Plug-N-Pwned: Comprehensive Vulnerability Analysis of OBD-II Dongles as A New Over-the-Air Attack Surface in Automotive IoT

Haohuang Wen
The Ohio State University
wen.423@osu.edu

Qi Alfred Chen
University of California, Irvine
alfchen@uci.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

Abstract

With the growing trend of the Internet of Things, a large number of wireless OBD-II dongles are developed, which can be simply plugged into vehicles to enable remote functions such as sophisticated vehicle control and status monitoring. However, since these dongles are directly connected with in-vehicle networks, they may open a new over-the-air attack surface for vehicles. In this paper, we conduct the first comprehensive security analysis on *all* wireless OBD-II dongles available on Amazon in the US in February 2019, which were 77 in total. To systematically perform the analysis, we design and implement an automated tool DONGLESCOPE that dynamically tests these dongles from all possible attack stages on a real automobile. With DONGLESCOPE, we have identified 5 different types of vulnerabilities, with 4 being newly discovered. Our results reveal that each of the 77 dongles exposes at least two types of these vulnerabilities, which indicates a widespread vulnerability exposure among wireless OBD-II dongles on the market today. To demonstrate the severity, we further construct 4 classes of concrete attacks with a variety of practical implications such as privacy leakage, property theft, and even safety threat. We also discuss the root causes and feasible countermeasures, and have made corresponding responsible disclosure.

1 Introduction

On-Board Diagnostics (OBD) [1] is a standard widely adopted for an automobile to self-diagnose and report its internal working status (such as voltage, fuel level, and speed). As the latest and most popular OBD standard, OBD-II is universally deployed in gasoline vehicles of US after 1996 for mandated emission inspection [2]. With the growing trend of the Internet of Things (IoT), a large number of wireless OBD-II dongles are developed, enabling vehicle owners to conveniently perform remote vehicle functions from companion mobile apps, like many other IoT devices, for simple status monitoring and diagnosis to sophisticated vehicle control such as disabling remote unlocking or seat-belt warnings.

While wireless OBD-II dongles do provide rich functions and great convenience, their usage exposes a wireless entry to the internal vehicle systems from the external world, which thus inevitably brings security concerns. On one hand, OBD-II dongles are connected with the in-vehicle Control Area Network (CAN) buses to fetch diagnostic data through the OBD-II port. On the other hand, they interact with external companion apps via wireless network to transfer data and commands. If not properly designed with security principles and practices, these dongles may enable a series of new over-the-air vehicle attack vectors, compromising not only the user property and privacy, but also the safety of drivers, passengers and pedestrians. For example, in 2017 it was found that the vulnerabilities on a Bosch Drivelog Connector OBD-II dongle enabled a nearby attacker to remotely shut down the engine while the vehicle was still in motion [3]. This dongle was soon removed from the market. As of today, there are still a great number of OBD-II dongles available on the market, which are popular among drivers, repair technicians, and also auto insurance companies. However, whether these dongles are vulnerable to remote or nearby attacks remains unknown to the public.

To fill this knowledge gap, in this paper we conduct the first comprehensive security analysis on *all* wireless OBD-II dongles available on Amazon in the US in February 2019, which were 77 in total. To systematically perform the analysis, we first define the attack surface based on the stages of how a remote or nearby attack could use the CAN bus through wireless OBD-II dongles: broadcast, connection and communication. Next, we design and implement an automated tool DONGLESCOPE that is capable of dynamically testing potential vulnerabilities at all these stages with a dongle under test plugged into the OBD-II port on a real automobile, with the assistance of companion mobile app analysis to reverse engineer the intended messages to these dongles, which are used to design test messages in the communication stage.

Through intensive experiments on these 77 dongles, we have identified 5 different types of vulnerabilities across the three attack stages, in which 4 are newly discovered. Among the 77 dongles, we find *each* of them exposes at least two types of these vulnerabilities across the three stages, which indicates a widespread vulnerability exposure among wireless OBD-II dongles on the market today. Specifically, we find that around 85% of these dongles have neither connection-layer nor application-layer authentication, which essentially provides a nearby attacker arbitrary access to the CAN bus once they are discovered in the broadcast stage. We further discover that 29 (37.66%) dongles are vulnerable to such malicious access even when the vehicle owner's mobile device is connected with them. In the communication stage, we find that 52 (67.53%) dongles fail to filter out CAN bus messages with functions unsupported in the dongles, meaning that attackers can send safety-critical vehicle control commands, e.g., gear shifting, even when the attacked dongle is originally designed only for diagnostic purpose. Even worse, a few dongles are vulnerable to over-the-air dongle firmware subverting or extraction. Last but not least, the aforementioned vulnerability can be fingerprinted using broadcast information for nearly half (42.86%) of the dongles, which allows nearby attackers to conveniently pinpoint which dongles to attack and how to attack them over the air.

To demonstrate the severity of these vulnerabilities, we further construct 4 classes of concrete attacks building upon these vulnerabilities and validated them on our testing automobile. These attacks can lead to a wide range of practical implications, including privacy leakage, property theft, and even safety threats to drivers, passengers, and pedestrians. Among the 77 dongles we collected, 84% of them are vulnerable to at least one of these four attack classes, and nearly 60% are vulnerable to at least three.

The analysis results in this paper evidently point out a general and systematic lack of security protection in wireless OBD-II dongles today, which is known for IoT devices in home setting [4–9], but for the first time comprehensively revealed and quantified for those in the vehicle setting. Since the vehicle setting is safety-critical, one would expect that its IoT devices have more scrutinized security practices. However, based on our results, this is very unfortunately not the case today. To proactively address this, we leverage the insights in our analysis to discuss the root causes and feasible countermeasures, and meanwhile we have also already made responsible disclosure to the corresponding dongle manufacturers. As IoT devices are increasingly used in safety-critical domains such as automobiles, we expect that our domain-specific findings and their security/safety implications can send a strong and timely message to start developing and deploying principled security designs in these critical application domains for the IoT.

Contributions. In short, we make the following contributions in this paper:

- **Comprehensive vulnerability analysis.** We conduct the first comprehensive security analysis on *all* wireless OBD-II dongles available on Amazon in the US in February 2019. Targeting the threat model of over-the-air vehicle attacks, we systematically identify the attack surface as attack vectors across three necessary attack stages: broadcast, connection, and communication. We design and implement an automated tool DONGLESCOPE that is capable of dynamically detecting potential vulnerabilities at these three stages with a real automobile.
- **Vulnerability discovery and quantification.** With DONGLESCOPE, we have systematically analyzed the 77 dongles we collected and identified 5 types of vulnerabilities across the three attack stages, in which 4 are newly discovered. Our results show that *each* of the 77 dongles has at least two vulnerabilities exposed across the three stages, which indicates a widespread vulnerability exposure among wireless OBD-II dongles on the market today.
- **Attack case-study construction.** To demonstrate the severity of the identified vulnerabilities, we further construct 4 classes of concrete attacks building upon these vulnerabilities and validate them on a testing automobile. These attacks can lead to a wide range of practical implications, including privacy leakage, property theft, and even safety threats to drivers, passengers, and pedestrians. We also discuss the root causes and feasible countermeasures, and have also made responsible disclosure to the corresponding dongle manufacturers.

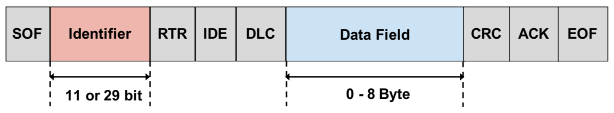

Figure 1: Structure of a CAN Bus Message.

Roadmap. The rest of this paper is organized as follows. Necessary background related to the Control Area Network and OBD-II dongles is introduced in §2. Next, we describe the attack model in §3. Then, we present the detailed design and implementation of DONGLESCOPE in §4. In §5, we present the vulnerability analysis results, followed by the attack case studies in §6 and discussions in §7. We review the related works in §8, and finally conclude in §9.

2 Background

2.1 Control Area Network

Automobiles are no longer isolated mechanical devices. Instead, they are sophisticated computer systems with a great number of Electronic Control Units (ECUs) responsible for different capabilities such as steering, braking, and accelerating. These ECUs form a complicated network with massive number of messages transferring back and forth at any time. To make sure that such a complicated system works properly, an in-vehicle network is necessary to coordinate the transfer of the messages between these ECU components.

Control Area Network (CAN) bus is the most ubiquitous message-based protocol deployed in the modern vehicles today [10]. In this network, ECUs are mutually connected with a bus system, constantly broadcasting and listening to CAN bus messages. The structure of a CAN bus message is shown in Figure 1 [11]. The identifier and the data field in a CAN bus message determine the function of a CAN bus message. The identifier of a message consists of 11 or 29 bits, indicating the sender ECU of this message (e.g., `0x191` represents the transmission system ECU). The data field of a message contains up to 8 bytes, storing the state parameters of the sender ECU (e.g., the third byte of data represents the engine speed).

2.2 OBD-II Dongles and Companion Apps

OBD-II dongles. OBD-II is a high-level communication protocol (a "language") on top of the CAN bus, offering diagnostic capability for vehicle owners, repair technicians, and also auto insurance companies, such as monitoring the speed and fuel of an automobile. Since 1996, it has been mandated on each gasoline automobile by the US government [2]. Nowadays, most of the vehicles have a diagnostic port installed under the steering wheel, which connects to the CAN bus and delivers CAN bus messages. As a message-based protocol, special OBD-II messages are defined to convey diagnostic information, which are known as the OBD-II Parameter IDs (PIDs) [12]. Unlike the highly customized CAN bus messages which are defined by specific vehicle manufacturers [13], these OBD-II PIDs are standardized. Moreover, as a kind of CAN bus message, PID has the similar structure as shown in Figure 1. Each PID for query uses `0x7DF` as identifier, and the data field contains a service number and a PID number [14]. In addition to these universal PIDs, manufacturers may also define private PIDs. Based on the OBD-II standard, a great number of OBD-II dongles are developed for car diagnosis such as monitoring the speed, fuel and engine status. After plugged into the OBD-II port, these dongles can constantly send CAN bus messages to query diagnostic data from the CAN bus.

Among all available OBD-II dongles on the market today, wireless dongle is the most dominant type, since they provide great convenience while offering a decent price to users [15]. When in use, they serve as end points for nearby mobile devices to connect and communicate via wireless network such as Wi-Fi, Bluetooth Classic, and BLE. As a result, there are also companion mobile apps for these dongles. In this paper, our vulnerability analysis focuses on wireless dongles, since they allow wireless access to the OBD-II port and thus are more realistic targets for attackers. Figure 2 shows the 77 wireless dongles we purchased in this research.

Figure 2: All 77 OBD-II Dongles in Our Study.

Companion mobile apps. Since wireless OBD-II dongles do not have user interfaces such as screen and keyboard, they rely on external devices to make them usable and user-friendly. As a result, their manufacturers or third-party developers have developed corresponding companion mobile apps. With such a companion app at hand, it is easy and convenient to monitor the status of an automobile. First, a user plugs the dongle into the OBD-II port locating under the steering wheel, and starts the engine. Second, she opens the companion mobile app and establishes a connection via the wireless network hosted by the dongle. Afterwards, the user is able to monitor the vehicle status from the app UI, and the app automatically interacts with the dongle which queries the vehicle status data from the CAN bus and delivers vehicle control commands if there is any to the CAN bus.

3 Attack Model and Attack Surface

3.1 Attack Model

In this paper, the attacker's goal we consider is to exploit the new vehicle attack surface exposed by emerging wireless OBD-II dongles and thus achieves *wireless* attacks onto the CAN bus of a victim vehicle. As introduced in §2, the wireless OBD-II dongles operate as wireless end points for surrounding devices to connect and communicate. As a result, the attacker must be within the range of the wireless network so that she is able to establish a connection with the target dongle, which is usually up to 100 meters. However, with an amplifier [16], an attacker can detect wireless signals at a remote distance (e.g., up to 1,000 meters using the BLE Antenna as demonstrated in BleScope [17]), which thus enables her to discover and approach the victim to perform attacks. The general threat model is presented in Figure 3. Before the attack, the first but very important step is to identify a nearby OBD-II dongle in the air. Afterwards, she tries to initiate a connection with it through wireless network. If the connection is successfully established, she then attempts to attack the ve-

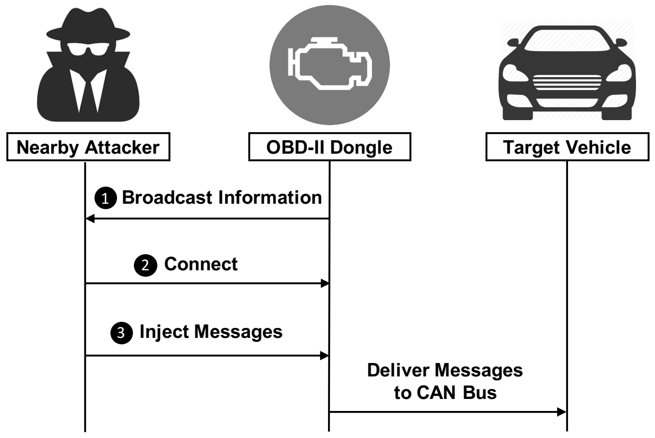

Figure 3: The General Threat Model.

hicle by injecting malicious messages to the CAN bus through the OBD-II dongle, e.g., reading sensitive data or causing unsafe vehicle driving behaviours. The attack is successful if the messages are successfully delivered by the dongle to the CAN bus and the corresponding attack consequences are triggered.

There is no specific constraint of when to conduct the attack. For example, an attacker may compromise vehicles that are still driving on the road with the driver and other passengers on board. Moreover, since an OBD-II dongle can still receive power supply from the OBD-II port even when the vehicle is off [18], she can even target the vehicles parked in a parking lot where she has a chance to sneak into the vehicle and steals all belongings. Some CAN bus messages with more direct safety consequences, e.g., stopping the engine, will be disabled when the vehicle is moving at a high speed [19]. Thus, if these messages are required for achieving a certain attack goal (which is not necessary for all potential attack goals as discussed later in §6), the attacker can choose to launch the attack during common low-speed driving scenarios such as those when waiting at red light, in a traffic jam, or in a drive-thru queue.

3.2 Attack Surface

Before we design our automated security analysis tool, it is necessary to first comprehensively identify the attack surface for these wireless OBD-II dongles. According to our attack model in §3.1, a successful attack must have the following three necessary stages: (I) *Broadcast Stage*, i.e., when the attacker is scanning for victim dongles before connection, (II) *Connection Stage*, i.e., when the attacker is connecting with the dongle, and (III) *Communication Stage*, i.e., when the attacker is injecting malicious messages after connection. Thus, the attack surface considered in our analysis is defined as attack vectors at each of these three stages:

(I) Broadcast Stage. Prior to connection, a wireless OBD-II dongle broadcasts its connection information to nearby devices to indicate its willingness. Therefore, nearby devices can discover and recognize it, and try to establish a connection. As a nearby attacker, she is capable of collecting this broadcast information, and her goal is to identify a victim dongle and establish connection based on the information.

(II) Connection Stage. In this stage, the attacker's task is to successfully establish a network connection with the dongle in order to send commands related to CAN bus message delivery to the dongle. When the connection is establishing, she may be required to provide sufficient credential before legitimately connecting to the dongle, such as a password or a PIN code. If no credential is needed, the attacker is able to arbitrarily connect to the dongle.

(III) Communication Stage. After the connection is established, the attacker is able to send unauthorized CAN bus messages to communicate with the dongle and perform attacks. Prior to that, she may need to first bypass the authentication step in the communication protocol between the app and the dongle. After that, the attacker sends the attacker-desired CAN bus messages to the dongle, requesting it to relay the messages to the CAN bus to trigger corresponding consequences. In this paper, we call the CAN bus messages that perform the designed functions of a dongle *predefined messages* and the others *undefined messages*. The former is allowed by design, and thus should be directly relayed to the CAN bus. However, the designed dongle functions can be quite limited, e.g., only diagnostic functions. Therefore, for certain attack goals, e.g., those that are safety related, it is of interest to inject undefined messages (e.g., those interfere the vehicle control). However, since these messages are not allowed by design, whether the attack can succeed depends on the message filtering process at the dongle side.

4 Analysis Methodology

Having identified the attack surface of OBD-II dongles, we design and implement an automated tool, named DONGLESCOPE[1], to measure a few objectives that can lead to vulnerabilities in wireless OBD-II dongles across broadcast, connection and communication stages. In this section, we first introduce the design overview, as well as the measurement objectives for each stage in §4.1. We then detail the design and implementation of our tool in §4.2.

4.1 Overview

Figure 4 presents the workflow of DONGLESCOPE. At a high level, it dynamically tests an OBD-II dongle on a real automobile, and also takes the corresponding companion mobile app for static analysis. The analysis is broken down into four main components associated with the three attack stages. For

[1]The source code of DONGLESCOPE is available at https://github.com/OSUSecLab/DongleScope.

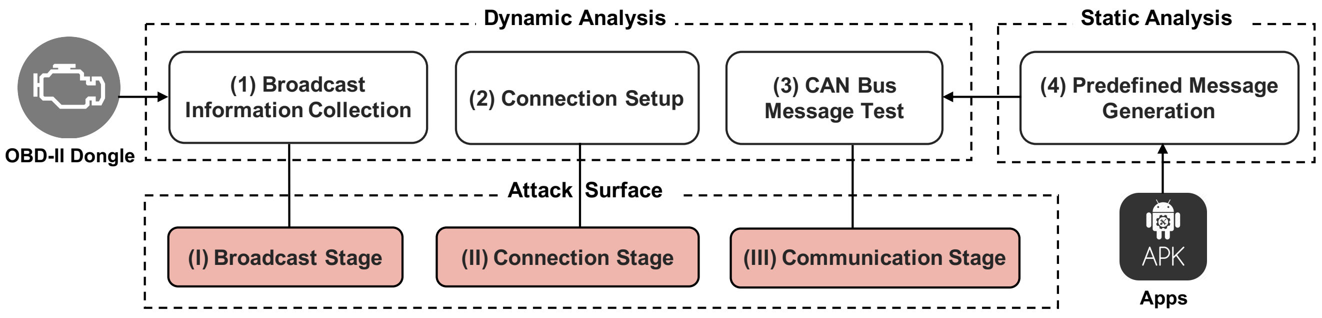

Figure 4: Design Overview of DONGLESCOPE.

Component	Measurement Objective(s)
(1) Broadcast Information Collection	① Broadcast information
(2) Connection Setup	② If connection can be successfully established. ③ If multiple access is allowed.
(3) CAN Bus Message Test	④ If predefined message is injected to CAN bus. ⑤ If undefined message is injected to CAN bus.

Table 1: Measurement Objectives of DONGLESCOPE.

each component in dynamic analysis, specific measurement objectives are defined, as shown in Table 1. During the broadcast stage, DONGLESCOPE collects all broadcast information from the wireless OBD-II dongle. Next, it tries to set up a connection with the dongle at the connection stage. Note that the tool also tests whether the connection is still successful while another mobile device is connected with the dongle, which simulates a real attack scenario (i.e., when the driver is still inside the vehicle). In this stage, there are two measurement objectives: if connection can be established and if multiple connections are allowed. After a connection is established, it tries to test with predefined and undefined CAN bus messages at the communication stage. The objective is to check if the predefined and undefined messages can be injected to the CAN bus so that corresponding attack consequences can be observed. Meanwhile, the predefined message generation step produces predefined messages to help design the messages for the CAN bus message test.

Prior to our design, there are a few challenges to be solved. First, since OBD-II dongles can adopt various manufacture-specific implementations (e.g., different message patterns and protocols), it is hard to come up with a fully generic approach. As a result, we assume all OBD-II dongles are ELM327-based [20], which is a common implementation for interpreting low-level CAN bus protocol and providing standardized interfaces for programming. According to the experiment results in §5.3, over 90% of the dongles in our study are ELM327-based. To achieve dongle configuration and message communication with the CAN bus, we leverage the ELM327 command set [20] to design the testing messages.

Second, it is necessary to find all predefined messages of each dongle so that we can make sure that the undefined messages to be tested are not predefined in the dongle. Inspired by previous IoT research which leverages companion mobile app analysis to understand black box IoT devices [21,22], we introduce a predefined message generation step using backward program slicing to extract all predefined messages of the tested dongles.

Third, we have to obtain undefined messages that are valid to the CAN bus so that we are able to observe effects brought by the injection of the messages. Intuitively, we should use the control CAN bus messages, because OBD-II dongles should not provide vehicle control capabilities by design. Therefore, we need to reverse engineer the CAN bus protocol, which has long been a tricky but valuable task for automotive researchers since protocols across different manufactures are highly customized but confidential [10, 13, 23]. The state-of-the-art for reverse engineering the CAN bus protocol is through CAN message fuzzing or manually triggering physical vehicle actions [10, 24]. Inspired by them, we also tried to analyze the CAN bus protocol on our testing automobile. Specifically, we first operated the vehicle with some physical actions (e.g., step on the throttle, apply the brake), and then observed the changes on the dynamic CAN bus traffic to see which CAN bus message led to the behavior. Therefore, we are able to obtain a number of control CAN bus messages on our testing vehicle.

In the experiment, we select a representative from the undefined messages for testing since it is unrealistic to test them all, given there are at least 2^{75} possible CAN bus messages in theory. Additionally, we assume the filtering policy is based on message format, since predefined OBD-II PIDs have distinct identifiers compared with other messages. In other words, if any message of a specific format can pass the filter, then all messages with the same format can also pass the filter, and vice versa. To verify this hypothesis, we conducted an experiment on 26 dongles by injecting 10 different undefined messages. We observed that these messages were all either accepted or filtered by each dongle, which confirms our assumption. Therefore, we can narrow the testing undefined messages to just one representative. Furthermore, testing one predefined message is sufficient since all predefined messages should be accepted by design, which has also been verified by similar experiments.

4.2 Detailed Design and Implementation

(1) Broadcast Information Collection. We categorize the OBD-II dongles into three types: Wi-Fi, Bluetooth Classic, and BLE, according to their connections. Since broadcast information varies across different types of dongles, DONGLESCOPE deals with them correspondingly. In this step, we first manually plug the dongle into the OBD-II port. Next, DONGLESCOPE starts to automatically collect necessary broadcast information from it, and the information is stored in a configuration JSON file. To summarize, all sniffable broadcast information includes Wi-Fi service set identifier (SSID), device name of Bluetooth Classic and BLE, as well as the universally unique identifier (UUID) of BLE dongles, etc. Note that to make sure the collected broadcast information is from the dongle, there should be no other broadcasting devices around when we perform the test.

(2) Connection Setup. After identifying an OBD-II dongle in the broadcast stage, DONGLESCOPE tries to establish a connection with it for further communication. During the connection, we simulate a real attack scenario by setting up a mobile device connected with the dongle, which acts as the driver's device. If DONGLESCOPE fails to connect with the dongle, implying that multiple connections are not allowed, we disconnect the driver's device with the dongle and try a single connection. When the system-layer connection is established, DONGLESCOPE tries to set up a communication channel on app-layer with the dongle. To achieve this, some additional information from the specifications is needed (e.g., IP address, port number), which is pre-loaded into DONGLESCOPE. For a Wi-Fi dongle, DONGLESCOPE follows the IP address as well as the port number to build up a socket for communication. For a Bluetooth Classic dongle, DONGLESCOPE first queries the Bluetooth address and port number from the dongle, and tries to setup a Bluetooth socket with it based on the Radio Frequency Communication (RFCOMM) protocol [25]. As for a BLE dongle, the process is more complicated, since it requires DONGLESCOPE to obtain the read and write characteristics which are necessary for communication with the dongle. These characteristics are the attributes in BLE devices conveying concrete data and can be identified by UUIDs [26]. Our solution is to inject an ELM327 command `AT E0` to each characteristic at a time to check which other characteristic echos an `OK` back. These two characteristics are regarded as the write and read characteristic respectively.

After the connection is established, DONGLESCOPE is able to communicate with the dongle through the ELM327 interface. We implement the communication process with the Python socket library [27], PyBluez [28] and PyBLE [29]. Prior to that, DONGLESCOPE still needs to configure the dongle, otherwise it may not get a valid response. Specifically, it injects the following ELM327 commands [20] to achieve the corresponding configuration purposes:

- **AT D**. Restore the dongle to its default setting.
- **AT E0**. Stop the messages from echoing.
- **AT AT0**. Disable timeout.
- **AT H1**. Show the message header in response.
- **AT CAF1**. Turn off the auto formatting.
- **AT SP 6**. Set the ISO 15765-4 CAN protocol as default (using 11-bit identifier).

(3) CAN Bus Message Test. When the connection is successfully set up, DONGLESCOPE is able to send messages to the CAN bus through the OBD-II dongle. In this step, DONGLESCOPE tests two representatives from the predefined and undefined CAN bus messages respectively, since testing all the predefined and undefined messages is unnecessary (discussed in §4.1). Specifically, DONGLESCOPE adopts a standard PID `09 02` as the testing predefined message, which is a diagnostic message to query the VIN. As for the undefined message, a CAN bus message `191 04 00 00` is used which sets the transmission gear to N in our testing vechicle. This undefined message is obtained through reverse engineering the CAN bus protocol. During the analysis, we used an `ATMA` command to dump the CAN bus traffic, and shifted the gear to different positions. During this process, we observed the changes on the CAN bus and determined the message that triggered the behaviour. After we obtained the undefined message, we cross-checked it with our app analysis results and made sure that it is not predefined for all dongles in our study.

Having obtained the messages of interest, DONGLESCOPE starts to automatically test them on the dongle. Specifically, DONGLESCOPE specifies the headers for the CAN bus messages to be sent with an `AT SH` command, and specifies the respond message header with an `AT CRA` command. First, DONGLESCOPE tests with the predefined message `09 02`. A successful query will return back a valid VIN number in hexadecimal form. Second, DONGLESCOPE sends an undefined message `191 04 00 00`. If successfully delivered by the dongle to the CAN bus, the ECU will echo a CAN bus message with the same identifier `0x191` showing its status. Otherwise, the message will be filtered by the dongle, and the tool will get a `NO DATA` response.

(4) Predefined Message Generation. In order to design the testing messages in the communication stage, DONGLESCOPE performs static analysis on the corresponding companion mobile app to generate the predefined messages. Specifically, it uses backward program slicing [30], which is a technique to obtain the program slices that are necessary for generating the target data. To start the analysis, we first

Algorithm 1: Backward Slicing Algorithm

```
Input: $G$: Control flow graph of current function, $V$: Variable set of our interest
Output: $P$: A set of data generation paths
1  $P \leftarrow \emptyset$ ;
2  $path \leftarrow \emptyset$ ;
3  $E \leftarrow$ Sub-graph of $G$ ending in the target APIs ;
4  for $edge(i, j) \in$ backward DFS order of $E$ do
5      $l \leftarrow$ left operand of $i$ ;
6      if $l \in V$ then
7          $V \leftarrow V \cup$ right variable operands of $i$ ;
8          if $i$ is a library function then
9              $path \leftarrow path \cup i$ ;
10         else
11             Dive into the implementation of $i$ for further slicing;
12         end
13     end
14     if No descendent edge then
15         $P \leftarrow P \cup path$ ;
16         Restore $path$ to the state of the latest branch point;
17         Restore $V$ to the state of the latest branch point;
18     end
19 end
```

identify the low-level network target APIs, including the TCP socket send function, as well as write functions of Bluetooth Classic and BLE. According to our observation, they are the only ways for a companion app to communicate with an OBD-II dongle to perform the designed functions. As a result, the variables in these APIs denote the messages sent to the dongle. We then design a backward slicing algorithm which starts from these identified APIs and iterates backward to record the necessary instructions that generate the messages, which is detailed in algorithm 1.

At a high level, the algorithm takes the control flow graph of the program (G) as well as a set of variables of our interest (V) as input, and produces a set of generation paths (P). First, it initializes the set of generation paths (P) and the temporary path ($path$) as empty (line 1-2). Next, it constructs E as a sub-graph of G where all leave nodes are the target APIs, and traverses E in backward DFS order (line 3-4). For each edge (i, j) where each node of it indicates an instruction, the algorithm detects if the left operand of node i is in V (line 6). If it is, the algorithm adds all variable operands on the right of i to V (line 7). Note that when i is not a library function, the algorithm needs to dive into the implementation of i and continue slicing; otherwise it adds the instruction i to the temporary path $path$ (line 8-12). Afterwards, when the current path reaches the end, the algorithm adds $path$ to P (line 14-15). Then, it starts traversing another path and restores the current $path$ and V to the state of the latest branch point (line 16-18). Ultimately, the algorithm outputs a number of data generation paths which contain instructions that generate the data of our interest. Based on the generation paths, DONGLESCOPE performs forward computation to reconstruct the actual value of the data being sent. The static analysis is built atop Soot [31], which is a popular static analysis framework for reverse engineering Android mobile apps.

Dongle Name	Type	App-specific?	# Review	Vulnerable?
BAFX OBD Reader	Wi-Fi		11,523	✓
BlueDriver Pro	Bluetooth	✓	3,764	✓
FIXD	BLE	✓	3,229	✓
VEEPEAK VP01 WIFI	Wi-Fi		1,571	✓
Veepeak Mini	Wi-Fi		1,505	✓
iSaddle WIFI OBD2	Wi-Fi		1,094	✓
Carista	BLE	✓	1,044	✓
GXG-1987 OBD-II Mini	Wi-Fi		799	✓
wsilroon Car WIFI OBD 2	Wi-Fi		708	✓
PLX Devices Kiwi 3	BLE		640	✓

Table 2: Top 10 Popular Dongles in Our Study.

5 Vulnerability Analysis

5.1 OBD-II Dongle and App Collection

OBD-II dongles. To achieve high comprehensiveness of our study, we bought *all* wireless OBD-II dongles available in the US from Amazon by searching combinations of all possible related keywords (i.e., "OBD-II" or "OBD2" or "OBDII" combining with "dongle" or "scanner" or "adapter") in February 2019. In total, this collection ended up with 77 OBD-II dongles (depicted in Figure 2). Among these 77 dongles, there are 44 (57.14%) Wi-Fi dongles, 3 (3.90%) Bluetooth Classic dongles and 30 (38.96%) BLE dongles, which shows Wi-Fi and BLE based dongles are the most popular ones on the market today.

The full list of dongles is shown in Table 3. To estimate the popularity of these dongles, we measure their review counts on Amazon. In Table 2, we present the top 10 most popular dongles based on the number of reviews in Amazon. As shown, the most popular one is a Wi-Fi dongle with over 10,000 reviews. This dongle provides basic diagnostic functions and are compatible with a large number of free third-party companion mobile apps. Similar dongles such as VEEPEAK, iSaddle, and GXG, are also very popular. In addition, some dongles such as BlueDriver, FIXD and Carista are app-specific, which are also popular as advanced diagnostic functions are provided. As shown in our experiment results later, DONGLESCOPE discovers that *all* of the top 10 most popular dongles contain at least two vulnerabilities, which suggests that majority of the vehicles with OBD-II dongles today are vulnerable to attacks.

Companion mobile app. Since DONGLESCOPE also involves the analysis of the companion apps, we also collect them from Google Play according to the dongles' specifications. In total, we collected 21 companion apps that can be mapped to all 77 OBD-II dongles in our experiment. We show all these 21 apps in Table 4. As indicated, the top six companion apps are downloaded over millions of times on Google Play, which implies their popularity among mobile app users. Moreover, we investigate whether these apps are designed for specific dongles, or *dongle-specific*. As shown in the table, over half of the companion apps are dongle-specific such as

Dongle Name	Type	Companion Mobile App	App-specific?	Vulnerability						Special Message
				V1.1	V1.2	V2	V3	V4	V5	
OBDLink MX	Wi-Fi	OBDLink			✓		✓		✓	
Automatic Pro	BLE	Automatic	✓	✓					✓	② ③
Innova 3211aDrive	BLE	RepairSolutions	✓	✓	✓			✓	✓	② ③
BlueDriver Pro	Bluetooth	BlueDriver	✓	◐	✓			✓	✓	② ③
HaulGauge OBD-II Connector	BLE	HaulGauge	✓	✓	✓				✓	②
PLX Devices Kiwi3	BLE	Kiwi OBD		✓	✓		✓		✓	
Carly WiFi GEN2	Wi-Fi	Carly for Toyota	✓	✓	✓		✓	✓	✓	③
OBDLink MX+	Bluetooth	OBDLink			✓		✓		✓	
JDiag AutoCar	BLE	FastLink M2	✓	✓	✓		✓		✓	
TT TOPDON Scanner Artibox	BLE	ArtiBox	✓	✓	✓		✓		✓	
nonda ZUS Smart Vehicle Health Monitor	BLE	ZUS-Smart Driving Assistant	✓	✓	✓		✓		✓	
JDiag Faslink M2 OBD2 Scanner	BLE	ArtiBox	✓	✓	✓		✓		✓	
JDIAG Bluetooth OBD2 Scanner	BLE	FastLink M2	✓	✓	✓		✓		✓	
Joaruy OBD2 Scanner	Wi-Fi	ArtiBox		✓	✓	✓	✓			
OBD2 Scanner Bluetooth 4.0	BLE	OBD Fusion		✓	✓		✓			
JDIAG Bluetooth Car Scanner	BLE	FastLink M2	✓	✓	✓		✓		✓	
LELink Bluetooth Low Energy BLE OBD-II	BLE	CarScanner		✓	✓		✓		✓	
Veepeak OBDCheck BLE OBD2 Scanner	BLE	OBD Fusion		✓	✓		✓			
Vgate iCar Pro BLE OBD2	BLE	OBD Auto Doctor		✓	✓		✓		✓	
OHP WiFi ELM327 Forscan OBD2 Adapter	Wi-Fi	Torque Lite	✓	✓	✓	✓				
DODYMPS OBD-II Scanner	BLE	DODYMPS	✓	✓	✓				✓	②
TONWON Car Bluetooth 4.0 OBD Code Readers	BLE	OBD Auto Doctor		✓	✓		✓		✓	
IKKEGOL iCar2 Mini OBD2	Wi-Fi	OBD Auto Doctor		✓	✓	✓	✓		✓	
BAFX OBD Reader	Wi-Fi	OBD Fusion		✓	✓	✓	✓		✓	
Vgate iCar 3 Wi-Fi	Wi-Fi	OBD Auto Doctor		✓	✓	✓	✓		✓	
Vgate iCar 2 Wi-Fi	Wi-Fi	OBD Auto Doctor		✓	✓	✓	✓		✓	
Washinglee WiFi OBD2 Scanner	Wi-Fi	EOBD Facile		✓	✓	✓	✓			
OBD2 Scanner OBD2 WiFi Adapter	Wi-Fi	OBD Auto Doctor		✓	✓	✓				
TONWON Car Bluetooth OBD2 Scan Tool	BLE	OBD Auto Doctor		✓	✓		✓		✓	
Juta α-Driver Bluetooth 4.0 OBD2 Scanner	BLE	Torque Lite		✓	✓		✓		✓	
iSaddle WiFi OBD2	Wi-Fi	Dash Command		✓	✓	✓				
X-ELM Elm327 Diagnostic Scanner	BLE	Car Scanner		✓	✓		✓		✓	
Vgate Wifi Auto Sleep	Wi-Fi	Dash Command		✓	✓	✓	✓		✓	
TOPDON Automate WiFi OBD2 Scanner	Wi-Fi	Automate		✓	✓	✓	✓			
TekkPerry Bluetooth OBD2 Scanner	BLE	Torque Lite		✓	✓				✓	
Keenso ELM327 WiFi OBDII Scanner	Wi-Fi	Dash Command		✓	✓					
FOXWELL FW601 Obd2 Scanner	Wi-Fi	Torque Lite		✓	✓	✓	✓			
XTOOL iOBD2 Mini Auto Scanner	BLE	iOBD2	✓	✓	✓		✓		✓	
Bluetooth 4.0 OBD2 Scanner	BLE	Dash Command		✓	✓		✓		✓	
Oummit OBD2 Scanner	Wi-Fi	Torque Lite			✓	✓	✓			
K-Cliffs OBD2 Car Code Reader	Wi-Fi	Dash Command		✓	✓	✓				
Auwell WiFi OBD2 Scanner	Wi-Fi	Dash Command		✓	✓					
AquaNine OBD2 Car Diagnostic Scanner	Wi-Fi	Dash Command		✓	✓	✓	✓			
KINGBOLEN WiFi OBD2 Scanner	Wi-Fi	Car Scanner		✓	✓	✓	✓			
NiceAndGreat OBD2 Car Code Reader	Wi-Fi	Dash Command		✓	✓					
TOPDON Auto Mate	BLE	Automate			✓		✓			
Launchh OBDII Auto Diagnostic Scanner	Wi-Fi	OBD Fusion		✓	✓					
Rapify OBD2 Scanner	Wi-Fi	OBD Auto Doctor		✓	✓	✓				
Kitbest OBD2 Scanner	Wi-Fi	OBD Fusion		✓	✓		✓			
Cllena Car WiFi OBD2 Scan Tool	Wi-Fi	OBD Auto Doctor		✓	✓	✓				
SaiDent KW903 OBDII Fault Code Scanner	BLE	Dash Command		✓	✓		✓		✓	
Veepeak Mini WiFi OBD2 Scanner	Wi-Fi	OBD Fusion		✓	✓	✓	✓			
V COOL OBD2 Scanner	Wi-Fi	OBD Auto Doctor		✓	✓	✓	✓			
Panlong WiFi OBD2 Scanner	Wi-Fi	OBD Fusion		✓	✓	✓	✓			
LJPXHHU Bluetooth OBD2 Diagnostic Scanner	BLE	OBD Auto Doctor		✓	✓		✓			
OBDII Scanner TOPDON	BLE	Torque Lite			✓		✓			
wsilroon Car WiFi OBD2 Scan Tool	Wi-Fi	OBD Auto Doctor		✓	✓	✓	✓			
Tu2Codez OBD Car Scanner	Wi-Fi	Dash Command		✓	✓		✓			
LJPXHHU Car WiFi OBD2 Scan Tool	Wi-Fi	Torque Lite		✓	✓					
GXG-1987 OBD-II Mini	Wi-Fi	Torque Lite		✓	✓					
RoverOne Super MINI v2.1 OBD2 Scanner	Bluetooth	Torque Lite		✓	✓				✓	
TOPTON Automate Code Reader	BLE	Automate			✓		✓			
Joaruy WiFi OBD2 Scanner	Wi-Fi	Dash Command		✓	✓	✓	✓			
Elm327 WiFi OBDII Interface OBD2 Scanner	Wi-Fi	Dash Command		✓	✓	✓				
ATDIAG Car WiFi OBD2 Scanner	Wi-Fi	OBD Auto Doctor		✓	✓		✓			
ZENHOX WiFi OBD2 Scanner	Wi-Fi	Torque Lite		✓	✓	✓				
Friencity Car WiFi OBD2 Scanner	Wi-Fi	OBD Fusion		✓	✓					
Car ELM327 Wifi OBD2 Code Reader	Wi-Fi	Torque Lite		✓	✓					
Audew Car WiFi OBDII Reader	Wi-Fi	Torque Lite		✓	✓	✓	✓			
Best OBD2 Scanner	Wi-Fi	Torque Lite		✓	✓					
EDIAG WiFi OBD2 Diagnostic Scanner	Wi-Fi	OBD Auto Doctor		✓	✓	✓	✓			
Jevogh V01HW OBD2 Scanner	Wi-Fi	Dash Command		✓	✓	✓	✓			
Giveet Car WiFi	Wi-Fi	Torque Lite		✓	✓	✓	✓			
Carista	BLE	Carista	✓	✓	✓		✓		✓	①
FIXD	BLE	FIXD	✓	✓	✓		✓		✓	
VEEPEAK VP01 WiFi	Wi-Fi	Torque Lite		✓	✓		✓			
VEEPEAK BLE	BLE	Torque Lite		✓	✓					

Table 3: Vulnerability Analysis Results and Special Messages of 77 OBD-II Dongles (Note that ① Non-diagnostic CAN bus message, ② Private command, ③ Firmware image of dongle, ◐ means vulnerable but with tight constraint for exploitation).

BlueDriver, Carly for Toyota and FIXD, providing advanced diagnostic and even remote-control functions. However, the most popular apps (e.g. Torque Lite) are developed by third parties for ELM327-based dongles, and thus they are compatible with most of the dongles we purchased.

5.2 Experiment Setup

Our dynamic analysis is conducted on a 2015 Honda Civic automobile in an empty parking lot to avoid unpredictable accident. It has a standard OBD-II port locating under the steering wheel, which allows us to plug in an OBD-II dongle for testing. The dynamic analysis part of DONGLESCOPE is implemented and deployed on a MacBook Pro laptop, with six Intel Core i7-8850H CPUs (2.6 GHz) and 16 GB RAM, running 10.14.5 macOS Mojave. Our static companion app analysis ran on a Linux server running Ubuntu 16.04 equipped by twelve Intel Core i7-8700 (3.2 GHz) CPUs and 32 GB RAM.

5.3 Vulnerability Analysis Results

After properly setting up our experiment environment, we applied DONGLESCOPE to the 77 OBD-II dongles as well as 21 companion mobile apps. Although previous security analysis on OBD-II dongles has revealed insufficient application-layer authentication vulnerability and security holes in message filtering [3], our analysis is more comprehensive in that we not only discover new vulnerabilities but also quantify them among the dongles on the market. The complete assessment results of all the dongles and companion apps are presented in Table 3 and Table 4. In summary, we discover 5 types of general vulnerabilities (with 4 being newly discovered) across broadcast, connection and communication stages, and find that *all* the dongles are vulnerable to at least two of these vulnerabilities, which shows a widespread vulnerability exposure in this new over-the-air vehicle attack surface today. In the following, we present the detailed results based on each stage of the analysis.

5.3.1 Connection Stage

V1. The majority (84.16%) of the dongles has neither connection-layer nor application-layer authentication. Our experiment results show that lack of authentication at the connection stage widely exists among OBD-II dongles, which can be further classified to the lack of authentication at the connection layer and the application layer. The former leads to arbitrary nearby connection since one can establish a connection without providing any credentials. Based on the established connection, the latter further enables any unauthorized users to communicate with the dongle, which essentially allows unauthorized access to the CAN bus. In summary, we find that *84.16% of the dongles have neither connection-layer or application-layer authentication, which provides an attacker arbitrary access to the CAN bus once discovered by the attacker in the broadcast stage.*

V1.1. Nearly all (92.21%) dongles have no connection-layer authentication by default. At the connection stage, our tool reported that *71 (92.21%) dongles can be arbitrarily connected by nearby devices* while only 6 (7.79%) dongles require authentication before connection, which implies weak protections on the connection layer among these dongles. With this vulnerability, an attacker can perform denial-of-service attack by simply keeping connected with the target dongle. We further discovered that there are two ways for these 6 dongles to implement connection-layer authentication, which is summarized in Table 5. As shown, OBDLink MX adopts WPA2-PSK to authenticate their connection, which requires users to enter a password. The entered credential then generates a cryptographic key to establish a secure communication channel to prevent eavesdropping attack. The other way to implement authentication is through button. Specifically, a user needs to press a physical button on the dongle to let it enter a discoverable mode so that external devices can scan and connect to it. This mode can last for approximately a few minutes only and the time window for connection is quite narrow. Among the 6 dongles, 5 of them have applied this implementation. Interestingly, we also discover one special case that the BlueDriver dongle will force itself to sleep after 60 seconds if no connection is established, which significantly narrows the time window for attacks.

V1.2. Only 1 out of 77 dongles has application-layer authentication by default. After connection is established, DONGLESCOPE tried to directly inject CAN bus messages to communicate with the dongle, and successfully queried the VIN from 71 (92.21%) dongles, which indicates that they have no application-layer authentication by default. We further manually investigated the companion apps of the remaining 6 dongles to understand the root cause of the injection failure. Surprisingly, we find that only 1 dongle (namely Automatic Pro) has implemented application-layer authentication. The rest 5 dongles, including BlueDriver, HaulGauge, Innova, DODYMPS, and OHP Forscan, also have no application-layer authentication, since they are not ELM327-based and only accept private manufacture-specific commands to perform vehicle diagnosis. In this paper we call these commands *private commands*, which can be found in the results of the predefined message generation from their companion apps. Interestingly, we also discover that the developers of Automatic Pro have defined private commands to query VIN or read parameters from the vehicle. The detailed content of these commands is presented in Table 6. Our results reveal that the private commands are usually human-readable strings or numbers, as shown in the last column of the table. These private commands are then interpreted by the dongles into CAN bus messages and relayed to the CAN bus. Note that these 5 dongles also lack connection-layer authentication, and thus nearby attackers can

App Name	Category	# Download	Dongle-specific?	Analysis Result
Torque Lite (OBD2 & Car)	Communication	5,000,000		①
DashCommand (OBD ELM App)	Communication	1,000,000		①
EOBD Facile - OBD 2 Car Diagstic for elm327 Wifi	Auto & Vehicles	1,000,000		①
ScanMaster for ELM327 OBD-2 ScanTool	Communication	1,000,000		①
Car Scanner ELM OBD2	Auto & Vehicles	1,000,000		①
OBDLink (OBD car diagstics)	Communication	1,000,000	✓	①
BlueDriver OBD2 Scan Tool	Auto & Vehicles	500,000	✓	① ③ ④
OBD Auto Doctor	Auto & Vehicles	500,000		①
Carly for Toyota	Auto & Vehicles	100,000	✓	① ④
FIXD - Vehicle Health Monitor	Auto & Vehicles	100,000	✓	①
Carista OBD2	Auto & Vehicles	100,000	✓	① ② ④
ZUS - Smart Driving Assistant	Liftstyle	100,000	✓	①
Automatic	Liftstyle	50,000	✓	③
RepairSolutions	Auto & Vehicles	10,000	✓	③ ④
OBD Fusion	Communication	10,000		①
Kiwi OBD	Tools	5,000	✓	①
Automate	Tools	1,000	✓	①
HaulGauge	Auto & Vehicles	500	✓	③
ArtiBox	Tools	500	✓	①
JDiag FasLink M2	Auto & Vehicles	100	✓	①
DODYMPS	Tools	100	✓	①

Table 4: Measurement and Analysis Results of 21 Companion Mobile Apps. (① Standard diagnostic PID, ② Non-diagnostic CAN bus message, ③ Private command, ④ Firmware image of dongle)

Dongle Name	Type	Authentication
OBDLink MX	Wi-Fi	Password
Oummit OBD2 Scanner	Wi-Fi	Button
OBDLink MX+	Bluetooth	Button
TOPDON Auto Mate	BLE	Button
OBDII Scanner TOPDON	BLE	Button
TOPDON AutoMate Code Reader	BLE	Button

Table 5: Connection Layer Authentication on Dongles.

still leverage these private commands to launch attacks. In summary, *for all but one dongle, attackers can directly get access to the CAN bus right after the connection is established.*

The only dongle, Automatic Pro, has demonstrated a way to implement authentication on application-layer. Specifically, prior to the communication stage, it requires users to manually enter a PIN code which is a random 6-digit length string printed on the dongle. However, one can still break the PIN code with a brute-force attack. Afterwards, it validates the PIN and leverages it to create a cryptographic key for communicating with the authorized user. When a message is transferred, it must be encrypted by the key and sent through the secure channel.

V2. 29 (37.66%) dongles can allow unauthorized access even when the vehicle owner's mobile device is connected. For most dongles, they only allow one mobile device to be connected at a time, which is expected since the most popular usage for these dongles is to exclusively connect with the companion app on the vehicle owner's smartphone. Surprisingly, we find that some Wi-Fi dongles are configured to allow multiple device connections and accesses. These multiple connected devices are treated equally at the communication stage, which implies that an attacker can attack these dongles even when the vehicle owner's device is connected. Among the 49 Wi-Fi dongles in our study, 29 (37.66%) of them are found to allow multiple device accesses. Additionally, only 1 out of these 29 dongles has implemented authentication, which means that over half (i.e., 28) of the Wi-Fi dongles can be attacked even when the vehicle owner's smartphone is connected.

5.3.2 Communication Stage

V3. The majority (67.53%) of the dongles fails to filter out undefined CAN bus messages (known but not quantified before [3]). At the communication stage, DONGLESCOPE tried to send an undefined CAN bus message which should not be accepted by the dongle and delivered to the CAN bus. However, our result reveals that *52 (67.53%) dongles fail to filter out this undefined CAN bus message*, which implies that they are vulnerable to undefined CAN bus message injection. An instance of such lack of filter has been discovered before on a Bosch dongle that is not available on the market today [3], and our study is the first to measure the prevalence of such vulnerability among a comprehensive set of dongles available today. On the contrary, DONGLESCOPE confirms that only 24 (31.17%) dongles (including 5 dongles that use private commands) recognize the undefined message and prevent it from being delivered to the CAN bus. As for the remaining 1 dongle (i.e., Automatic Pro), our tool is unable to determine whether it can filter the undefined CAN bus message or not due to the application-layer authentication.

We also discovered from the set of predefined messages that 2 dongles (i.e., Carista and Automatic Pro) also support non-diagnostic capabilities in addition to diagnostic functions.

Dongle Name	**Type**	**Connection Auth.?**	**Implementation**	**Private Commands**
Automatic Pro	BLE	No	PIN, Private commands	`IGN`, `VIN_STRING`, `DEVID`, `obd_protocol`
BlueDriver Pro	BLE	No	Private commands	`LMIF0`, `LMIF1`, `ATIF1`, `LMH0`
HaulGauge OBD-II Connector	BLE	No	Private commands	`4` (checkHardwareVersion)
Innova 3211a Drive	BLE	No	Private commands	`16557` (readVIN), `-1895767379` (BootLoader)
DODYMPS OBD-II Scanner	BLE	No	Private commands	`AA000B5000010001000A00005155`
OHP Forscan OBD2 Adapter	Wi-Fi	No	Private commands	`020000`, `020300`, `020400`, `020600`

Table 6: Application Layer Authentication and Private Commands on Dongles.

Dongle Name	**Companion App**	**Vulnerable?**	**Firmware Available?**
Automatic Pro	Automatic		
Carly WiFi GEN2	Carly for Toyota	✓	✓
BlueDriver Pro OBDII	BlueDriver		✓
Innova 3211a Drive	RepairSolutions	✓	✓

Table 7: OTA Firmware Subverting and Extraction Vuln.

For example, Carista provides remote control functions such as disable remote door locking, removing seat belt warning, and modifying parking sensor, which affects the control behaviour of an automobile. Since this dongle also does not have any authentication, an attacker is able to extract these valid non-diagnostic CAN bus messages by reverse engineering the companion app and then inject them to launch attack. The other dongle, Automatic Pro, allows tracking of current GPS location with a private command `gps_location`. Fortunately, this dongle has implemented authentication on application layer so that nearby attackers cannot easily inject the corresponding private commands.

V4. Some (3) dongles are vulnerable to over-the-air firmware subverting or extraction. In addition to predefined messages of the OBD-II dongles, we surprisingly found that the outputs of the predefined message generation step also include large blocks of data that are apparently not CAN bus messages. Based on heuristic clues such as keywords "firmware" and "upgrade", we found that these are the firmware images of the OBD-II dongles. Since OBD-II dongles usually do not have cellular network, they rely on the companion mobile apps as gateway to download and transfer their firmware packet over the air to achieve upgrade. In general, a dongle needs to enter a BootLoader mode prior to the upgrade, which is done by sending a specific command from the app to the dongle, such as `AT^` and `AT@BL`. Next, the companion app transfers the firmware packet via wireless network channel to the dongle. Since the upgrade process is initiated by the mobile app, it is possible for an attacker to spoof the dongle and subvert its firmware by transferring a malicious one. As indicated in Table 7, we discover that 4 OBD-II dongles have firmware upgrade capability. Therefore, we further investigated their upgrade process by manually analyzing their companion apps. Our analysis reveals that *3 (75%) of the 4 dongles are vulnerable to firmware subverting or extraction.*

Since subverting the firmware of a dongle requires one to first have access to it, we eliminate those that have authentication. Among the 4 dongles, only Automatic Pro has application-layer authentication and thus is not subjective to the attack. For the rest 3 dongles without any authentication, their firmware is at risk of being subverted. In order to subvert the dongle's firmware, it is necessary to make sure that there is no integrity check on the dongle side. Therefore, we tried to perform the attack by injecting a modified firmware, in which we found that the upgrade process of Carly and Innova accepts arbitrary firmware image. Though the Innova dongle verifies integrity by validating the checksum appended after each message, the algorithm of calculating the checksum can be easily obtained through analyzing the companion apps, which thus enables an attacker to construct a spoofed upgrade message to achieve the attack.

Furthermore, the firmware images of three dongles can even be extracted from their companion apps through reverse engineering. For instance, the BlueDriver app exposes the download URL and authentication credentials of its firmware image. Thus, we successfully downloaded its firmware images of all available versions. Even worse, some apps directly hardcode the firmware images in the app code, including Carly for Toyota and RepairSolutions. Having the access to the dongle firmware, an attacker is then able to discover more vulnerabilities with them such as whether containing any backdoors.

5.3.3 Broadcast Stage

V5. Vulnerability status of nearly half (42.86%) of the dongles can be uniquely identified using broadcast information. Having identified the various vulnerabilities for exploitation, we then analyze whether it is possible to fingerprint the vulnerabilities of these OBD-II dongles in the broadcast stage. This can help an attacker pinpoint which dongles to attack and then attack correspondingly, and such fingerprinting can significantly improve the attack success rate. As shown in Table 8, we aggregate all the dongles by their connection name and show those with the same vulnerability status. In total, we find that using such broadcast information, 33 (42.86%) dongles can be uniquely fingerprinted for their vulnerability status. Among these 33 dongles, each of them contains at least one vulnerability, which indicates that they can be uniquely fingerprinted in the broadcast stage and exploited with the vulnerabilities discovered earlier

Connection Name	Type	# Dongle	Vulnerability				
			V1.1	V1.2	V2	V3	V4
V-Link	Wi-Fi	4	✓	✓	✓	✓	
FastLink M2	BLE	4	✓	✓		✓	
OBDBLE	BLE	3	✓	✓		✓	
V-checker	BLE	2	✓	✓		✓	
OBDII SCANNER	Wi-Fi	1	✓	✓	✓	✓	
OBDLink MX	Wi-Fi	1		✓		✓	
Carly Adapter	Wi-Fi	1	✓	✓		✓	✓
BlueDriver 2.39-B350	Bluetooth	1	✓				✓
OBDII	Bluetooth	1	✓	✓			
OBDLink MX+ 38611	Bluetooth	1		✓		✓	
7Q-Automatic Pro (LE)	BLE	1	✓				
Carista	BLE	1	✓	✓		✓	
DODYMPS OBD2	BLE	1	✓				
Dongle	BLE	1	✓				✓
FIXD	BLE	1	✓	✓		✓	
HGC	BLE	1	✓				
iOBD2 mini	BLE	1	✓	✓		✓	
IOS-Vlink	BLE	3	✓	✓		✓	
JUTA OBD II IOS	BLE	1	✓	✓		✓	
Kiwi 3A	BLE	1	✓	✓		✓	
TOPDON_760110	BLE	1	✓	✓		✓	
Viecar	BLE	1	✓	✓			

Table 8: Dongle Vulnerability Status by Connection Name.

(V1 to V4). The dongles with duplicated connection names possibly share the same development model, and thus it is likely that they also share the same vulnerability status.

6 Attack Case Studies

To demonstrate the severity of the vulnerabilities discovered in §5, we construct 4 classes of concrete attacks building up these vulnerabilities and validated them on our testing vehicle. These 4 attacks can lead to a wide range of security implications, including privacy, property theft, and even the safety of the drivers, passengers and pedestrians.

Prior to launching these attacks, an attacker first sniffs broadcast connection information from surrounding wireless network. Based on the sniffed information, she can identify a vulnerable OBD-II dongle by leveraging V5, or arbitrarily tries to connect to a nearby dongle for attacks. Next, lack of connection-level authentication vulnerability (V1.1) allows the attacker to establish a connection. Note that multiple device access vulnerability (V2) can enable such a malicious connection even when the dongle is connected with the vehicle owner's mobile device, which significantly increases the attack flexibility. Having established the connection, if lack of application-level authentication vulnerability (V1.2) exists, the attacker can conduct at least 4 attacks on the victim vehicle including vehicle-related data leakage (A1), property theft (A2), vehicle control interference (A3), and in-vehicle network infiltration (A4). Each attack needs one or more identified vulnerabilities as precondition, which is summarized in Table 9. We also present the number of dongles vulnerable to each attack, including the statistics with or without two optional preconditions (V2 or V5). When either V2 or V5 exists, the flexibility or success rate of the attack is much higher. Next, we describe each attack in detail.

A1. Vehicle-related Data Leakage. This attack only requires V1 to enable a nearby attacker to connect and read private data from the vulnerable OBD-II dongle with OBD-II PIDs [14]. We have demonstrated a few cases which can harvest location, vehicle diagnostic data, and CAN bus traffic from a victim vehicle through a vulnerable dongle, which endangers user privacy or assists other complicated attacks. As indicated in the table, 65 (84.42%) OBD-II dongles in our study are vulnerable to this attack.

- **A1.1. Location Leakage.** Using the diagnostic PID `09 02`, an attacker is able to get the vehicle identification number (VIN) which uniquely identifies an automobile. Since the VIN is also printed on the dashboard on the driver side and can be seen from outside, the attacker is capable of locating the target vehicle where a vulnerable dongle is installed, and performs further attacks.
- **A1.2. Diagnostic Data Leakage.** In addition to the VIN, an attacker is also able to read diagnostic data from the vehicle with the PIDs, including odometer, fuel rate, engine RPM, etc., which invades the privacy of the vehicle owner. Additionally, she can also analyze the driving behaviour and fingerprint drivers with the leaked data such as vehicle speed and throttle position [32, 33].
- **A1.3. CAN Bus Traffic Leakage.** Reverse engineering of the CAN bus protocol is non-trivial but of great value [10, 13, 23]. We discover that by injecting an `ATMA` command to an ELM327-based OBD-II dongle, one is able to dump the CAN bus traffic to analyze the CAN bus protocol. Therefore, an attacker is able to harvest safety related CAN bus control messages (e.g., applying brake) to perform arbitrary CAN bus message injection attack when V3 exists.

A2. Property Theft. To achieve this attack, V1 and V3 are required. Therefore, 46 (59.74%) dongles are vulnerable to property theft. During the experiment, we found one CAN bus message that is able to disable the wireless locking capability of our testing vehicle. Using this message, we construct a property theft attack. First, an attacker locates a target vehicle as mentioned in A1.1. Next, she injects the message and disables the wireless locking capability, and waits for the owner on the vehicle to leave. When the driver exits the vehicle and locks the vehicle remotely with his key as usual, he or she may not know the locking is unsuccessful and thus leaves without any concern. Afterwards, the attacker has the opportunity to sneak into the vehicle and steal all the belongings.

A3. Vehicle Control Interference. Fuzzing is a technique widely used in software testing, which can help find bugs by sending random inputs to a computer program [34]. Similarly, attackers can fuzz diagnostic CAN bus messages to a vehicle through a vulnerable dongle with V1 for denial-of-service (DoS) purpose. Moreover, fuzzing with control related CAN bus messages to a dongle with V3 can even cause interference on the vehicle control, which threatens the safety of drivers,

Attack Case		Precondition						# Vulnerable Dongle (%)		
		V1.1	V1.2	V2	V3	V4	V5	w/o V2,V5	w/ V2	w/ V5
A1.1	Location Leakage	✓	✓	○			○	65 (84.42%)	27 (35.06%)	26 (33.77%)
A1.2	Diagnostic Data Leakage	✓	✓	○			○	65 (84.42%)	27 (35.06%)	26 (33.77%)
A1.3	CAN Bus Traffic Leakage	✓	✓	○			○	65 (84.42%)	27 (35.06%)	26 (33.77%)
A2	Property Theft	✓	✓	○	✓		○	46 (59.74%)	20 (25.97%)	24 (31.17%)
A3	Vehicle Control Interference	✓	✓	○	✓		○	46 (59.74%)	20 (25.97%)	24 (31.17%)
A4	In-vehicle Network Infiltration	✓	✓	○		✓	○	2 (2.60%)	0	2 (2.60%)

Table 9: Proposed Attack Cases and Vulnerable Dongle Statistics. ✓indicates mandatory precondition, ○ indicates optional precondition that are not necessary but can increase the attack flexibility (e.g., with V2) or attack success rate (e.g., with V5).

passengers, and pedestrians. Previous research has confirmed the serious consequences caused by the fuzzing attack, which can affect the engine, instrumentation panel, and brake system [10, 35]. Among the collected dongles, 46 (59.74%) are vulnerable to this attack. Note that to trigger the actual effects on the vehicle, one must fuzz with high frequency to overwrite the normal messages. To validate this attack, we wrote a Python script that injects random CAN bus messages every 10 milliseconds to a vulnerable dongle with V3, which resulted in abnormal behaviour on our testing vehicle since its alert system went on. We had to stop it before the fuzzing caused permanent damage to the vehicle.

A4. In-vehicle Network Infiltration. V1 and V4 allow an unauthorized attacker to send a malicious firmware packet to subvert the dongle's firmware. Since OBD-II dongles are directly connected with CAN bus, the attacker is able to infiltrate the in-vehicle network by replacing the firmware to achieve malicious purposes such as spoofing and eavesdropping attacks. Among all the OBD-II dongles in our study, 2 (2.60%) (including 2 false negatives due to private messages) dongles are vulnerable to this attack.

7 Discussion and Future Works

7.1 Tool Effectiveness

First, we discuss the effectiveness of DONGLESCOPE in terms of its false positives and false negatives in correctly achieving the measurement objectives summarized in Table 1.

False positives. In the design and implementation of DONGLESCOPE, the measurement objectives across all the three attack stages are tested dynamically with the dongle under test plugged into a real automobile. As a result, the analysis results do not have false positives. The vulnerabilities identified are true vulnerabilities and confirmed with dynamic analysis.

False negatives. False negatives may exist in both our dynamic dongle analysis and static mobile app analysis. In the broadcast and connection stage, DONGLESCOPE follows the default configuration to collect broadcast information and sets up connection, which does not result in false negatives, otherwise the dongle is also not usable for normal users. The analysis in communication stage can bring false negatives in our results. For example, one source of false negative is our design assumption that all the dongles are ELM327-based (described in §4.2). However, among the dongles we collected (detailed in §5.1), we find that a small portion of dongles has their own implementations of the communication protocol between the dongle and the app. For these dongles, DONGLESCOPE failed to get responses by testing with standard ELM327 commands. We discover these cases using app analysis results and manual confirmation, which results in 5 false negatives during the connection stage analysis in §5.3. Since the implementations for each of these dongles may be different, it is non-trivial to design a generic approach to cover these cases, which is thus left as future work. As for our static app analysis results, false negatives may exist due to code obfuscation which confuses the control flow of the program. As a result, the set of messages we can identify through static analysis is a subset of all messages supported by the dongle.

To summarize, DONGLESCOPE does not have false positives but may have false negatives. Thus, the analysis results in the paper present a *lower bound* of the vulnerability status of the dongles in our experiments.

7.2 Root Causes and Countermeasures

In this paper, we have uncovered 5 general vulnerabilities on wireless OBD-II dongles that lead to remote or nearby attacks. To summarize, there are two root causes. On one hand, OBD-II dongles have direct access to the CAN bus through the OBD-II port. On the other hand, unauthorized access allows a nearby attacker to write malicious messages to OBD-II dongles. To eliminate these vulnerabilities, countermeasures can be deployed on any of the three entities: the CAN bus, the OBD-II port, or OBD-II dongles, which are detailed as follows.

Authentication on the CAN bus. Deploying secure authentication on the CAN bus is a fundamental solution, since it eliminates all unauthorized messages regardless of the security of the OBD-II port and OBD-II dongles. This has been well studied in the literature (e.g., [36–40]). However, due to the insecure nature of the CAN bus protocol design as well as the high demand on performance (e.g., low latency), there is no effective and practically deployable solution so far [41].

Firewall on the OBD-II port. Another countermeasure is to build a firewall on the OBD-II port to prevent malicious message injection. For example, a physical gateway module is developed for Chrysler models, which has deployed cloud authentication for access control [42]. Specifically, unauthorized devices only have limited harmless capabilities such as read operations to the diagnostic CAN bus. The drawback of this countermeasure is that it requires vehicle owners to purchase an additional gateway to protect the OBD-II port. Moreover, the existing gateway device is only compatible with a few car models, which is apparently not a universal solution. Therefore, the design and development of such a generic gateway is left to another future work.

Authentication on OBD-II dongles. As demonstrated in our paper, lack of authentication on connection layer and application layer of OBD-II dongles are the necessary preconditions for any nearby attacks to a vehicle. Therefore, deploying secure authentication is also an effective way to prevent the attacks. However, it is a non-trivial task for two reasons. First, since OBD-II dongles usually neither have cellular network nor user interface such as a display or keyboard, they require an external device (e.g., a mobile app) to first authenticate itself by sending specific credentials before communication, which can introduce a new attack surface. For example, the hardcoded credentials are discovered in the Bosch dongle app. Therefore, the developers also need to spend a significant amount of effort to secure the authentication process in apps (e.g., by using sophisticated algorithms or involving cloud). Second, deploying secure authentication is costly. On one hand, infrastructure such as cloud needs to be involved. On the other hand, additional effort may be needed to customize the dongle firmware, such as hardcoding a random PIN in each dongle. Possibly due to these reasons, a majority of dongles available on the market today is still vulnerable to attacks. In our future work, we plan to design and develop a secure authentication protocol between OBD-II dongles and mobile apps atop some well-known platforms such as OpenXC [43].

7.3 Responsible Disclosure

On August 7, 2019, we reported the discovered vulnerabilities via email to 29 vendors, which covers 47 (61%) OBD-II dongles in our study. For the remaining 30 (39%) dongles, we were not able to find the contact information of their vendors. As of November 19, 2019, which is over 3 months after our disclosure, we have received responses only from a handful vendors in total. Among them, 2 vendors have decided to deploy authentication in the future versions of their dongles, while the other are still discussing our reported problems.

We believe the reasons for such a low response rate may be two-fold. First, most of these OBD-II dongles are quite cheap, e.g., 75% of them are actually less than $30. Such a low price can increase their product competitiveness on the market, but this also means that the vendors may not be able to afford adding extra security features. As discussed in §7.2, deploying authentication in these dongles requires significant efforts, which thus will inevitably increase the dongle cost. Second, we find that there is a lack of security awareness among some vendors. In particular, among the responses we received, some vendors did not consider the leakage of some CAN bus data, e.g., speed and VIN, as privacy leakage. However, as discussed in §6 and shown by previous work [32, 33], these data can indeed lead to privacy breaches such as leaking the driver's identity.

Based on our experience above, it is actually both ineffective and inefficient to address the OBD-II dongle vulnerabilities by contacting the vendors directly, since it is not only difficult to find their contact information, but also hard to convince them to take security enhancement actions. Thus, we have already reported all the vulnerabilities discovered in this paper to CVE in order to ensure enough public disclosure of this class of security problems. As of November 27, 2019, our findings have been acknowledged by a set of CVEs [2]. At the same time, we also hope our work can raise immediate attention in the community, and make joint efforts to secure the products in automotive IoT.

8 Related Work

Attack and defense on the CAN bus. As the core of a modern automobile, CAN bus has long been an important target for security research. It is often regarded as a fragile protocol vulnerable to a significant number of attacks because of its design-level flaws. As a broadcast protocol, CAN bus does not support authentication, and thus it is not capable of distinguishing spoofed CAN bus messages from normal messages [36], which leads to unauthorized external access. According to previous research, one kind of common attack is to inject CAN bus messages through the OBD-II port which directly connects to the internal CAN bus, which ultimately affects the behaviour of the vehicle. Prior efforts have demonstrated the attack on real automobiles (e.g., a Chevy Impala, a Ford Escape, or a Toyota Prius) as well as eavesdropping the CAN bus protocol through the OBD-II port [10, 13, 19, 44, 45]. The injection attack can also lead to serious consequences such as engine shut down and sudden braking, which brings severe safety concerns.

In addition to the OBD-II port, a large body of efforts has focused on analyzing other attack surfaces of modern vehicles including IVI system, radio system, sensors, LiDAR, connected vehicle device, etc [44, 46–53]. Most recently, with the massive growing popularity of the IVI system, it becomes

[2]The complete list of CVEs is available at https://github.com/OSUSecLab/DongleScope/blob/master/ResponsibleDisclosure.md where we will also update the progress of our responsible disclosure.

a great target since it has direct access to the CAN bus. For instance, Mazloom et al. discover the security holes of the MirrorLink protocol in an IVI system allowing injecting malicious messages to the CAN bus [54]. Mandal et al. conduct static program analysis on the IVI Android apps and reveal tens of vulnerabilities such as poor access control [55]. Miller et al. uncover a software vulnerability on the Uconnect system on a Jeep Cherokee that allows remote attack via cellular network [44].

While prior efforts have revealed novel vulnerabilities and demonstrated attacks on automobiles, most of them are adhoc and the vulnerability analysis is not comprehensive. In addition, most of the attacks require direct physical access to the vehicle [13, 19], which narrows the flexibility of attacks. Compared to these works, we are the first to conduct a comprehensive vulnerability study on a large number of OBD-II dongles on the market, and our uncovered vulnerabilities can lead to wireless attacks on nearby vehicles.

In order to counter attacks on the CAN bus, novel defenses have also been proposed overtime. It is non-trivial to make harmful impacts to the vehicle by directly injecting CAN bus messages since there are complicated defensive approaches deployed by the car manufacturers. One common defense countering the CAN bus message injection attack is conflict detection. As the ECUs are always broadcasting CAN bus messages at a fixed frequency, it is possible to detect anomalous messages from malicious senders [19]. Consequently, many attacks become impractical in reality, such as directly injecting CAN bus message to operate the vehicle. In addition, there are also other defensive approaches such as intrusion detection [56–58] and data authentication [36–40].

Mobile app based vulnerability discovery. With the massive growth of the mobile app market today, mobile app analysis has been widely used to uncover various vulnerabilities. One particular analysis is the data flow analysis that has been used to track the data flow, trace the data of interest and identify vulnerabilities in the phone (e.g., [59–63], remote servers (e.g., vulnerable authentication [64], authorization [65], and cloud leakage [66]). Inspired by these efforts, DONGLESCOPE adopts static analysis to extract predefined messages from the companion apps to study the security of OBD-II dongles.

9 Conclusion

In this paper, we perform the first comprehensive security analysis on 77 wireless OBD-II dongles available on Amazon in the US in February 2019. To systematically perform the analysis, we design and implement an automated analysis tool DONGLESCOPE, and use it to identify 5 different types of vulnerabilities, with 4 being newly discovered. Our results show that each of these 77 dongles exposes at least two types of these vulnerabilities, which indicates a widespread vulnerability exposure among wireless OBD-II dongles on the market today. To demonstrate the severity, we construct 4 classes of concrete attacks that can cause privacy leakage, property theft, and safety threat. We also discuss the root causes and feasible countermeasures, and have performed responsible disclosure.

Finally, as IoT devices are increasingly used in safety-critical domains such as vehicles, we expect that our domain-specific findings and their security/safety implications can send a strong and timely message to start developing and deploying principled security designs in these highly critical application domains such as automotive IoT.

Acknowledgment

We thank our shepherd Nils Ole Tippenhauer as well as the anonymous reviewers for their insightful comments. This research was supported in part by National Science Foundation (NSF) Awards 1834215 and 1850533. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily of the NSF.

References

[1] "Obd-ii - on-board diagnostic system," http://www.obdii.com/.

[2] D. S. Eisinger and P. Wathern, "Policy evolution and clean air: The case of us motor vehicle inspection and maintenance," *Transportation Research Part D: Transport and Environment*, vol. 13, no. 6, pp. 359–368, 2008.

[3] A. Kovelman, "A Remote Attack on the Bosch Drivelog Connector Dongle," https://argus-sec.com/remote-attack-bosch-drivelog-connector-dongle, 2017.

[4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[5] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home internet of things (iot)," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 255–272.

[6] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 636–654.

[7] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms," in *NDSS*, 2017.

[8] J. Erickson, Q. A. Chen, X. Yu, E. Lin, R. Levy, and Z. M. Mao, "No One In The Middle: Enabling Network Access Control Via Transparent Attribution," in *ACM AsiaCCS*, 2018.

[9] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, "Understanding Fileless Attacks on Linux-based IoT Devices with HoneyCloud," in *ACM MobiSys*, 2019.

[10] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462.

[11] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media, 2012.

[12] M. Ruta, F. Scioscia, F. Gramegna, and E. Di Sciascio, "A mobile knowledge-based system for on-board diagnostics and car driving assistance," in *UBICOMM 2010, The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*. Citeseer, 2010, pp. 91–96.

[13] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, pp. 260–264, 2013.

[14] "OBD-II PIDs," http://obdcon.sourceforge.net/2010/06/obd-ii-pids/.

[15] "Top-5 best obd2 scanners worth buying in 2019 | buyer's guide," https://gadgets-reviews.com/review/735-top-best-obd2-scanner.html.

[16] "Parani-ud100 bluetooth 4.0 class1 usb adapter," http://www.senanetworks.com/ud100-g03.html, 2019.

[17] C. Zuo, H. Wen, Z. Lin, and Y. Zhang, "Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019, pp. 1469–1483.

[18] R. Malekian, N. R. Moloisane, L. Nair, B. T. Maharaj, and U. A. Chude-Okonkwo, "Design and implementation of a wireless obd ii fleet management system," *IEEE Sensors Journal*, vol. 17, no. 4, pp. 1154–1164, 2016.

[19] C. Miller and C. Valasek, "Can message injection," *OG Dynamite Edition*, 2016.

[20] "Elm327dsk," https://www.elmelectronics.com/wp-content/uploads/2017/01/ELM327DS.pdf.

[21] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing." in *NDSS*, 2018.

[22] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, "Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 122–133.

[23] K. Kuchera, "How to Hack a Car - A Quick Crash Course," https://www.freecodecamp.org/news/hacking-cars-a-guide-tutorial-on-how-to-hack-a-car-5eafcfbbb7ec, 2017.

[24] "How to Hack a Car - A Quick Crash Course," https://medium.freecodecamp.org/hacking-cars-a-guide-tutorial-on-how-to-hack-a-car-5eafcfbbb7ec.

[25] Bisdikian and Chatschik, "An overview of the bluetooth wireless technology," *IEEE Commun Mag*, vol. 39, no. 12, pp. 86–94, 2001.

[26] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[27] "socket - low-level networking interface," https://docs.python.org/3/library/socket.html.

[28] "Pybluez," https://github.com/pybluez/pybluez.

[29] "Pyble," https://github.com/jesstess/PyBLE.

[30] M. Weiser, "Program slicing," in *Proceedings of the 5th international conference on Software engineering*. IEEE Press, 1981, pp. 439–449.

[31] "Sable/soot: Soot - a java optimization framework," https://github.com/Sable/soot.

[32] S.-H. Chen, J.-S. Pan, and K. Lu, "Driving behavior analysis based on vehicle obd information and adaboost algorithms," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2015, pp. 18–20.

[33] M. Enev, A. Takakuwa, K. Koscher, and T. Kohno, "Automobile driver fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 1, pp. 34–50, 2016.

[34] M. Sutton, A. Greene, and P. Amini, *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[35] H. Lee, K. Choi, K. Chung, J. Kim, and K. Yim, "Fuzzing can packets into automobiles," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. IEEE, 2015, pp. 817–821.

[36] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011.

[37] D. K. Nilsson, U. E. Larson, and E. Jonsson, "Efficient in-vehicle delayed data authentication based on compound message authentication codes," in *2008 IEEE 68th Vehicular Technology Conference*. IEEE, 2008, pp. 1–5.

[38] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, "Libra-can: a lightweight broadcast authentication protocol for controller area networks," in *International Conference on Cryptology and Network Security*. Springer, 2012, pp. 185–200.

[39] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata, "Cacan-centralized authentication system in can (controller area network)," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.

[40] A.-I. Radu and F. D. Garcia, "Leia: A lightweight authentication protocol for can," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 283–300.

[41] M. Bozdal, M. Samie, and I. Jennions, "A survey on can bus protocol: Attacks, challenges, and potential solutions," in *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE, 2018, pp. 201–205.

[42] "Fca secure gateway module," https://diag.net/msg/m1fsoznwl3nndqti9pxq9k4nz0.

[43] "openxc-android," https://github.com/openxc/openxc-android, 2019.

[44] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[45] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: A story of telematic failures," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[46] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *black hat USA*, vol. 2014, p. 94, 2014.

[47] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, vol. 4. San Francisco, 2011, pp. 447–462.

[48] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 546–556, 2014.

[49] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial Sensor Attack on LiDAR-based Perception in Autonomous Driving," in *ACM CCS*, 2019.

[50] Q. A. Chen, Y. Yin, Y. Feng, Z. M. Mao, and H. X. Liu, "Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control," in *NDSS*, 2018.

[51] Y. Feng, S. Huang, Q. A. Chen, H. X. Liu, and Z. M. Mao, "Vulnerability of Traffic Control System Under Cyber-Attacks Using Falsified Data," in *Transportation Research Board (TRB)*, 2018.

[52] Y. Tu, Z. Lin, I. Lee, and X. Hei, "Injected and delivered: fabricating implicit control over actuation systems by spoofing inertial sensors," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1545–1562.

[53] Y. Feng, S. Huang, Q. A. Chen, H. X. Liu, and Z. M. Mao, "Vulnerability of Traffic Control System Under Cyberattacks with Falsified Data," *Transportation Research Record (TRR)*, vol. 2672, no. 1, pp. 1–11, 2018.

[54] S. Mazloom, M. Rezaeirad, A. Hunter, and D. McCoy, "A security analysis of an in-vehicle infotainment and app platform," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[55] A. K. Mandal, A. Cortesi, P. Ferrara, F. Panarotto, and F. Spoto, "Vulnerability analysis of android auto infotainment apps," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*. ACM, 2018, pp. 183–190.

[56] K.-T. Cho and K. G. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *USENIX Security Symposium*, 2016.

[57] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *2010 Sixth International Conference on Information Assurance and Security*. IEEE, 2010, pp. 92–98.

[58] W. Wong, S. Huang, Y. Feng, Q. A. Chen, H. X. Liu, and Z. M. Mao, "Trajectory-Based Hierarchical Defense Model to Detect Cyber-Attacks on Transportation Infrastructure," in *Transportation Research Board (TRB)*, 2019.

[59] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transaction on Computer Systems (TOCS)*, 2014.

[60] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Acm Sigplan Notices*, vol. 49, no. 6. ACM, 2014, pp. 259–269.

[61] Q. A. Chen, Z. Qian, Y. J. Jia, Y. Shao, and Z. M. Mao, "Static Detection of Packet Injection Vulnerabilities: A Case for Identifying Attacker-Controlled Implicit Information Leaks," in *ACM CCS*, 2015.

[62] Y. Shao, Q. A. Chen, Z. M. Mao, J. Ott, and Z. Qian, "Kratos: Discovering Inconsistent Security Policy Enforcement in the Android Framework," in *NDSS*, 2016.

[63] Y. J. Jia, Q. A. Chen, Y. Lin, C. Kong, and Z. M. Mao, "Open Doors for Bob and Mallory: Open Port Usage in Android Apps and Security Implications," in *IEEE Euro S&P*, 2017.

[64] C. Zuo, W. Wang, R. Wang, and Z. Lin, "Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS'16)*, San Diego, CA, February 2016.

[65] C. Zuo, Q. Zhao, and Z. Lin, "Authscope: Towards automatic discovery of vulnerable authorizations in online services," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS'17)*, Dallas, TX, November 2017.

[66] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *Proc. IEEE Symposium on Security and Privacy*, 2019.