IoT Penetration Testing: Security analysis of a car dongle

Aldin Burdzovic and Jonathan Matsson

Abstract— **The ambition for Internet of Things (IoT) devices of becoming a part of our everyday lives, is not only done by entering our homes but also our vehicles. The demand of attachable smart IoT products for cars is high. One such product is the AutoPi, which connects the car to the internet and allows for various features, usually found in high-end luxury cars.**

This paper presents an analysis of the cyber security aspects of AutoPi. The findings presented shows that there is a critical vulnerability in the system. The AutoPi can be exploited and full access of the devices can be granted. The paper also discusses what possible harm can be done through the found exploit.

Sammanfattning— **Ambitionen för Internet of Things (IoT) apparater att bli en del av det vardagligalivet sker inte endast i våra hem, utan även i våra fordon. Efterfrågan på smarta IoT produkter för bilar är hög. En sådan produkt är AutoPi, vilket ansluter bilen till Internet och möjliggör för diverse funktioner vanligtvis funna i avancerade lyxbilar.**

Denna uppsats presenterar en analys av cybersäkerheten för AutoPi. Upptäckterna som presenteras visar på att det finns en kritisk säkerhetsbrist i systemet och full åtkomst till apparaten kan uppnås. Uppsatsen diskutera även möjliga skador som kan göras genom den funna sårbarheten.

I. INTRODUCTION

The Internet of Things (IoT) is one of the hottest tech terms today and is an increasingly debated topic as there seems to be a boundless potential for improving everyday lives. The idea of IoT is to attach embedded devices to everyday objects to make them ”smart”. IoT is already taking over the automotive industry were newer vehicles often come standard equipped with internet connection and various IoT technology such as autonomous driving [1]. Since the automotive market to a large extent consist of second-hand vehicles, the demand of attachable smart IoT products is high. Many companies are now attempting to develop such products [2].

The company AutoPi[1] have developed a smart IoT dongle for the car that enables various features to help and assist the end-user. The AutoPi dongle supplies the user with valuable information and diagnostics about the vehicle while allowing various smart features, usually found in high-end luxury vehicles. However, the amount of connected devices that comes with the implementation of IoT technology and especially having them so present in our daily lives, the important topic of security arises. Manufacturers can often overlook security in attempt of getting their product out on the market as quick as possible. So how great is the security risk of these devices and what harm can be done? This paper presents an analysis of the cyber security aspects of AutoPi.

We use threat modeling to plan and prioritize the work, as well as ethical hacking (penetration testing) to analyze the dongle security. As a result we found a vulnerability in AutoPi Wi-Fi/NB and 4G/LTE devices, that up until the writing of this report on 2019-10-15 wasn't fixed. The company has been informed and we have followed common responsible disclosure protocols (90 days plus an additional 30 days after discussions with the company). The vulnerability has been assigned CVE-2019-12941.

II. BACKGROUND

This section introduces the reader to the topic and background information necessary for understanding the report.

A. ODB-II

On-Board-Diagnostics-II (ODB-II) is a standard which regulates the look of the plug for the built-in car diagnostics port. The ODB-II port allows for access to the vehicles various sensors through communication with the car's Electronic Control Unit (ECU). The port is a way for external hardware to communicate with the vehicle internal system, often used by workshops for diagnostics and identifying errors. In 1994, the ODB-II was standardized for all cars in the United States, with Europe following in 2001 for all gasoline fueled cars and in 2004 for all diesel cars [3]. Since then, ODB-II has evolved into a much higher level of functionality allowing more advanced diagnostics with a much greater detail. Today, there is a growing market for devices that utilizes ODB-II in order to provide various functionality to the end user[2].

B. CAN

The Controller Area Network (CAN) is the standardized internal network protocol in the automotive industry. CAN is an asynchronous, multi-layer serial bus communication protocol accessible via the cars OBD-II port. It is the first widely accepted automotive bus protocol and has been the standard for internal network in passenger cars for over 30 years. CAN is a broadcast type of bus, meaning that all messages that are sent on the network are available system-wide. The nodes in the CAN network are in fact ECUs, each controlling a certain set of functions within the vehicle. It relies on several rules for which node gets to transmit over the network and which listens. The CAN frame includes a destination field and data is multicasted on the bus where nodes only address data which is addressed to them [5]. However, CAN was not designed to be secure from intrusion [4], but rather to enable fast and stable communication. It

[1]https://www.autopi.io

[2]https://www.marketwatch.com/press-release/global-obd-aftermarket-industry-to-surpass-15bn-by-2024-global-market-insights-inc-2018-08-28

relies on that only the desired receivers are connected to the network since there is no information about the source in the frames, meaning that receiving nodes cannot now from where the messages was sent and ultimately determine if it is trustworthy or not.

C. Raspberry Pi

In 2012, the first version of the Raspberry Pi was released and has since become an attractive product with its small size, relative good performance, low power consumption and affordable price. The Raspberry Pi is a simple single-board computer, which unlike a microcontroller, runs an operating system and also has a much faster CPU. The result is a credit-card sized computer capable of performing most of the tasks of a regular computer. The platform also features WiFi, Bluetooth, Ethernet, HDMI and USB ports. It runs on an operating system named Raspbian which is a Debian-based Linux distribution [10].

D. AutoPi

AutoPi provides a service to make your car a ”smart car”. A dongle is inserted into the OBD-II port of the car which gives the dongle access to the cars internal systems. AutoPi also provides a cloud service that lets you communicate with the dongle remotely over the Internet.

The dongle is built on a Raspberry PI Zero which makes it a very powerful IoT-device. Hardware of the dongle that is of interest in this paper are WiFi, Bluetooth, 4G, A-GPS, two USB ports and a mini-HDMI port. The dongle runs a Web server and a Secure Shell (SSH) server which are reachable from the internal WiFi network.

The dongle also runs software developed by the AutoPi team to simplify communication with the car and dongle. For instance, the provided API lets the user run simple HTTPS requests to record and replay commands on the CAN bus. The software is open source under the Apache License and can be found on github[3].

The AutoPi is sold in several editions offering different services. This paper will address the ”4G/LTE Edition GEN2”-edition which is the fully equipped high end model. Some results presented in this paper might be applicable to other models as well.

E. Threat Modelling

Threat modeling is used to get a better understanding of possible security threats to a system [12, p. 32]. The process usually starts by producing a very general idea about possible threats and stepwise produce more tangible and detailed threats. A good threat model will not only help finding threats, but also help prioritize threats according to their severity and discoverability.

[3]https://github.com/autopi-io/autopi-core

F. Ethics

The paper is focused on testing security of an IoT device intended for cars. This is done by hacking and finding vulnerabilities in the device. This raises an ethical dilemma. Is it morally okay to find and publish vulnerabilities of devices which can be used for something harmful, even if the motive behind it is good?

To make tech products unhackable, they basically have to be very simple with less functionality. However, tech products are getting more and more complex with advanced systems and greater functionality. This leaves much more room for security flaws in those products. These security flaws can be exploited by hackers. Normally, when people hear the word hacker, they think of criminals. But there are ”ethical hackers”, who for a living, exposes the vulnerabilities of these products. The reasoning behind ethical hacking is that it is better for someone ”good” to find the vulnerabilities before someone ”bad” finds them. Hence, it is better for someone trusted to find and report the vulnerabilities before criminals exploit them.

When finding a vulnerability, it is important to disclose it in a responsible way. This is done by notifying the developers of the vulnerability and giving them time to patch it before disclosing the vulnerability to the public. For the vulnerabilities found in this paper, a 90 day disclosure deadline was given to the developers. This method of responsible disclosure is taken from the Google Project Zero[4] to match industry standards. A deadline also pushes the developers to patch the system and improve their security in a timely manner.

III. Threat Modeling

The thread model is the foundation of which the security testing is based upon. The threat modeling for the AutoPi system documented in this paper follows the steps described in the book ”IoT Penetration testing cookbook” [12, p. 42].

A. System Model

The premise of the AutoPi service is to let its end users have full control over their dongles and modify them to fit their needs. This opens up for possible security holes as the end users might not be particularly experienced with security. Since the possibility of modification is practically endless, it is impossible to consider all possible security risks in this paper. Therefore, the paper is focused on security of dongles using the pre-installed hardware and software with only slight modifications of the default settings.

Figure 1 is a simple overview of the system components that pose a security risk. Every item in the figure is explained in more details in the list underneath. Components that we do not see as a possible security threat have been excluded from our system model.

1) **AutoPi:** This is the main device. The dongle is built on a Raspberry PI Zero with Raspbian as the pre-installed operating system. This opens up for potential

[4]https://googleprojectzero.blogspot.com/2015/02/feedback-and-data-driven-updates-to.html

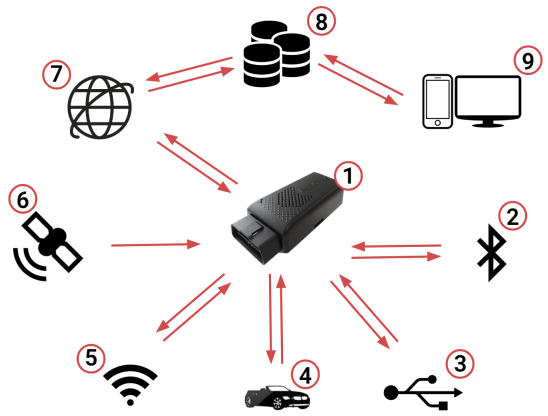

Fig. 1. Simplified threat model

attack surfaces since the Raspberry PI contains more complexity compared to a simple embedded system.
While the car is turned off, the dongle will sleep for cycles of 2 hours and wake up for 5 minutes between sleep cycles. This is to prevent drainage of the car battery.

2) **Bluetooth:** The AutoPi comes with Bluetooth 4.1 and Bluetooth Low Energy (BLE). There are no default software on the device which uses Bluetooth. It is mainly for connecting third-party products through, combined with self-written code on the device, to accomplish some wanted feature.
3) **Physical connections:** The devices comes with physical ports that can be used to for implementing additional functionality to the dongle. The dongle has two USB 2.0 ports, one mini-HDMI port and 18 GPIO pins.
4) **OBD:** The device is connected to the OBD-II port of the car. The OBD-II port provides the dongle with power and is also used for communication between dongle and car. Some examples of functions that this port can be used for are remotely starting the car[5] or unlocking the car[6].
5) **WiFi:** The device can act both as a WiFi hotspot and a WiFi client. When connected to 4G, the device can be a WiFi hotspot so that other may connect to the network to gain internet connection. Also, the device itself can connect to a WiFi network to establish an internet connection without depending on 4G connection. When connected to the AutoPi dongles WiFi network, one have access to the local web portal of the devices. The web portal allows for various network configuration, also including a terminal to run commands on the device.
When connected to the WiFi, it is also possible to SSH into the device. Both the web portal terminal and the SSH terminal grants root access, meaning that full access of the devices is given when connected through WiFi.
6) **GPS:** The device comes with a GPS module for real-time tracking of position, speed and altitude. It includes Assisted-GPS (A-GPS) to improve startup performance.
7) **4G:** Internet connection is provided through a built in 4G-module. A sim-card is required. This is a highly secure network since 4G encrypts the traffic between the device and the base station [11].
8) **Cloud Servers:** There are two cloud servers providing different services. One server communicates with end users and one communicates with dongles. The communication with end users will be sent over HTTPS and the communication with dongles will be sent with the SaltStack protocol.
9) **Web Portal:** The web portal, also known as the AutoPi Cloud software platform, allows for a dashboard environment where the user remotely can monitor and perform certain actions regarding the AutoPi. For instance, the web portal displays data both from the car's internal computer and from external devices connected to the AutoPi. The web portal also includes a terminal for sending commands to be run on the AutoPi. The terminal provided grants the user with root access.

[5]https://www.autopi.io/use-cases/remote-start/
[6]https://www.autopi.io/use-cases/auto-lock-unlock/

B. Identifying Threats

The simplified threat model in Figure 1 gives an overview of attack vectors. The threat model is then used to identify explicit threats to help with documentation of threats. The STRIDE method is used to get a general understanding of possible threats. The most severe and discoverable threats found via the STRIDE method are documented in greater detail and ranked according to the DREAD method.

C. STRIDE

The STRIDE method is used to identify and categories threats [12, p. 49] and is a commonly used in threat modeling of vehicles [6]. STRIDE is an acronym for *Spoofing of user identity*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service* and *Elevation of Privilege*.

These categories are used to help and ensure that all type of threats are considered. The threats found with the method can be seen in the list underneath:

Spoofing of user identity

- Claiming to be another user to get control over other dongles.
- Pretending to be the cloud server and intercept traffic from users and dongles destined to the real cloud server.
- Impersonating another dongle to retrieve unauthorized information from the cloud server.

Tampering

- Modifying data sent between client, dongle and server.

Repudiation

Information Disclosure

- Intercept data sent between client, dongle and server.
- Capture data sent on the vehicles CAN bus.
- Set up a monitoring access point.

Denial of Service

- Bring down the dongles WiFi to prevent communication between client and dongle.
- Bring down the dongles 4G connection to prevent communication between dongle and server.

Elevation of Privilege

- Bypass WiFi authorization and connect to the device with root access.
- Brute force web portal password to access dongle web platform which gives root access.

The four threats that we saw as most severe and discoverable was documented in greater detail. These threats can be seen in Tables I through IV.

TABLE I
THREAT 1

Threat description	Intercepting and modifying traffic sent between dongle and server
Threat target	Network interface between dongle and server
Attack techniques	Man-in-the-middle between dongle and server
Countermeasures	Authorize the dongle and server to each other

TABLE II
THREAT 2

Threat description	Attacker bypasses the WiFi authorization and connects to the dongles WiFi network
Threat target	AutoPi dongle
Attack techniques	The attacker brute forces a large variety of common/random passwords to authenticate to the network
Countermeasures	Use complex password

TABLE III
THREAT 3

Threat description	Claiming to be another user to get control over dongles that the perpetrator should not have access to
Threat target	Web server and access tokens
Attack techniques	Phishing or bruteforce to obtain login credentials. Modification of access tokens.
Countermeasures	Preventing large numbers of login attempts in a short amount of time and reduces login verification speed. Thorough checks on access tokens

D. DREAD

The threats were ranked according to the DREAD method[12, p. 33]. DREAD is an acronym for *Damage potential*, *Reproducibility*, *Exploitability*, *Affected users* and *Discoverability*.

TABLE IV
THREAT 4

Threat description	Vulnerable services running on the dongle
Threat target	AutoPi dongle
Attack techniques	Scanning ports
Countermeasures	Keep services up-to-date and implement good firewall rules

Every threat were given a score between 1 through 3 for every category (1 being the lowest value and 3 the highest). The score of all categories were summed to give a total score. The threats can then be prioritized according to their total score. The DREAD ranking can be seen in Table V.

TABLE V
DREAD RANKING

	Threat 1	**Threat 2**	**Threat 3**	**Threat 4**
D	2	3	3	2
R	2	1	1	2
E	1	2	3	3
A	3	1	1	3
D	1	2	2	2
Total	9	9	10	12

IV. THEORY

With consideration to the threat model from previous chapter, it is evident that the greatest attack vectors are communication involving the cloud server and the WiFi network since three out of the four threats are applicable to those component. The WiFi is remotely accessible from outside of the car and a host connected to the WiFi network will have root access to the device. The same applies to the cloud server. This paper is therefore primarily focused on threats regarding those two components.

A. Dongle services

Services running on the dongle that are of interest to this paper is services that are remotely reachable. This includes services that are listening on a specific port that is reachable through the firewall of the dongle or services that in some way communicate with hosts outside the dongle. The Iptable rules of the dongle[7] specifies the open ports on which the dongle listens (these services are only reachable from the dongles local WiFi network). They can be seen in Table VI.

Because of the strict firewall rules, the only services reachable from hosts outside the dongles own local WiFi are services that initiates the connection towards outside hosts.

Some of the applications running on the dongle might have known vulnerabilities that can be used to exploit the dongle.

[7]https://github.com/autopi-io/autopi-core/blob/master/src/salt/base/state/network/wlan/hotspot/iptables-ipv4.rules

TABLE VI
OPEN PORTS AND CORRESPONDING SERVICES

Service	Port
SSH	22 (TCP)
DNS	53 (TCP & UDP)
DHCP	67 (UDP)
HTTP	80 (TCP)
HTTP (API)	9000 (TCP)

A common way of finding vulnerabilities for applications are with the use of the Common Vulnerabilities and Exposures (CVE) list[8] which contains publicly known vulnerabilities.

The services that are reachable remotely within WiFi range are: the WiFi hotspot (hostapd version 2.4), the WiFi client (wpa_supplicant version 2.4) and the WiFi DHCP client (dhcpcd version 6.11.5). Services that communicate over the Internet are SaltStack (version 2017.7.5) and HTTPS request are sent via the python library requests (version 2.12.4). There are no severe vulnerabilities reported of these services applicable to the dongle.

B. Wifi hotspot

The WiFi hotspot is configured to use WPA2 encryption with a 12 hexadecimal number as password. The password is obtained from the first 12 characters of the dongle id and the SSID is the 12 last characters of the dongle id prepended with "AutoPi-". The dongle id is the same as the minion id which is used by the dongle to identify itself to the salt-master. The process of producing the minion id can be seen on row 9 in the minion install file[9]:

```
- name: "grep Serial /proc/cpuinfo | awk '{print
$3}' | md5sum | awk '{print $1}' | tee
/etc/salt/minion_id | cut -c21- | sed
's/^/autopi-/g' > /etc/hostname"
```

The minion id is a Message Digest 5 (md5) hash of the Raspberry Pi's serial number found in /proc/cpuinfo. Md5 is a hash function which purpose is to create signatures of large files and is therefore designed to be a fast hash function [9]. It is not intended to encrypt the given input. The serial number is a random string between "00000000" and "FFFFFFFF" with 8 zeros padded in front. 8 hex characters gives a total of 16^8 possible combinations. This means that there are 16^8 possible outputs from the md5 hash function with a serial number as input. So even though the md5 hash is 32 hex characters long, there are only a small subset (16^8) of those combinations used. One can also see that the last 12 characters of the md5 hash (prefixed with "AutoPi-") is used as the hostname of the dongle.

C. Wifi client

The WiFi client is continuously trying to connect to known WiFi networks. If it is connected to a WiFi AP, the WiFi connection will be preferred over the 4G network. This means that the dongle will send all outgoing traffic over the WiFi connection, including its DNS request.

D. Cloud servers

The cloud service can be divided into three distinct parts: The *website*, the *RESTful API* and the *salt-master*.

1) **Website:** The website uses django auth for authentication[10]. Anyone is free to create an account. A dongle is linked to a specific account by entering the dongles dongle id. An account can be linked to multiple dongles. As default, a dongle is only allowed to be linked with one account, but that limit can be increased by the AutoPi staff manually if requested. Most of the websites functionality uses the RESTful API as backend.
2) **RESTful API:** The API service runs over HTTPS and provides a simple way to communicate with the cloud service. Authentication is done using the *Authorization* header of the HTTPS request. There are two types of tokens that can be used to authorize an API call: a *"bearer"*-token or a *"token"*-token.
   The *"bearer"*-token is obtained by providing a valid username and password. The returned token has the JSON Web Token (JWT) format[11]. The JWT token is base64 encoded and separated into three parts: Header, payload and signature.
   The **headers** (in AutoPi's implementation of the JWT) specifies the algorithm used for the signature and that this is in fact a JWT token. The algorithm used is HMAC-SHA256 [7] which is highly secure unless a very simple key is provided during encryption.
   The **payload** contains the username, user id and e-mail of the user that this token is valid for. It also contains the date at which this token becomes invalid. This is set to eight hours.
   The **signature** is, as state previously, created with the HMAC-SHA256 algorithm which takes the headers and payload as input combined with a secret key. This provides integrity as a modification of the headers or payload will invalidate the signature.
   The *"token"*-token is a static value that is used by the dongle to communicate with the cloud server autonomously without any user interaction. It can only be used to upload event data and retrieve custom modules from the cloud.
3) **Salt-Master:** AutoPi uses SaltStack to simplify the infrastructure and communication between their cloud server and the dongles. SaltStack uses a publish and subscribe pattern. The dongles (also known as salt-minions) subscribe to topics and the server (also known as the salt-master) publishes data on those topics. Saltstack's implementation of the pattern ensures that it is the salt-master that initiates all communication.
   The authentication between salt-master and salt-minion are done using a minion id (which is the same as

[8] https://cve.mitre.org/

[9] https://github.com/autopi-io/autopi-core/blob/master/src/salt/base/state/minion/install.sls

[10] https://docs.djangoproject.com/en/2.2/topics/auth/

[11] https://jwt.io/introduction/

the dongle id) and RSA keys. The first time the salt-minion connects to the salt-master, the salt-master saves the RSA public key received from the salt-minion and links it to the corresponding minion id. The salt-minion saves the RSA public key received from the salt-master. This procedure is done before the product is sent to the customer and ensures that the salt-master and the salt-minion have a way to authenticate each other. All subsequent traffic sent between the two is encrypted using Advanced Encryption Standard (AES).

V. METHOD

This chapter introduces the methodology used throughout the work.

A. WiFi hotspot

As all traffic on the WiFi network is securely encrypted, there are not much information gained from sniffing the traffic from a host outside the network. The only information that can be gathered are the MAC-addresses of computers on the network and the SSID used by the access point. This leaves two possible entry points: gaining access by manipulating the back end hostapd application during the WPA2 handshake or gaining access by sending the correct password.

The fastest way to brute force a WiFi network is to catch the 4-way handshake used in the WPA2 protocol to authenticate a client with the AP[12]. These packets can then be used to brute force the password locally. Since the password is a 12 character hex string, there are 16^{12} possible combinations. Using hashcat[13] on a GPU doing ˜180 kHashes/sec would go through all possible 12 hex character passwords (16^{12}) in:

$$\frac{16^{12}}{180000} \approx 50\ years$$

With the knowledge that there are only 16^8 possible dongle id's (from which the WiFi password is taken), one can brute force the passwords in 16^8 tries:

$$\frac{16^{8}}{180000} \approx 6.6\ hours$$

The SSID of the network contains the 12 last characters of the dongle id. The SSID is broadcasted to everyone in the vicinity of the car. This information can be used to deduce the whole dongle id which contains the WiFi password.

All dongle ids with the last 12 characters equal to the 12 characters of the SSIDs is candidates for being the correct dongle id. This method does not require the attacker to catch a WPA2 handshake which means that it can be used without the need for an external user to be connected to the network. This method is also a lot faster since the md5 hash is designed to do fast hashing of large files [9] while the PBKDF2 used in WPA2 is deliberately slow to reduce the effectiveness of brute force attacks [8].

[12]https://www.aircrack-ng.org/doku.php?id=cracking_wpa

[13]https://hashcat.net/wiki/

A program was written in java to exploit this vulnerability. The program took the 12 hex characters of the SSID as input and returned all possible dongle ids. This program went through multiple iterations to optimize the run time. The program was later branched out into two programs using different methods: one using GPU supported brute forcing and one precomputing a wordlist containing all possible dongle ids sorted by their last 12 characters (the part found in the SSID) that can be search through by e.g. a binary search algorithm.

As the first method requires a powerful GPU and the other method requires a lot of disk drive space, both programs were run on a desktop computer. The programs listened on a TCP port for the input SSID and returned the correct hash over the TCP connection which allows a perpetrator to perform the hack remotely within the WiFi range of the car.

B. WiFi client

When connected to the AutoPi through its WiFi hotspot, one can access the local web portal of the device through local.autopi.io. This portal is, among others, used for network configuration and is where the user would configure the 4G or the WiFi connection. However, the WiFi already comes preconfigured with one network. There is a preconfigured WiFi network with SSID "AutoPi QC" and password "autopi2019", which we assume is for the manufacturer's quality control, hence the "QC". To exploit this, a hotspot was set up with these credentials. The AutoPi is configured so that it prioritizes known WiFi networks over a 4G connection. Since the AutoPi is constantly scanning for known WiFi networks, the connection to the fake WiFi hotspot was established in less than a minute and all traffic is directed to the WiFi network instead of via the 4G connection.

C. Cloud servers

To exploit the found WiFi client vulnerability further, a DNS spoofing attack was done. The goal behind doing a DNS spoofing attack is to make the AutoPi believe it is communicating with the AutoPi cloud server, when in fact it is communicating with our "fake" server. Whenever the AutoPi send a DNS request, the response will be the IP address of our fake server, since the AutoPi is connected to our controlled network. As the AutoPi receives the IP address, it will set up a TCP connection to the fake server. AutoPi uses SaltStack for communication between server and dongles. It also send specific event data over HTTPS.

Since the tests in this paper is done directly on the live cloud servers, care have been taken to not disturb the service. Only test that have no way of reading, editing or in some way affect other users data or service have been performed.

Authentication tokens have been modified in different ways to try and gain unauthorized access to send commands to the dongle via the cloud API.

VI. RESULTS

This chapter describes the findings of the work.

A. WiFi hotspot

The two vulnerabilities found compliments each other which makes the WiFi hotspot, using the default SSID and password, exploitable. The first vulnerability is that the dongle ids are derived from a input with a 8 hex character variance. This reduces the possible subset of dongle ids from 16^{32} to 16^8 and possible passwords from 16^{12} to 16^8. The other vulnerability is that the last 12 characters of the dongle id is broadcasted as the SSID. This, in combination with the first vulnerability, allows for a faster brute force attack without the need to catch a WPA handshake.

Since the method used to derive the password from the SSID is done by taking 12 characters of the hash and trying to find the whole 32 character hash, the method could return multiple candidates since multiple hashes might have the same 12 last characters. The probability of a evenly distributed 32 hex character hash having the same last 12 characters is:

$$\frac{16^{20}}{16^{32}} = \frac{1}{16^{12}}$$

This probability is the same as the probability of at least two dongles having the same SSID. Since it is such a small number, it is negligible.

As stated before in the method paragraph, the end result where two programs utilizing different methods: one using GPU supported brute forcing and one precomputing a wordlist containing all possible dongle ids sorted by their last 12 characters (the part found in the SSID) that could searched through with a binary search algorithm. The code can be found in the Appendix of this document.

The GPU program is written in java with CUDA[14]. Running the program on a GeForce GTX 1060 going through all 16^8 possible combinations took <1 second.

The wordlist created with the second method contained 16^8 hashes with every hash being 16 bytes (128 bits). This gave a file size of:

$$16^8 \cdot 16 \approx 69\ GB$$

Using a binary search algorithm on the sorted list with 16^8 hashes gives a maximum time complexity of:

$$\log_2 16^8 = 32$$

B. WiFi client

We are not quite sure if the preconfigured WiFi interface is just a random error or a production flaw. But we know for certain that the two AutoPi dongles which we have access to, came preconfigured with the ”AutoPi QC” WiFi network and with the same ”autopi2019” password. Therefore, it is possible to set up a WiFi hotspot using this information and the AutoPi dongle will in a short time connect to that hotspot, without the owner being aware of it. The one in control of the hotspot can then perform several attacks such as traffic sniffing or DNS spoofing.

[14]https://developer.nvidia.com/cuda-zone

The dongle includes its hostname in the DHCP discovery broadcasted to the DHCP server. The dongles hostname contains the last 12 characters of the dongle id.

The Iptable rules for the WiFi client interface only allows related connections, forwarding and output[15]. This means that we were able to reach hosts on the dongles internal network via the forwarding rule, but all traffic directed directly towards the dongle is dropped under the input rule. We were therefore only able to reach the dongle directly when it sets up outgoing connections.

C. Cloud servers

By performing a DNS spoofing attack, the AutoPi dongle can be tricked into believing it is communicating with the AutoPi Salt-Master server. As the dongle receives the response of the DNS request with the fake IP address, it will try to set up a TCP connection with that server. However, the AutoPi dongle and server uses RSA keys for authentication during the SaltStack handshake. The dongle sets up the TCP connection and sends its public RSA key which then gets to the fake server. It also identifies itself with its minion id, which is the same as the dongle id, and contains the SSID and WiFi password. When the fake server responds with its public key, the connection is shut down since the AutoPi dongle notices that it is not matching the real AutoPi Salt-Master key. Hence, the DNS spoofing attack was not successful. Any man-in-the-middle attack is futile.

The dongle does also send event data over HTTPS to the server. Since HTTPS needs a valid certification, in this case for the domain ”autopi.io”, the dongle will not send any data to the fake server. The HTTPS sent from the dongle uses the *”token”*-token in the authorization header. This is the weaker authentication with very limited use. So even if one is able to fake a valid certificate, the HTTPS data and the intercepted token would not be to any great use.

VII. DISCUSSION

Depending on what add-ons is combined with the dongle, AutoPi presents a load of features. It is truly a product that brings a great upgrade to the car. But is it secure?

The premise of the AutoPi service is to let its end user have full control over their product. To accommodate this, restrictions have to be relaxed to allow custom code and modifications. This leads greater damage potential for found vulnerabilities and it is therefore important to have a very secure “outer layer”. AutoPi achieves this by using external libraries and software that have proven themselves to be secure. Everything sent from the dongle and cloud servers are encrypted ensuring confidentiality, integrity and authentication.

The found vulnerabilities stems from human configuration errors rather than vulnerable software. The vulnerability found regarding the WiFi credentials can be exploited on any AutoPi dongle using the default WiFi settings. There is no way of knowing exactly how many dongles that are

[15]https://github.com/autopi-io/autopi-core/blob/master/src/salt/base/state/wlan/hotspot/iptables-ipv4.rules (interface wlan0)

vulnerable to the exploit. Since the default password seems to be a 12 hex character long random generated string, it might give people the illusion of being secure and it would thus reduce the amount of people changing the password.

Because of the previously mentioned balance between availability and security, the found exploit gives a perpetrator full root access to the dongle.

The AutoPi is marketed as product with various features. Depending on the vehicle combined with the AutoPi, the execution of certain operations can be achieved. One such operation is to record and replay commands sent on the vehicles CAN bus. All communication to the ECUs goes through the CAN bus. On certain car models, commands such as unlocking the vehicle and starting the engine runs on the CAN bus. Hence, the manufacturer has provided a feature that lets the one in control of the AutoPi unlock and start the car.

There is a substantial amount of actions that can be performed by controlling an AutoPi unit connected to a car. But the most severe is the controlling of the CAN bus. By being able to send commands on the CAN bus, the actions of the vehicle can be manipulated. Hence, raising a serious amount of safety and security issues.

VIII. FUTURE WORKS

Since this paper has been done independently of AutoPi, there are a lot more to test regarding the cloud service. Great care have been taken to not affect the service of the cloud servers which constrains the amount of test that can be done and how thorough those tests can be.

The tests in this paper have been performed on a device with default settings and no extra add-ons. The premise of the AutoPi dongle is to allow implementation of custom code and adding extra hardware. This is something that could be looked into further. Examples are the Bluetooth module and the USB ports. Since they are not used with default software and hardware, there have been no security testing of them in this paper.

IX. CONCLUSIONS

This paper shows that a product might have vulnerabilities even though the development of the product have been heavily security focused. A simple oversight regarding the generation of the SSID and password of the device led to a security exploit in an otherwise very secure device.

ACKNOWLEDGEMENT

We would like to thank our supervisors Robert Lagerström and Pontus Johnson for their support and guidance throughout the entire work.

REFERENCES

[1] A. Meola "Automotive Industry Trends: IoT Connected Smart Cars & Vehicles", Business Insider, Dec 2016.
Available: https://www.businessinsider.com/internet-of-things-connected-smart-cars-2016-10?r=US&IR=T

[2] Ericsson, "Digital transformation and the connected car", Ericsson Mobility Report, Nov 2016.
Available: https://www.ericsson.com/assets/local/mobility-report/documents/2016/emr-november-2016-digital-transformation.pdf

[3] European Parliament, "DIRECTIVE 98/69/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 October 1998 relating to measures to be taken against air pollution by emissions from motor vehicles and amending", page 21, paragraph 8.2, Oct 1998.
Available: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1998L0069:19981228:EN:PDF

[4] R. Currie, "Hacking the CAN Bus: Basic Manipulation of a Modern Automobile Through CAN Bus Reverse Engineering", SANS Institute Information Security Reading Room, page 2. paragraph 1, May 2017.
Available: https://www.sans.org/reading-room/whitepapers/threats/paper/37825

[5] International Organization for Standardization, "Road vehicles – Controller area network (CAN)", ISO 11898-1, page 5, paragraph 6.1, Dec 2013.
Available: http://read.pudn.com/downloads209/ebook/986064/ISO%2011898/ISO%2011898-1.pdf

[6] W. Xiong, F. Krantz, and R. Lagerström, "Threat modeling and attack simulations of connected vehicles: a research outlook," in the Proc. of the 5th International Conference on Information Systems Security and Privacy (ICISSP), page 2, paragraph 2.4, Feb 2019.

[7] D. Eastlake and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", Internet Request for Comments, vol. RFC 4634, page 14, paragraph 7, Jul 2006.
Available: https://tools.ietf.org/html/rfc4634

[8] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0", Internet Request for Comments, vol. RFC 2898, page 8, paragraph 5.2, Sep 2000.
Available: https://www.ietf.org/rfc/rfc2898.txt

[9] R. Rivest, "The MD5 Message-Digest Algorithm", Internet Request for Comments, vol. RFC 1321, page 0, paragraph 1, Apr 1992.
Available: https://www.ietf.org/rfc/rfc1321.txt

[10] Raspbian, "Raspbian FAQ", paragraph "What is Raspbian?", Apr 2019.
Available: https://www.raspbian.org/RaspbianFAQ#What_is_Raspbian.3F

[11] J. Cichoniski and J. Franklin, "LTE Security – How Good Is It?", RSA Conference 2015, slide 34, Apr 2015.
Available: https://www.rsaconference.com/writable/presentations/file_upload/tech-r03_lte-security-how-good-is-it.pdf

[12] A. Guzman and A. Gupta, "IoT Penetration Testing Cookbook", Packt Publishing Ltd., Nov 2017.