the test results will be concluded. The last paragraph will cover possible aspects that may be researched in the future.

2 RELATED WORK

In the area of performance evaluation of CAN drivers the work of Sojka et. al is mentionable. They performed an extensive timing analysis of the commonly used driver SocketCAN, in comparison with their own solution LinCAN on different Linux Kernels [21][20]. This differs from the research proposed in this paper, since it focuses solely on the driver itself, while we observe CAN tool solutions as a whole. Further research regarding CAN tools was published by Lebrun et. al in 2016. Lebrun and Demay introduced CANSPY, a tool for CAN frame inspection and manipulation, especially built to aid with security evaluations of CAN devices [9].

Besides the work done on CAN testing, in the field of web applications, alike surveys relevant to analyzing security tools have been conducted in the past. For example, Fonseca et. al benchmarked web vulnerability scanning tools using criteria such as vulnerability

*Produces the permission block, and copyright information

ACM-CSCS'18, September 2018, Munich, Germany

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

coverage and the false positive rate [3]. Doupe et. al accomplished similar research with the evaluation of vulnerability scanners for web applications in 2010 [1].

3 GENERAL REQUIREMENTS FOR AUTOMOTIVE PENETRATION TESTING

This section introduces some basic requirements for automotive penetration testing. A competitive tool should fulfill the following requirements. All mentioned requirements are derived from practical use-cases in automotive penetration testing.

3.1 Protocol Viewer

During any security tests on automotive networks, a clear way to inspect the current traffic on a network is absolutely necessary to improve the efficiency of automotive security investigations. A security researcher needs an interface which displays the live traffic and allows detailed investigations on captured data. A penetration tool has to detect and interpret the used protocol automatically. In addition to that, such a viewer needs to be extensible with additional brand specific protocol information. The following protocols should be supported

- CAN (ISO 11898)
- Unified Diagnostic Services (UDS) (ISO 14229)
- General Motor Local Area Network (GMLAN)
- CAN Calibration Protocol (CCP)
- Universal Measurement and Calibration Protocol (XCP)
- ISO-TP (ISO 15765)
- Diagnostic over Internet Protocol (IP) (DoIP) (ISO 13400)
- On-board diagnostics (OBD) (ISO 15031)
- Ethernet (ISO 8802/3)
- TCP/IP (RFC 793, RFC 7323 / RFC 791, RFC 2460)

3.2 Packet-Manipulation and Fuzzing

An automotive penetration test tool should be able to fuzz all common automotive and Ethernet protocols. A fuzzer with an easy to use interface and the capability of fuzzing and listening on different network interfaces at the same time should be available in a penetration test tool. The desired response of a fuzzed message often shows up on a different interface, maybe with a different protocol, or on output pins of an Electronic Control Unit (ECU). Another feature is conditional fuzzing. In many situations, an ECU has to be set into a special state before a fuzzer can send the message which has to be fuzzed.

3.3 ECU-Simulation

Usually, an ECU under test needs a special environment for its normal operation. This environment has to be simulated with periodic CAN and UDS messages on a connected bus. For more advanced investigations, a remaining bus simulation is required. Vector CANoe TODO CITEME for example is specialized in this kind of simulation. For white-box security tests, this would be the tool of choice for complex remaining bus simulations. On black- and gray-box security tests, periodic messages are sufficient to spoof a certain operation state. Capturing, modifying and periodically replaying of various messages has to be supported by this penetration test tool.

3.4 Man in the Middle (MITM) Capabilities

In order to investigate the communication from a specific ECU, this ECU has to be isolated through a MITM attack. An advanced penetration test tool needs some functionality to setup a MITM proxy between two CAN or Ethernet interfaces. In addition, functions to filter or hijack the communication between ECUs are very useful during black-box security investigations.

3.5 Media Access Layer Requirements

The mentioned higher level requirements are absolutely necessary for efficient penetration testing in automotive networks. It is possible to derive the following media access requirements from these existing requirements. For verification of this media access layer requirements, multiple benchmarks will be created on available interfaces for the media access to CAN. A performance evaluation of media access interfaces for automotive penetration tests will be introduced in the final section of this paper.

3.5.1 Latency. The latency between the initiation of a write to an automotive network from an user space application to the actual presence of this data on the bus is crucial for all kind of fuzzing and spoofing tests. If one wants to for example react as soon as possible on a certain CAN message and send out a specific message as response, the latency of the media access interface has to be as short as possible.

3.5.2 TX-Throughput. During the investigation of denial of service or flooding attacks on the CAN bus, a penetration test tool should be capable of creating a throughput which leads to a bus load of 100 percent. To achieve this load from user space writes of a penetration test application, the interface driver, the operating system and the media access device itself need to be able to handle writes to the bus faster as the time a message is present on the bus.

3.5.3 RX-Throughput. A common use case in automotive penetration testing is the sniffing of firmware updates on the CAN bus. During firmware updates of ECUs on the CAN bus, usually only the target ECU and the ECU or the repair shop tester, which are delivering an update, are allowed to communicate on the bus. All other ECUs remain silent until the flashing procedure finishes. This dedicated communication to only two ECUs on the bus allows maximum bandwidth for the firmware transfer. This leads also to a maximum bus load. A media access device for penetration tests, the operating system and the used drivers need to be capable of handing over this bandwidth to a user space application. Otherwise, the penetration tester is missing messages which leads to inconsistent results.

3.5.4 Reliability. For any security investigation in automotive networks, the penetration test tool needs a certain reliability. Often, traces and captures can be done only once or require extensive preparations to the device under test. Unreliable tools will make difficult security investigations in automotive networks even harder or lead to wrong results.

4 CLASSIFICATION OF MEDIA ACCESS TYPES

Whenever one wants to start with automotive penetration testing, a soft- and hardware tool for the communication with a car's

CAN-bus is required. To get an overview of existing open-source automotive penetration test frameworks, a survey on existing applications and tools for automotive penetration tests was conducted. This survey focused on hard- and software interfaces to access automotive networks (CAN). The following table gives an overview of common used hard- and software for the media access to CAN.

	SLCAN Device	ELM327	SocketCAN	Python-CAN	Others
Busmaster [13]					X
c0f [18]			X		
can-utils [14]			X		
CANiBUS [17]	X				X
CANToolz [16]	X				X
Caring Caribou [15]				X	
Kayak [10]			X		
Metasploit [12]		X	X		
O2OO [25]		X			
pyfuzz_can [6]				X	
python-OBD [26]		X			
Scapy [24]			X	X	
UDSIM [19]			X		
Wireshark [27]			X		

Table 1: Summary of used hard- and software for CAN-bus access in open-source automotive penetration test frameworks.

With respect to the evaluation of used media access interface on public available penetration test frameworks, the most used media access interfaces will be taken into account on a performance evaluation. In order to be able to compare available CAN media access solutions for automotive penetration testing, representative groups for the different interface solutions have been selected. Devices inside a group have an identical hard- and software architecture, and will therefore show a similar behavior during benchmarks.

The following list shows the chosen groups for the media access layer comparison of CAN-bus interfaces:

(1) **Native-CAN**
The CAN-peripheral module is directly accessible from the main processor. A BeagleBoneBlack with an AM335x 1 GHz ARM Cortex-A8 processor and a Banana Pi Pro with an Allwinner A20 dual core ARM processor are the used devices under test in this group. All automotive penetration test frameworks which use SocketCAN or Python-CAN are able to use lower layer CAN-bus interfaces from this group.

(2) **Serial Peripherial Interface (SPI)-to-CAN**
The CAN-peripheral module is accessible over a SPI connection. A Raspberry Pi with a MCP2515 TODO CITE_ME SPI-CAN module is used as device under test. SocketCAN is the common way to give an user space application access to the CAN-bus in this group.

(3) **Universal Serial Bus (USB)-to-CAN over Serial Line CAN (SLCAN)**
The CAN-peripheral module is accessible over a serial line communication. The SLCAN protocol is used to access the CAN-bus from the device under test. The open hard- and software project USBtin TODO CITE_ME is used as an interface in this class. User space applications can either access the CAN-bus directly over a serial connection, or can connect to a SocketCAN socket, which is offered by an application called slcand.

(4) **ELM327 TODO CITE_ME**
The ELM327 is an OBD-to-serial interface with a custom AT command set. As a device under test, an ELM327 with USB-interface is used. The CAN-bus is accessible over a serial interface.

(5) **USB-to-CAN**
The CAN-peripheral is accessed over a USB-interface in this class. As devices under test, a PEAK PCAN TODO CITE_ME and a Vector VN1611 TODO CITE_ME USB to CAN adapter are used. Devices inside this group require proprietary drivers. For user space applications, the CAN-bus is accessible through dynamic linked libraries supported by python-CAN or SocketCAN sockets.

The following listing shows media access devices which were excluded from this benchmark. A brief reason for for the exclusion is given:

- **CANtact [2]**
  CANtact has a very similar architecture to USBtin. It is likely, that the test results would be very similar to each other.
- **CANBadger [4]**
  The CANBadger has to be build on your own. It is currently not possible to order a ready-to-use piece of hardware, the parts and software for the CANBadger are available and have to be assembled together according to the datasheets.
- **Kvaser [8], IXXAT [11] and INTREPID Control Systems tools [7]**
  These tools are similar to tested tools in the USB-to-CAN group. Including them into the survey would probably just produce duplicated results, which does not create and an added value to this research.
- **GoodThopter12 [5]**
  The GoodThopter12 has a similar architecture as the USBtin. The results should be similar to the results in the USB-to-CAN over SLCAN group.

5 BENCHMARKING CRITERIA AND PROCESS

5.1 Test-Setups

Every media access device has to be tested individually and with a different setup. All tests on media access devices were performed by user space applications, running with maximum priority and compiled with maximum optimization settings (where applicable). All the tests involving a USB-to-CAN device were executed on the same laptop computer. All operating systems were tested with the minimum amount of modifications from the moment of installation, including installation of the required software to run the tests,

drivers for the CAN interfaces, and device trees for the native CAN implementations.

- **TI AM3358 (Native CAN)**
  - Platform: BeagleBone Black
  - OS: Debian GNU/Linux 8
  - Kernel: 4.4.54-ti-r93 #1 SMP
  - User space application: Compiled program, written in C, using SocketCAN
- **Allwinner A20 (Native CAN)**
  - Platform: Lemaker Banana Pro
  - OS: Armbian GNU/Linux 9
  - Kernel: 4.14.18-sunxi #24 SMP
  - User space application: Compiled program, written in C, using SocketCAN
- **MCP2515 (SPI-to-CAN)**
  - Platform: Raspberry Pi 2 Model B
  - OS: Raspbian GNU/Linux 9
  - Platform-specific settings: SPI clock frequency set to 8 MHz
  - Kernel: 4.14.30-v7+ #1102 SMP
  - User space application: Compiled program, written in C, using SocketCAN
- **USBtin (USB-to-CAN over SLCAN)**
  - Platform: Dell Latitude E5470
  - OS: Antergos Linux
  - Kernel: 4.15.15-1-ARCH #1 SMP PREEMPT
  - User space application: Compiled program, written in C, using SocketCAN
- **ELM327 (USB-to-OBD)**
  - Platform: Dell Latitude E5470
  - OS: Antergos Linux
  - Platform-specific settings: UART baud rate set to 460.8 kHz
  - Kernel: 4.15.15-1-ARCH #1 SMP PREEMPT
  - User space application: Compiled program, written in C, using glibc termios
- **PEAK PCAN (USB-to-CAN)**
  - Platform: Dell Latitude E5470
  - OS: Antergos Linux
  - Kernel: 4.15.15-1-ARCH #1 SMP PREEMPT
  - User space application: Compiled program, written in C, using SocketCAN
- **Vector VN1611 (USB-to-CAN)**
  - Platform: Dell Latitude E5470
  - OS: Windows 10
  - User space application: Python script

5.2 Test Procedure

All tests were executed with the maximum baud rate supported by the tested device. This maximum baud rate was 1 MHz for every device except for the ELM327 which only supports a maximum rate of 500 kHz.

All tests involved connecting the device under test to a microcontroller and a logic analyzer for taking measurements. The logic analyzer was connected to the microcontroller CAN TX and RX pins, in parallel to the transceiver, and the length of the bus connecting the tested device to the microcontroller was 1.2 meters.

5.2.1 Latency Test. The objective of this test is to measure the amount of time taken by each tool and framework to forward a CAN frame from the physical layer to the user application and vice versa. This measurement is important because it shows whether a tool is suitable for timing-critical tasks, such as replying to a frame before an ECU does, or triggering an action every time a packet is detected on the bus.

In order to test for latency, a microcontroller with CAN capabilities (Espressif ESP32 [22]) was connected to the bus and set to send a CAN frame every 20 ms, while the device under test was configured to receive these CAN frames and reply as soon as possible with another frame. A logic analyzer was connected to the receive and transmit lines between the transceiver and the microcontroller, and set to sample data at 10 million samples per second. Since the maximum CAN baud rate reachable by the tested devices is 1 MHz, a sampling rate of 10 MHz is sufficient to capture and parse the individual CAN frames. An C application was developed using the Sigrok library to interface with the logic analyzer and capture precise timings of the time between a CAN frame coming from the microcontroller and the response coming from the device under test [23].

Given a pair of "request" and "response" CAN frames (where the request is sent by the microcontroller and the response is sent by the device under test), we define the latency introduced by the tested device and framework to be the time passed from the "ACK" field of the request CAN frame and the "START OF FRAME" field of the response frame. During this time, the frame is received by the hardware of the Device Under Test (DUT) and it is forwarded to the user space application, which immediately generates the response CAN frame without any processing or delay. The measured latency therefore represents the time which a CAN frame takes to travel from the physical layer to the application layer and back to the physical layer.

It is notable, that the latency measured in this way can never be lower than ten baud lengths according to the CAN specification, since between the "ACK" field and the "START OF FRAME" field of the next frame there must always be an "END OF FRAME" field consisting of seven recessive bits and an "INTERMISSION" field consisting of three recessive bits. However, while achievable on a microcontroller, such a low latency was never obtained on any of the tested devices, since they all involve a user space application running on a non real-time operating system.

5.2.2 TX-Throughput Test. The objective of this test is determining how fast a given device can write frames to the CAN bus, and what is the maximum data rate it can transmit at. Such a test is important to verify that a device is capable of flooding the bus with frames and simulating an high load on the bus.

To execute this test, a small program was written for each tested software stack that would simply send the same CAN frame one million times in a loop, while connected to a microcontroller that would only acknowledge every frame. The CAN lines were probed with a logic analyzer to measure the time passed between each frame, and statistical data was extracted from these measure.

The test was repeated with two different kinds of CAN frames: Frames without a payload and frames with a payload of the maximum size. The smaller frames which still have an effective amount

of data of 11 bits in the identifier (contained in the "ARBITRATION FIELD") were used to test the maximum frame rate achievable by each device. The longer frames which contain 75 bits of data were used to test the maximum achievable bit rate by each device.

In this test, there was no noticeable difference in the results when testing an high-level framework (e.g. Scapy) or the low layer implementation (e.g. SocketCAN), due to the amount of buffering done by the operating systems and the relatively low data rate of the CAN bus when compared to the speed which frames are processed at. Therefore, only one test setup result is shown for each tested device.

5.2.3 RX-Throughput Test. The objective of this test was determining how fast a given device can read frames from the CAN bus, and what is the maximum data rate it can receive at. This test is important for determining if a device can receive all CAN frames in a situation of high bus load, such as when an ECU firmware is transmitted as part of a software update.

To execute this test, the microcontroller was programmed to output a CAN frame one million times with the smallest delay allowed by the CAN specification between one frame and the next. In order to achieve maximum consistency in the transmission interval, as well as sending the same number of frames whether they were being acknowledged or not, the CAN frames were transmitted using the Inter-Integrated Circuit (IC) Sound (I2S) peripheral of the ESP32 microcontroller. The validity of the CAN frames sent and of their timing was verified with the help of a logic analyzer.

The test was repeated multiple times for each device, with both short (no payload) and long (8 bytes payload) frames. The time between the start of two consecutive frames was 48 bauds for the short frames, and 111 bauds for the long frames. During each test sequence, the number of frames received by the device under test was recorded with a simple program, and the test was repeated 1000 times, for a total of 10^9 frames sent to each device for both long and short frames.

6 EVALUATION OF MEDIA ACCESS DEVICES

6.1 Latency Evaluation

The results for the latency tests are presented in the form of histograms, Empirical cumulative distribution function (ECDF) and box plots.

The vertical axis of the histogram represents the relative frequency of the time measured in one request-response pair. The most desirable result would be to have a single sharp peak centered as close to 0 ms as possible, and a wide curve indicates a large variance in the measurements. The presence of multiple peaks in many histograms might indicate that a buffer is being filled asynchronously with the received CAN frames and it is only being read after a timeout expires, or it might be due to the operating system executing code for handling interrupts from other peripherals.

The ECDF plots present the integral of the data shown in the histograms. Given a point (x, y) in these plots, y is the probability that a frame was replied to in time less than x ms. The intersection between the curve and the 0.5 horizontal line in the ECDF plot shows the average latency for that device, while the upmost part represents the worst case scenario.

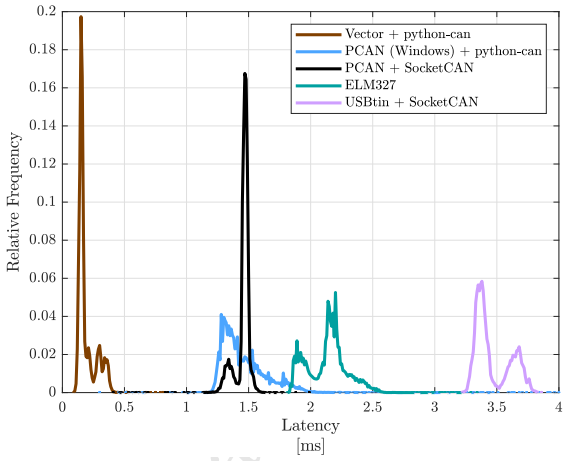

Figure 1: Latency histogram for USB to CAN devices, all tested on the same laptop computer.

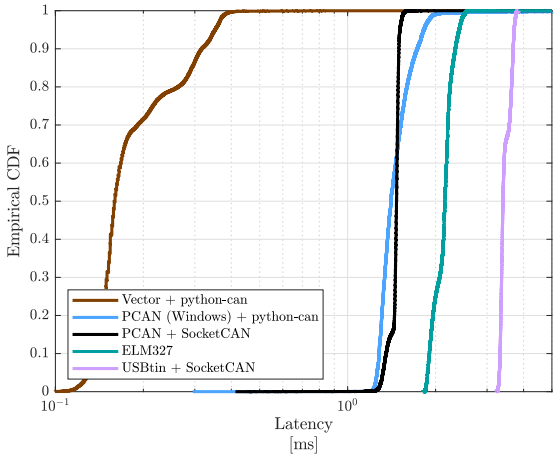

Figure 2: Empirical distribution function of the latency for USB to CAN devices, all tested on the same laptop computer.

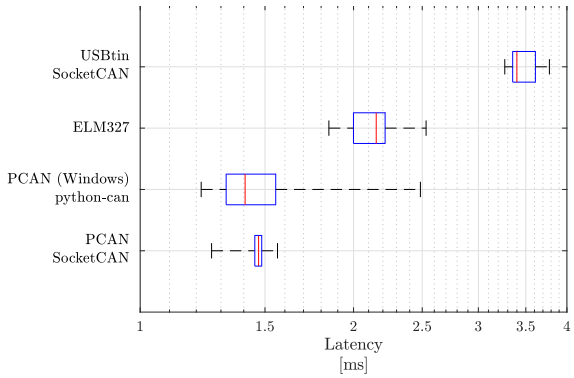

Figure 6: Box plot of the measured latency for the tested devices. Whiskers represent the 5‰and 995‰quantiles.

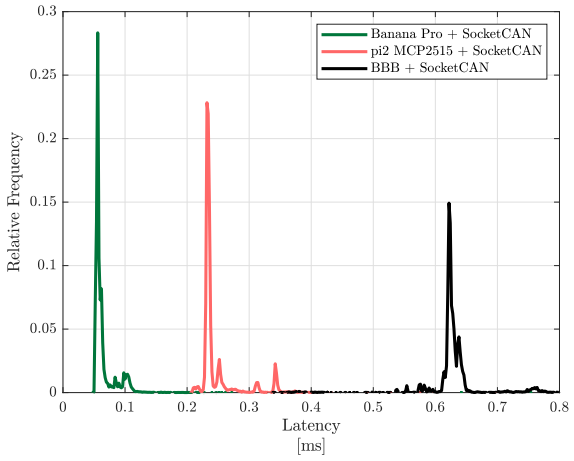

Figure 3: Latency histogram for single board computers using SocketCAN.

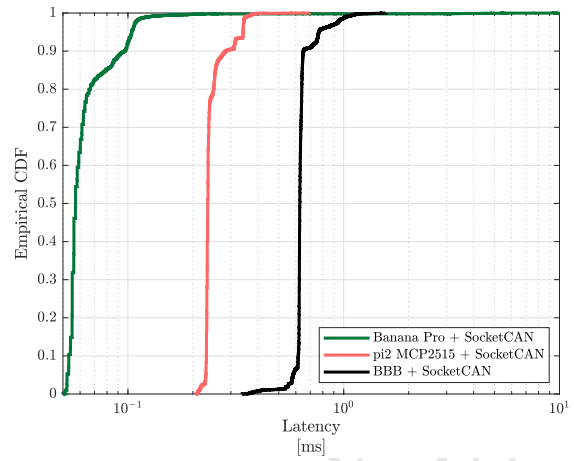

Figure 4: Empirical distribution function of the latency for single board computers using SocketCAN.

6.2 TX-Throughput Evaluation

When testing the transmission speed of the PCAN and Vector USB-to-CAN interfaces, on any framework or operating system, the performance was always good enough to obtain a 100% utilization of the bus. Any variance found in the results of these interfaces is only present in the first couple of frames, after which the buffer becomes full and any subsequent sent frame will enter the bus as soon as it becomes free.

Single board computers with native interfaces performed could almost constantly send enough CAN frames to reach 100% bus load, but had a larger variance in the results, likely due to other processes sharing time on the limited CPUs.

The bottleneck for the transmission speed of the USBtin is the emulated USB UART device, which appears to operate at a maximum speed of 411 kb/s. The speed is further limited by the overhead

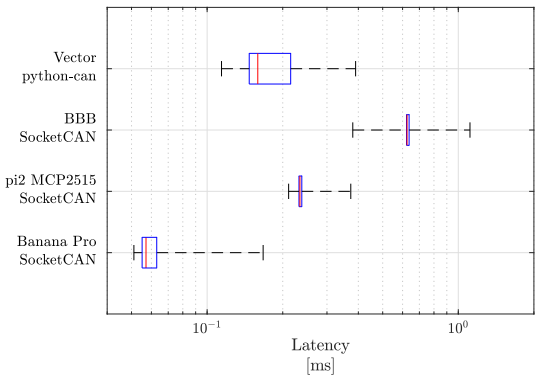

Figure 5: Box plot of the measured latency for the tested devices. Whiskers represent the 5‰and 995‰quantiles.

of the SLCAN protocol, which is introduced by the use of hexadecimal characters and the framing of SLCAN commands, which more than halve the effective bitrate.

The MCP2515 SPI-to-CAN interface is also a bottleneck. While the speed of the SPI interface would be sufficient to transfer all the commands necessary to send a CAN frame in a time smaller than the duration of the CAN frame itself, each successful transmission enables the TX0IF flag in the CANINTF register of the MCP2515. TODO CITE_Datasheet Then this happens, the interrupt pin is signaled, and the MCP2515 driver on the operating system will query the interface for the pending interrupt. The three SPI commands necessary for reading and resetting the TX0IF flag and the interruption caused by the driver amount to a total overhead of approximately 70 μs for each sent CAN frame when the SPI clock frequency is set to 8 MHz.

	μ [kb/s]	σ [kb/s]
USBtin + SocketCAN	57.9261	0.94254
pi2 MCP2515 + SocketCAN	66.0764	4.0136
Banana Pro + SocketCAN	209.29	7.054
BBB + SocketCAN	225.2256	14.7024
PCAN + SocketCAN	229.1065	0.046076
PCAN + python CAN on linux	229.1066	0.0461
PCAN + python CAN on windows	228.5465	1.6716
Vector + python CAN	228.568	0.042923

Table 2: Results for TX-Throughput evaluation with short CAN-frame on the tested media access devices.

	μ [kb/s]	σ [kb/s]
ELM327	46.8751	2.4707
USBtin + SocketCAN	138.7733	7.3381
pi2 MCP2515 + SocketCAN	348.8258	9.6974
Banana Pro + SocketCAN	649.1451	5.1531
BBB + SocketCAN	672.3208	12.6014
PCAN + SocketCAN	675.5985	0.2309
PCAN + python CAN on linux	674.9145	0.11954
PCAN + python CAN on windows	674.9153	0.052812
Vector + python CAN	674.9029	0.13812
Vector + scapy	674.9034	0.05797

Table 3: Results for TX-Throughput evaluation with long CAN-frame on the tested media access devices.

6.3 RX-Throughput Evaluation

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

TODO: ELM327 is missing

	μ [kb/s]	σ [kb/s]
USBtin + SocketCAN	89.0387	0.00010953
pi2 MCP2515 + SocketCAN	217.5796	2.8753
BBB + SocketCAN	94.3395	1.4791
Banana Pro + SocketCAN	228.855	1.5459
PCAN + SocketCAN	229.1495	0.028675
PCAN (Windows) + python-can	229.1667	0
Vector + python-can	229.1667	0

Table 4: Results for RX-Throughput evaluation with short CAN-frame on the tested media access devices.

	μ [kb/s]	σ [kb/s]
USBtin + SocketCAN	179.9568	0.0020292
pi2 MCP2515 + SocketCAN	482.6823	16.6219
BBB + SocketCAN	674.4699	2.2591
Banana Pro + SocketCAN	675.5324	0.30141
PCAN + SocketCAN	675.6757	0
PCAN (Windows) + python-can	675.6757	0
Vector + python-can	675.6757	0

Table 5: Results for RX-Throughput evaluation with long CAN-frame on the tested media access devices.

7 EVALUATION OF PENETRATION TEST FRAMEWORKS

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est

Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

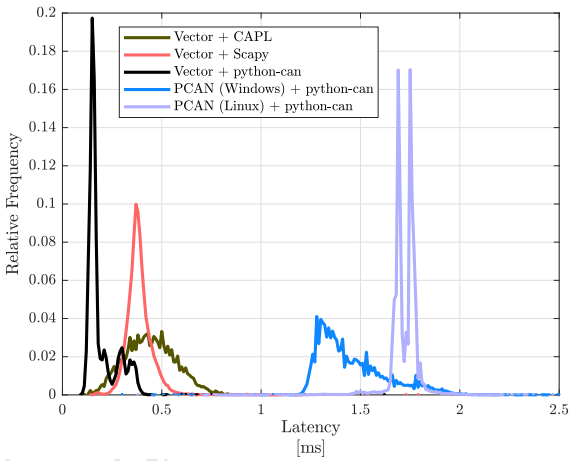

Figure 7: Latency histogram for USB to CAN frameworks, all tested on the same laptop computer.

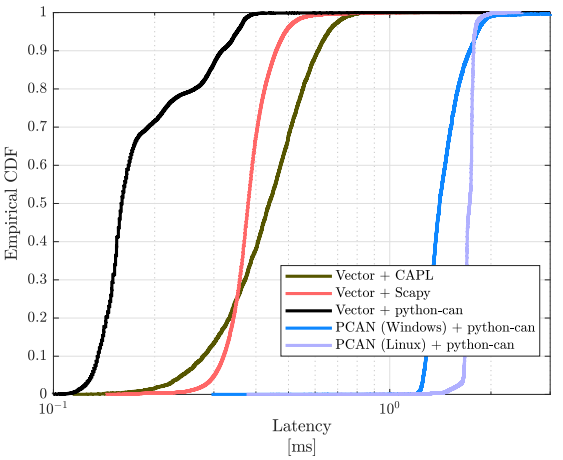

Figure 8: Empirical distribution function of the latency for USB to CAN frameworks, all tested on the same laptop computer.

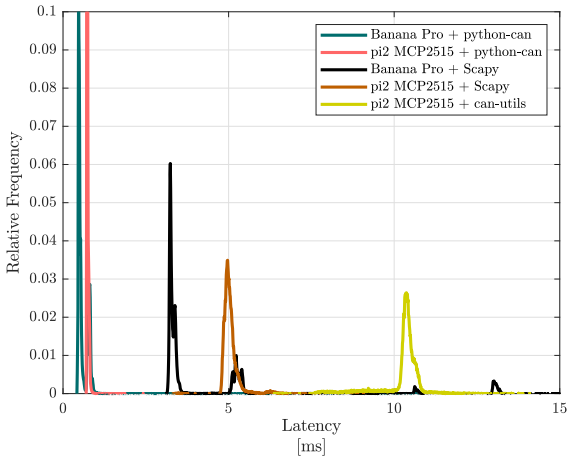

Figure 9: Latency histogram for single board computers using different frameworks.

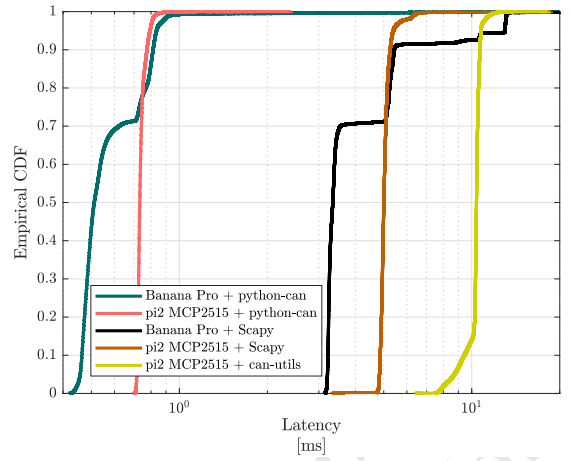

Figure 10: Empirical distribution function of the latency for single board computers using different frameworks.

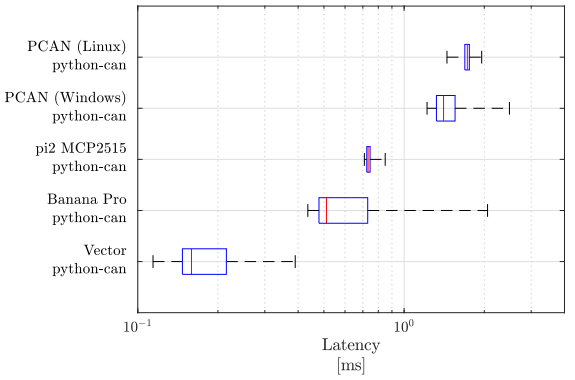

Figure 12: Box plot of the measured latency for the tested frameworks. Whiskers represent the 5‰and 995‰quantiles.

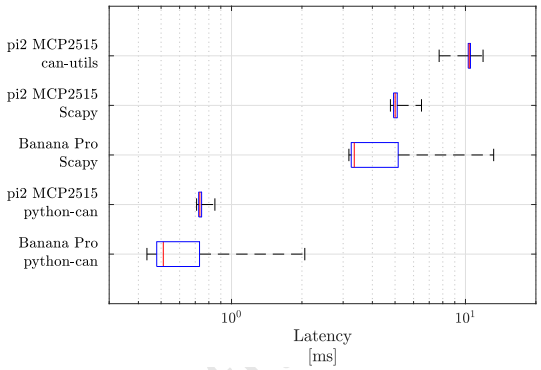

Figure 11: Box plot of the measured latency for the tested frameworks. Whiskers represent the 5‰and 995‰quantiles.

8 CONCLUSION

Non of the investigated tools and frameworks showed an error-free performance on Linux. Proprietary tools on Windows performed very good overall. The python-can framework delivered good results on both tested operating systems, Linux and Windows. Furthermore, this framework supports open and closed source media access devices.

Very popular tools like the ELM327 or USB-to-CAN over SLCAN devices had a bad outcome in our performance evaluation. These tools aren't usable for advanced penetration tests with higher CAN baud rates.

Single-board-computers are a very good alternative to professional CAN media access devices. The availability of a full-featured operating system creates a high level of usability in a wide variety of applications. The tested devices are available for less than 100 euros. This fact brings single-board-computers into the same price range as popular CAN interfaces like the ELM327 or USB-to-CAN over SLCAN devices.

Professional CAN tools for automotive engineering tasks showed the best results in terms of reliability and throughput. However, these tools are of course much more expensive than single-board computers.

9 FUTURE WORK

Inspected automotive penetration test frameworks showed only very limited penetration test capabilities. Most tools only support one specific use case. An open source tool with support of various car brands and proprietary automotive protocols is not available, yet. On the other hand, commercial tools do not perfectly fit for penetration testing tasks and are very expensive in general. Also the fact that commercial tools are closed source software products decreases the suitability and flexibility for specific penetration tests. To improve automotive penetration testing in general, comprehensive open source tools, which fulfill the mentioned requirements from section 3 are necessary. This paper showed that the software

and hardware interfaces, python-can and SocketCAN, are appropriate for the implementation of advanced automotive penetration frameworks.

In the future, our research in the area of automotive penetration testing tools will be focused on developing a new framework based on existing open-source software. The experience gathered during the work for this paper will serve as a base of requirements when designing the tool. Later, similar tests will be conducted to evaluate and verify the framework's performance.

REFERENCES

[1] Adam Doupé, Marco Cova, and Giovanni Vigna. 2010. Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 111–131.

[2] Eric Evenchick. 2016. CANtact. (2016). Retrieved April 16, 2018 from http://linklayer.github.io/cantact/

[3] Jose Fonseca, Marco Vieira, and Henrique Madeira. 2007. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*. IEEE, 365–372.

[4] Code White GmbH. 2016. CANBadger. (2016). Retrieved April 16, 2018 from https://gutenshit.github.io/CANBadger/

[5] Travis Goodspeed. 2017. GoodThopter12. (2017). Retrieved April 16, 2018 from http://goodfet.sourceforge.net/hardware/goodthopter12/

[6] Bill Hass. 2018. pyfuzz_can. (2018). Retrieved April 16, 2018 from https://github.com/bhass1/pyfuzz_can

[7] INC. INTREPID CONTROL SYSTEMS. 2018. INTREPID. (2018). Retrieved April 16, 2018 from https://www.intrepidcs.com/

[8] Kvaser. 2018. Kvaser. (2018). Retrieved April 16, 2018 from https://www.kvaser.com

[9] Arnaud Lebrun and Jonathan-Christofer Demay. 2016. Canspy: a platform for auditing can devices. (2016).

[10] Jan-Niklas Meier. 2014. Kayak. (2014). Retrieved April 16, 2018 from http://kayak.2codeornot2code.org/

[11] HMS Industrial Networks. 2018. IXXAT. (2018). Retrieved April 16, 2018 from https://www.ixxat.com

[12] Rapid7. 2018. Metasploit. (2018). Retrieved April 16, 2018 from https://www.metasploit.com/

[13] RBEI and ETAS. 2017. BUSMASTER. (2017). Retrieved April 16, 2018 from https://github.com/rbei-etas/busmaster/

[14] Volkswagen Group Electronic Research. 2018. Linux-CAN / SocketCAN user space applications. (2018). Retrieved April 16, 2018 from https://github.com/linux-can/can-utils

[15] Christian Sandberg, Kasper Karlsson, Tobias Lans, Mattias Jidhage, Johannes Weschke, and Filip Hesslund. 2018. Caring Caribou. (2018). Retrieved April 16, 2018 from https://github.com/CaringCaribou/caringcaribou

[16] Alexey Sintsov. 2017. CANToolz - framework for black-box CAN network analysis. (2017). Retrieved April 16, 2018 from https://github.com/CANToolz/CANToolz

[17] Craig Smith. 2013. CAN Device Vehicle Research Server (OpenGarages.org). (2013). Retrieved April 16, 2018 from https://github.com/Hive13/CANiBUS

[18] Craig Smith. 2015. CAN of Fingers (c0f). (2015). Retrieved April 16, 2018 from https://github.com/zombieCraig/c0f

[19] Craig Smith. 2017. UDSim. (2017). Retrieved April 16, 2018 from https://github.com/zombieCraig/UDSim

[20] Michal Sojka, Pavel Píša, Ondřej Špinka, and Zdeněk Hanzálek. 2011. Measurement automation and result processing in timing analysis of a Linux-based CAN-to-CAN gateway. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, Vol. 2. IEEE, 963–968.

[21] M. Sojka, P. PÃŋÅąa, M. Petera, O. Åăpinka, and Z. HanzÃąlek. 2010. A comparison of Linux CAN drivers and their applications. In *International Symposium on Industrial Embedded System (SIES)*. 18–27. https://doi.org/10.1109/SIES.2010.5551367

[22] Espressif Systems. 2018. Espressif Systems ESP32. (2018). Retrieved April 16, 2018 from https://www.espressif.com/en/products/hardware/esp32/overview

[23] Bert Vermeulen Uwe Hermann. 2018. Sigrok. (2018). Retrieved April 16, 2018 from https://sigrok.org/

[24] Guillaume Valadon and Pierre Lalet. 2018. Scapy. (2018). Retrieved April 16, 2018 from http://www.secdev.org/projects/scapy/

[25] Folkert van Heusden. 2014. O2OO. (2014). Retrieved April 16, 2018 from https://www.vanheusden.com/O2OO/

[26] Brendan Whitfield. 2016. python-OBD. (2016). Retrieved April 16, 2018 from https://github.com/brendan-w/python-OBD

[27] Peter Wu. 2018. Wireshark. (2018). Retrieved April 16, 2018 from https://www.wireshark.org/