FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Proof of Concept of Hacking Cryptocurrency Hardware Wallets

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Markus Reichel
Matrikelnummer 01529191

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Mitwirkung: Univ.Lektor Dipl.-Ing. Dr.techn. Adrian Dabrowski

Wien, 3. Mai 2019

Markus Reichel	Edgar Weippl

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Proof of Concept of Hacking Cryptocurrency Hardware Wallets

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Markus Reichel

Registration Number 01529191

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Assistance: Univ.Lektor Dipl.-Ing. Dr.techn. Adrian Dabrowski

Vienna, 3rd May, 2019

Markus Reichel	Edgar Weippl

Erklärung zur Verfassung der Arbeit

Markus Reichel
Brünner Straße 111-113
2201 Gerasdorf bei Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Mai 2019

Markus Reichel

Kurzfassung

Um Kryptowährungsbestände zu sichern bedarf es an Langzeitsicherheit. Viele Hardware-Cryptocurrency-Wallets wurden daraufhin entwickelt, da diese eine Möglichkeit anbieten, den privaten Schlüssel offline aufzubewahren. Diese Arbeit ist in zwei Teile geteilt. Die erste Hälfte analysiert fünf bestehende Wallets in ihrer Software- und Hardwaresicherheit und Attestation-Methoden. Danach wird eine Klassifikation bereits bestehender Attacken von diesen Wallets präsentiert.

Die zweite Hälfte zeigt einen Proof-of-Work eines Supply-Chain-Angriffs, bei dem ein USB-Gerät ein solches Hardware-Wallet emuliert und das Opfer dazu bringt, sein Kryptovermögen an den Angreifer zu überweisen. Es zeigt sich, dass die Wallet-Software der Hardware zu sehr vertraut und so ein Wallet emuliert werden kann. Deswegen fehlen den Geräten immernoch wichtige Attestations.

Abstract

Securing cryptocurrency funds require long-term safety. A lot of hardware cryptocurrency wallets were developed, because they offer a way to store the private key offline. This thesis is split up in two parts. The first half analyzes five existing wallets in terms of their software and hardware security design and attestation methods. Then, it presents a classification of exiting attacks for these wallets.

The second half provides a proof of a work of a possible supply chain attack by emulating a hardware wallet on a USB device and tricking the victim into sending his funds to the attackers cryptocurrency account. It proves that wallet software puts too much trust in the hardware and that such a hardware wallet can be emulated. Therefore, the devices are missing important attestations.

Contents

CHAPTER 1

Introduction

In 2008, a whitepaper [1] from a person with the pseudonym "Satoshi Nakamoto" appeared in the internet about the design of the cryptocurrency Bitcoin. Shortly after that, in 2009, the first Bitcoin Software came available [2]. Since then, decentralized cryptocurrencies such as Bitcoin are rising in terms of popularity and acceptance [3], regardless of a lot of up and downs. Also, multiple hundreds of such cryptocurrencies were developed. With their increasing use, also the security characteristics were getting more important. Cryptocurrencies rely, as the name says, on cryptography, and in order to do transactions, a user needs to authenticate with his private key.
In the beginning, special software called "wallets" was developed to be responsible for storing the private key and managing it, but recently, more and more companies are developing hardware wallets, which are physical devices [4]. The idea behind them is that if the PC is compromised, important parts of the pay process are still done in the supposedly safe hardware.
However, this does not solve all security relevant problems, because a physical device can also be manipulated or stolen. Therefore, these devices must also be evaluated in terms of security proficiency. This bachelor thesis will first investigate security properties of the most prominent solutions in the cryptocurrency hardware wallet market. Then, we will create a proof of concept to show that cryptocurrency hardware wallets can be hacked when gaining physical access of the device.

In in this work, the following scenario is evaluated:
The victim gets a fake device from a supply chain attacker, e.g., a flash drive sized computer like the USB armory [5], which speaks the protocol of the host application over USB. Within this scenario, the following questions arrive:

1. Is it possible to emulate a hardware wallet on a USB armory?
2. Would the victim notice the attack?

In order to implement the fake device which should disguise itself as a hardware wallet, information about the specific implementation of the protocol of the wallet was collected. We compare the existing work where such wallets got compromised with each other. Papers and blogs regarding the protocol for some vendors already exist (see Section 3). If it is not information about the protocol, it has to be reverse engineered with a sniffer for the USB traffic.

CHAPTER 2

Background

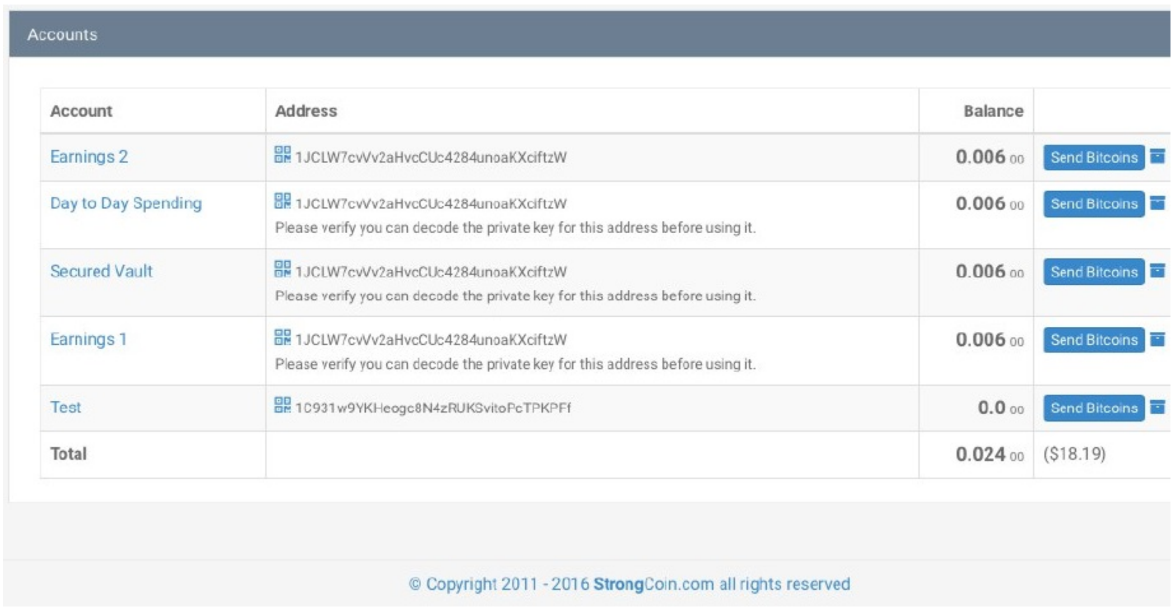
In order to work in the field of cryptocurrency hardware wallet security, understanding several fundamental concepts of such a system is essential. Therefore, this section will contain a broad sprectrum of topics necessary to understand the basics of a wallet.

2.1 Cryptocurrency Wallets

Cryptocurrencies like Bitcoin use asymmetric cryptography to control the access of the money. The coins are not stored as a plain amount but in a huge series of blocks of transactions, called a blockchain. With this, the public key acts like a receiver address of a transaction, like an IBAN. However, the public key can only be used to receive money or follow the transactions, because only with the private key, a transaction can be signed right to send money, similar to a nearly ideal manual signature. Therefore, in contrast to the public key, the cryptocurrency owner has to protect the private key because getting this key is synonymous with getting control over all the money of this account. There are several ways to store such a private key. In this work, we differentiate between hot and cold wallets [6].

2.1.1 Hot Wallets

Hot wallets are typically software wallets running on a device connected to the internet like a PC or a smartphone. With web wallets (see Figure 2.1), the private keys are stored on the servers of the provider, so one needs special trust in the company running those services. In terms of security, hot wallets all share the risk that the private key can be compromised over the internet. When e.g. the database of a cryptocurrency online exchange gets hacked, all of the customers' private keys are leaked. It is recommended to use hot wallets like a purse, particularly to not store a huge amount in one wallet.



Accounts

Account	Address	Balance	
Earnings 2	1JCLW7cvVv2aHvcCUc4284unoaKXciftzW	0.006 00	Send Bitcoins
Day to Day Spending	1JCLW7cvVv2aHvcCUc4284unoaKXciftzW Please verify you can decode the private key for this address before using it.	0.006 00	Send Bitcoins
Secured Vault	1JCLW7cvVv2aHvcCUc4284unoaKXciftzW Please verify you can decode the private key for this address before using it.	0.006 00	Send Bitcoins
Earnings 1	1JCLW7cvVv2aHvcCUc4284unoaKXciftzW Please verify you can decode the private key for this address before using it.	0.006 00	Send Bitcoins
Test	1C931w9YKHeogc8N4zRUKSvitoPcTPKPFf	0.0 00	Send Bitcoins
Total		0.024 00	($18.19)

© Copyright 2011 - 2016 StrongCoin.com all rights reserved

Figure 2.1: The web inteface of strongcoin, a hot wallet [7].

2.1.2 Cold Wallets

Cold wallets are not connected to the internet and can be used to deposit the money. Just writing the private key on a paper ("paper wallet") to lock it up in a safe is the simplest example. Also special hardware wallets (see Figure 2.2), devices which store the private key offline, fall under this category. When the attacker has no physical access to the device, the only risk of getting hacked is when the hardware wallet is connected to a computer. There are several approaches to make a hardware wallet secure, some devices strive for openess of the system, while e.g. Ledger builds wallets where the private key is stored in a closed secure element [4]. Section 5 will give a more detail overview of the different security approaches.

2.2 Hardware Security Basics

The main goal of a hardware cryptocurrency wallet is to protect the private key. In order to analyze its security, it is necessary to go into detail about their architectures and attestation methods.

2.2.1 Hardware Security Architecture

Most modern cryptocurrency hardware wallets are build with a common architecture. They are embedded systems, in their cores consisting of a microcontroller. Rather than a microprocessor, a microcontroller unit (MCU) today is a system-on-a-chip solution with RAM and ROM together on the same chip with the arithmetic logic unit (ALU) and other

Figure 2.2: The Trezor Model T, a cold wallet [8].

peripherals such as clocks and signal generators. Coprocessors to speed up cryptographic algorithms are also popular. The firmware, which is how the software running on the microcontroller is called, communicates over an interface (mostly USB). On the host side, software on the PC receives the data and processes it further. The two main computer architectures are von Neumann, where data and instructions share the memory space, and Harvard, where data and instructions are seperated. Harvard offers more protection against so called "micro-probing" attacks [9], where the CPU is interrupted and data can be read from the data bus. It is possible to use debugging interfaces to extract and manipulate the firmware, so designers try to hide or remove them. Companies also make the housing tamper-resistant with coatings or encapsulations, for example with epoxy. However, for an attacker, it is technically possible to open the casing to decap microcontroller and memory chips to read out the data [9]. Other attacks like fault attacks where the supply voltage is manipulated to skip logic in the firmware or side-channel attacks where a channel like the voltage leaks critial information also exist. For this reason, secure elements were developed. They are designed to contain confidential data like cryptographic keys or IDs and to be tamper-resistant. A wide-known example of secure elements are smartcards, which are widely used in the fields of telephony, passports, pay TV and banking. These contain mechanisms such as light sensors to detect chip openings, an obfuscated layout and are designed against side channels and fault detection. The common criteria (CC) [10] framework aims to evaluate products in terms of security, with a rating ranging at the beginning from EAL1 to EAL7 at the end (with an optional + for higher features). Typical smart cards are classified as EAL4+. Section 5 contains a table with the security certifications of the hardware wallets.

2.2.2 Remote attestation

To create a secure enviroment where manipulation can be detected, one can use hardware and software mechanisms to create a concept called remote attestation, which etablishes trust in a device. Attestation defines a verifier, who verifies the state of a prover over an attestation protocol [11]. There are three remote attestation types [12]:

1. Hardware-Based Remote Attestation: The hardware supports protecting keys and summarizing the state of the system as a hash, often realized as a Trusted Plaform Module (TPM).
2. Software-Based Remote Attestation: Witout any additional hardware it is possible to develop checksum algorithms which include run-time side-effects for attestation.
3. Hybrid Remote Attestation: This attestation uses hardware and software approaches together in order to get the advantages from both (e.g. get immutability via read-only memory from the hardware and exclusive resource access control via a formally verified firmware).

Adversary models on remote attestation are: (i) remote adversaries, (ii) local adversaries and (iii) physical adversaries. Remote adversaries attack the software of the platform via injecting code in the network, while local adversaries are capable of manipulating and sniffing the communication channel of the prover. Physical adversaries have full access to the hardware of the device and use side channels or physical memory extraction to manipulate it. Remote attestation is mainly able to deflect type i and ii of adversaries, while tamper-resistant methods mentioned in Section 2.2.1 target type iii. The focus of our proof of concept is a supply chain attack, therefore, we choose the physical adversary as our model.

2.3 Hardware Wallets

In Section 2.1.2, we classify hardware wallets as cold wallets. They store the private key in their memory, either on a normal chip or a secure element (see Section 2.2.1). Access to it is easier to control in contrast to a software wallet, because every interaction with the wallet goes over the USB port and is controlled by various types of attestations. Regarding the workflow, after purchasing a hardware wallet, it must first be setup in order to make transactions.

2.3.1 Setup

Hardware wallets always have to be initialized before usage. In addition to general information such as the device name, important information like the cryptocurrency account has to be generated on the device. It depends on wether the new wallet comes with shipped firmware or needs software and an internet connection for initialization, but in general it will generate a random seed phrase with a procedure standardized in Bitcoin Improvement Proposal 39 (BIP39) [13]. The words contained in this phrase are mnemonics, which can together with an optional passphrase be converted into a binary seed used to create a deterministic wallet. Therefore, a mnemonic phrase can also be used to backup an existing wallet, assuming that the words are written down when they were shown at setup.

2.3.2 Transactions

After that, the user can send cryptocurrencies with the hardware wallet. An advantage of the asymmetric cryptography is the fact that the private key never has to be revealed by the hardware wallet. The host can build a transaction and send it to the wallet, which signs it with the key and then sends the signed transaction back to the host. The host is now able to send the signed transaction to the cryptocurrency network without the knowledge of the private key.

2.4 Universal Serial Bus (USB)

Every in this thesis evaluated hardware wallet uses the Universal Serial Bus (USB) to communicate with the host computer. Simply put, USB [14] is a serial bus system where the communication works asymmetrically. The computer acts as the host part who issues requests and the device acts as the device part, answering those.

2.4.1 Protocols

It was already mentioned that the USB protocol is host-centric, which means that the so called USB transactions are initiated from the host. USB uses four types of lowlevel packets: (i) token packets which act like headers, (ii) data packets which contain the payload, (iii) handshake packets used for transaction logic and (iv) start-of-frame packets for the synchronization of the transmission. However, when writing software for USB, a programmer will typically not get in touch with these packets because the hardware (in case of a PC the so called USB host controller) already handles these lowlevel messages. Still, a lot of devices rely on a protocol stack with multiple layers, e.g. the hardware wallet Trezor One uses the USB HID (Human Interface Device) - protocol to implement the wallet-specific commands.

2.4.2 Endpoints

Endpoints are a central concept in USB, as they allow a single physical connection to be split up logically. They are like a device-side socket like in TCP/IP. It is very common that a USB device offers multiple endpoints, e.g., the Trezor One offers one endpoint for the regular communication and one endpoint for two-factor-authentication. An example endpoint configuration is shown in Figure 2.3. As the bus is host-controlled, the device cannot send something over the bus by itself, therefore, there is a seperate in and out buffer for every endpoint.
The communication between the host and an endpoint is called pipe. As described in Section 2.4.1, the lowlevel packets are not important from a programmer's perspective,

but the higher-level USB pipe has, in addition to parameters like maximum bandwidth per pipe, the so called transfer type. With this, an endpoint can have four different transfer types:

1. Control Tranfers: This transfer-type is used for commandlike messages or status updates.
2. Interrupt Transfers: As the name says, they deliver device data to trigger interrupts. Input devices like mice and keyboards use them.
3. Isochronous Transfers: Here, the packets are sent continuously in a constant interval. Video and audio streams are example applications.
4. Bulk Transfers: These are used to send big chunks of data, e.g. printer jobs. Error-checking is used to ensure the integrity.

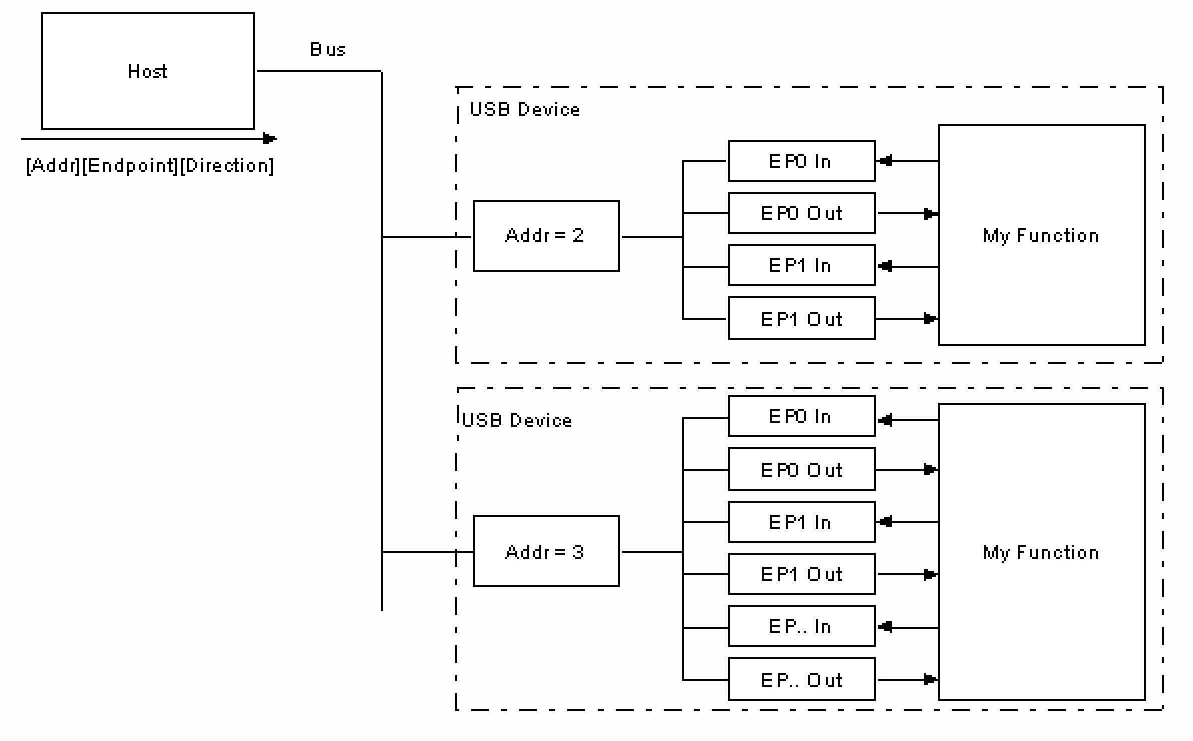


Figure 2.3: Logical view of a host connected to two devices (in USB called "functions"). Every function has multiple endpoints [14].

2.4.3 Descriptors

Especially in case of USB, there are many different types of devices which have a variety of different functions (device classes). When a USB device is plugged into the computer, the operating system needs a way to identify not only the device but also the capabilities and the protocols it uses. This is implemented with the USB descriptors; they form a hierarchical network of information about the device which can be retrieved from the host. Hardware manufacturers have to ensure that the device responds with these descriptors.

1. Device Descriptors: contain general information about the device like the name, device class and manufacturer.
2. Configuration Descriptors: used to configure power consumption and get the number of interfaces.
3. Interface Descriptors: describe multiple endpoints bundled together into one logical interface.
4. Endpoint Descriptors: contain endpoint information like transfer type and address.
5. String Descriptors: are optional, but add human-readable information about the device into the description.
6. Report Descriptors: It is used if the USB device is a HID device. In order to reduce the amount of drivers the operation system has to provide, the USBHID protocol provides report descriptors to describe the message format used by HID devices. For example a mouse report descriptor contains information how the two buttons, the wheel and the x/y speed is sent.

CHAPTER 3

Related Work

Recently, Roth et al. [15] gave a talk about several cryptocurrency hardware wallet vulnerabilities. The explained a wide range of attacks, from side-channel attacks and fault attacks to lowlevel attacks. The proof of concept in this work reverse engineers and emulates the Trezor One low-level protocol. Similar work has been done for the Ledger Nano and Ledger Nano S. Gkaniatsou et al. [16] showed that low-level protocols of hardware wallets (in this case Ledger devices) can be attacked. They analyzed the Ledger-specific protocol, categorize and explained possible attacks and then offered fixes for the protocol. Other related work includes attacks on the cryptocurrency protocols itself and attacks on embedded systems which are similar to hardware wallets. Datko et al. [17] presented in a talk at DEF CON 2017 that the similar architecture that Trezor and Keepkey hardware wallets share is vulnerable to fault attacks (in the presentation also called "glitches") which can be used to skip critical code sections such as password-checking routines.
But before cryptocurrency hardware wallets were analyzed in terms of security, a lot of work on attacking cryptocurrencies itself such as Bitcoin was done. The big Bitcoin exchange MtGox filed for bankruptcy [18] after beeing attacked with the so called transaction malleability, which made it possible for a user to issue a withdrawal while making the exchange belief that the transaction failed, which resulted in a double withdrawal. Rosenfeld [19] analyzed the possibility of double spending with hash-based attacks. The danger of double spending was also already mentioned in the initial Nakamoto whitepaper [1]. An example for another cryptocurrency and another attack vector is the work of Atzei et al. [20], who did a survey on attacks on Ethereum smart contracts. Ethereum introduced this small programs which should be executed in a network containing mutually untrusting nodes, however, the platform contained several security vulnerabilities.
In the hardware security field, there is a wide selection of attack approaches available. General vulnerability reviews such as the one from Dalton et al. [21] looked at known

threats such as buffer overflows, format string attacks and information disclosure to discuss the effectivity of novel hardware architectures against these. Roland et al. [22] introduced attacks against platforms with secure elements, in this case secure elements on smartphones, which are attacked via NFC. In the end, they were able to not only perfom a denial of service (DoS) attack against the secure element, but also remotely use the element without knowledge of the victim.

CHAPTER 4

Methology

The work of this thesis is split into two parts, a market review of hardware wallet security features to get an overview about hardware wallet security, and a specific proof of concept (PoC) to show weaknesses in the security model.

4.1 Methology of the Security Feature Review

Before a specific proof of concept was implemented, classification of the field of hardware cryptocurrency wallet security was needed. We did not only need know about the architecture of prominent wallets and their security approaches, but also about existing attacks to get an understanding of common attack vectors. Literature research was the primary method to gather information about the wallet architecture. Especially in the blogs of the individual companies, information about existing vulnerabilities could be found, which were then classfied in terms of type. After that, the wallets were initialized and analyzed hands-on in order to get their attestation methods and outer security mechanisms. Important differences between the wallets could be summed up in tables.

4.2 Methology of the Proof of Concept

With the finished review, we decided for which wallet the proof of concept is going to be implemented in order to demonstrate further vulnerabilities with hardware wallets. The hardware wallet approach assumes that the PC software should not be trusted, but this does not justify that the hardware can be trusted. This resulted in the goal to emulate a hardware wallet. The emulation is build on a USB stick with software capabilities of a full computer, because one can profit from the rich software libraries and tools available. Therefore, a flash drive sized computer was used. This malicious wallet had to show a fake cryptocurrency address to trick the victim into sending it to the evil account. Because wallets communicate over USB, a full USB stack had to be emulated. The proof of concept could be easily verified by checking if the software thinks a wallet is plugged in and the malicious address works as described.

CHAPTER 5

Market Review of Hardware Wallet Security Features

Before the proof of concept was implemented, research about the five hardware wallets was done. After creating a basic overview, the attestations in respect of the different wallets were examined from hardware to software and then, the unpacking and initialization procedures of the wallets were analyzed. In the end, already known security holes were collected and classified.

5.1 General Overview

Five hardware wallets were chosen for the study, which were seen as the most prominent devices. The selected wallets were: (i) Trezor Model One, (ii) Trezor Model T, (iii) Keepkey, (iv) Ledger Nano S and (v) Ledger Blue. The openess of the platforms can be seen in Table 5.1. These wallets have very different hardware characteristics, as seen in Table 5.2.

Fully Open Source	**Software**	**Firmware**	**Hardware**
Trezor One	yes	yes	yes
Ledger Nano S	yes	no	no
Keepkey	yes	yes	no
Trezor T	yes	yes	yes
Ledger Blue	yes	no	no[1]

[1] Only development version: `https://github.com/LedgerHQ/blue-schematics`.

Table 5.1: Openess of the platforms [23] [24].

Architecture	**Microcontroller**	**Secure Element**	**Certifications**
Trezor One [25]	STM32F205	n/a	n/a
Ledger Nano S [4]	STM32F042K	ST31H320	CC EAL 5+
Keepkey [26][27]	STM32F205RGT6	n/a	FIPS PUB 140-2, FIPS PUB 180-2
Trezor T [25]	STM32F427VIT6	n/a	n/a
Ledger Blue [4]	STM32L476	ST31G480	CC EAL 5+

Table 5.2: Architectural overview: microcontrollers and certifications.

5.2 Attestation Methods

Regarding the attestation, the packaging and hardware was observed, and on the websites, information about the firmware and software attestations were gathered. Note that Ledger states that hologram stickers are just a "false impression of security" because they can be easily duplicated [28].

5.2.1 Packaging

In the beginning, the packaging of the wallets was inspected by ourselves (see Table 5.3).

Packaging	**Verified by User**
Trezor One	Two hologram stickers, sealed with strong glue. Plastic foil on the device.
Ledger Nano S	No special sealing.[1]
Keepkey	One hologram sticker, package tampering is barely visible.[2] Plastic foil on the device.
Trezor T	One hologram sticker on the USB port of the device.
Ledger Blue	No special sealing.[1]

[1] Company says that it does not need temperevident packaging because of its strong device security.
[2] The package can be closed again and the indicator that it has been openend is only barely visible.

Table 5.3: Shows how the devices were shipped to the customer.

5.2.2 Hardware

Then, the hardware attestations of the devices were gathered (see Table 5.4).

Hardware	**Verified by User**
Trezor One	Plastic, ultrasonically welded to be tamperproof.
Ledger Nano S	Plastic, users can verify the integrity of the hardware with images of the printed circuit board (PCB) online.
Keepkey	Anodized aluminium case, build to be tamperproof.
Trezor T	Plastic, ultrasonically welded to be tamperproof.
Ledger Blue	Plastic and aluminium. Users should verify the integrity of the hardware with images of the PCB online, there are no pictures of the PCB!

Table 5.4: Shows how the casings are protected.

5.2.3 Bootloader

Every device needs to check the autenticity of the bootloader with their security model (see Table 5.5).

Bootloader	**Verified by Firmware**	**Verified by Secure Element**
Trezor One	Authenticity is checked by the firmware via a SHA256 hash.	n/a
Ledger Nano S	n/a	Checks the authenticity. Also times sending.
Keepkey	Authenticity is checked by the firmware via a SHA256 hash.	n/a
Trezor T	Authenticity is checked by the firmware via a SHA256 hash.	n/a
Ledger Blue	n/a	Checks the authenticity. Also times sending.

Table 5.5: Shows how the bootloader is checked.

5.2.4 Firmware

The firmware is one of the critical parts and has a lot of attestation methods (see Table 5.6).

Firmware	**Verified by User**	**Verified by Bootloader**	**Verified by Software**	**Verified by Secure Element**
Trezor One	Not shipped with firmware.[1] Optional PIN (1-9 digits) and optional passphrase. Default: 24-word recovery.	Checks the signature (signed by SatoshiLabs).	The Trezor wallet checks the signature (signed by SatoshiLabs).	n/a
Ledger Nano S	PIN (4-8 digits) and optional passphrase. Default: 24-word recovery.	n/a	Ledger live checks the signature of the secure element firmware (signed by Ledger).	Checks the authenticity of the MCU code. Also times sending.
Keepkey	Optional PIN (1-9 digits) and optional passphrase. Default: 12-word recovery.	Checks the signature (signed by Keepkey).	The Keepkey app checks the signature (signed by Keepkey).	n/a
Trezor T	Not shipped with firmware.[1] Optional PIN (1-9 digits) and optional passphrase. Default: 24-word recovery.	Checks the signature (signed by SatoshiLabs).	The Trezor wallet checks the signature (signed by SatoshiLabs).	n/a
Ledger Blue	PIN (4-8 digits) and optional passphrase. Default: 24-word recovery.	n/a	Ledger live checks the signature of the secure element firmware (signed by Ledger).	Checks the authenticity of the MCU code. Also times sending.

[1] Must be installed on first use.

Table 5.6: Shows how the firmware is checked.

5.2.5 Software

At last, the software was measured in terms of its attestation methods (see Table 5.7).

Software	**Verified by User**
Trezor One	Access to the private keys must be confirmed on the device. Enforced to set a PIN, but can be disabled.
Ledger Nano S	Access to the private keys must be confirmed on the device. Enforced to set a PIN. Shows a security checklist to ensure that a genuine device was bought.
Keepkey	Access to the private keys must be confirmed on the device. Enforced to set a PIN.
Trezor T	Access to the private keys must be confirmed on the device. Enforced to set a PIN, but can be disabled.
Ledger Blue	Access to the private keys must be confirmed on the device. Enforced to set a PIN. Shows a security checklist to ensure that a genuine device was bought.

Table 5.7: Shows how the software is checked.

5.3 Wallet Initialization

The process of unpacking and initializing the wallets is particulary interesting. This is because Section 5.2 shows that a lot of attestation is done by the user or can be observed, so this process of initializing the wallets makes is clearer how attestation is performed.

5.3.1 Opening

Trezor One: The manufacturer first hints that the seals should be intact and the device should be sealed in a plastic foil. The packaging is also sealed with strong glue, so opening the box ruptures is visable.

Ledger Nano S: A simple box with no additional security mechanisms. The manufacturer notifies the user on its website to check the origin, content, and that the device is not initialized. Advanced users can also check the hardware integrity by themselves with pictures of the PCB on the vendor website to ensure no hardware tampering did take place.

Keepkey: A black box with the hologram. There are no hints in the getting started guide about the physical appearance of the box and the device. However, there is also plastic foil on the joining of the case parts.

Trezor T: The packaging can be opened without destroying it. The manufacturer first hints that the USB Port of the device has a seal.

Ledger Blue: A simple box with no additional security mechanisms. The manufacturer notifies the user on its website to check the origin, content, and that the device is not initialized. Advanced users can also check the hardware integrity by themselves to ensure no hardware tampering did take place. However, the official Ledger support article only contains pictures of the Nano S PCBs [29].

5.3.2 Setting up

Trezor One: The device prompts "visit trezor.io/start". Therefore, you are not able to set it up on an offline computer without setting up your own webserver [30]. On the website, the manufacturer explicitely notifies the user about the holograms from the packaging and supports him with the installation routine. The software "Trezor Bridge" has to be installed. Then, the website says that it is time to install the firmware, because the trezor did not get shipped with existing firmware. After the firmware is installed, a new wallet can be created or an old wallet can be recovered with the backup words. After creating a wallet, the Trezor needs a backup. The user is notified to not make a digital copy of the recovery seed (24 English words). Then, you can also set a name and a PIN lock with max. 9 digits. After that, the device is ready. Note that everything works over the browser. The Trezor Bridge sets up a webserver on localhost, where the Trezor wallet app fetches everything over POST.

Ledger Nano S: The firmware was already on the device. There, the user can set it up and set a pin directly on the device. Then, the recovery phrase can be written down (also 24 words). It also has to be confirmed with the two buttons on the device. After that, the device is ready. All this could be done offline. Then, on "start.ledgerwallet.com", additional apps may be installed. No cryptocurrencies are installed, only the settings are available. Now, Ledger live has to be installed. This security checklist is really helpful. It asks the user about the setup and checks the firmware. Ledger live is also able to be secured by a password. After that, the cryptoasset apps can be installed.

Keepkey: The cover of the box refers to "keepkey.com/getting-started". Connecting the device only results in the Keepkey logo on the screen. Keepkey's software is a Chrome app which as to be downloaded. The firmware and bootloader updates are done via the Chrome app. Also the device initialization works over the app. Note that the recovery sentence is only 12 words long. Then, the Chrome app shows the accounts, but only when connected to the Internet.

Trezor T: The device prompts "visit trezor.io/start". Therefore, you are not able to set it up on an offline computer without setting up your own webserver [30]. On the website, the manufacturer explicitely notifies the user about the holograms on the USB port and supports the user with the installation routine. The software "Trezor Bridge" has to be installed. Then, the website says that it is time to install the firmware, because the trezor did not get shipped with existing firmware. This works as the same like the Model One, because it uses the same software. Therefore, we will this time try to recover the seed from the Model One: The seed has to be typed on the Trezor screen. The user does not have to set a PIN, however, a warning will be displayed on the Trezor. Then, the Trezor T is ready.

Ledger Blue: The firmware was already on the device. The user can set it up and set a pin directly on the device as the Nano S. We will recover the words from the Nano S. After that, the device is ready. All this could be done offline. Then, on "start.ledgerwallet.com", additional apps may be installed. No cryptocurrencies are installed, only the settings are available. Now, Ledger live is needed again. Now, everything works just as with the Nano S.

5.4 Existing Vulnerability Classification

Finally, this study collected existing vulnerabilities for the five mentioned hardware wallets and classify them in terms of the attack vector. Table 5.8 displays the collected existing vulnerabilities of the five hardware wallets. It can be seen that there were a wide spectrum of attacks possible, from using the power of the device as a side channel [31] to using temperature to read the private key from the RAM [32]. The power supply of the device was also identified as a possible method to skip the PIN check [33]. Ledger also had decent vulnerabilities, not also the prominent supply chain attack where code could be injected into the MCU between the host and the secure element [34], but software

Existing Attacks	Hardware				Firmware				Software
Classification	Supply Chain	Side Channel	Fault	Data Remanence	Supply Chain	Lowlevel	Timing	Cryptography	Remote
Trezor One	Due to a chip issue, the [37] memory write-protection of the bootloader did not work. Therefore, an Attacker could modify the bootloader.	When [31] analyzing the power consumption with an oscilloscope, one could identify currently executed code. This could be used to extract the private key.	When [33] manipulating the voltage of a modified Trezor device, the pin could be skipped. However, this couldn't be done on an unmodified device.	The private keys [32] are kept in plain text in SRAM. The memory was not wiped, and with a chip freezer, an attacker could extract the keys.		A specially crafted transaction could [38] trigger a buffer overflow, with which an attacker was able to extract the private key (PIN and passphrase required). Specially crafted USB packets [39] could trigger overflows on old firmware, which could be used for remote code execution.	TOCTOU: A specially crafted [40] multisig transaction could contain a change output of an attacker, which wasn't confirmed by the user.		
Ledger Nano S					Due to compiler intrinsics, one can put exploit code in the unsafe Ledger chip [34] memory while still sending the secure element the original firmware ("MCU fooling"). With this, modified firmware stayed undetected for the genuity check.	Syscalls of the embedded OS BOLOS did not check the pointers in the [41] arguments correctly, which could result in dereferencing a nullpointer. An attacker could use this to dump part of memory.		Upgrading BOLOS [41] is done with a secure channel inspired by SCP-02. However, the MAC part was not implemented so an padding oracle attack was possible. The researcher was able to decode a few bytes of the datastream.	Transaction Javascript code of the Ledger Chrome [35] Application was writable without privileges (AppData folder). An Attacker could change e.g. the receiver address.
Keepkey	Due to a chip issue, the [42] memory write-protection of the bootloader did not work. Therefore, an Attacker could modify the bootloader.	Might have been [31] vulnerable to the side channel attack because it has the same architecture of the Trezor One (No PoC).	The authors of the Trezor One fault attack stated that the Keepkey [33] might also be vulnerable to a voltage fault.	Suspected by the author to be also vulerable to the [32] chip freezing attack, see Trezor One.		The device was vulnerable [36] to a format string attack. With special characters sent, an attacker could execute exploit code or retrieve sensitive data.			
Trezor T			Voltage fault attack might also succeed because of similar [33] architecture.	Suspected by the author to be also vulerable to the [32] chip freezing attack, see Trezor One.					
Ledger Blue					The author of the Ledger [34] Nano S MCU fooling stated that the Ledger Blue might also be vulnerable.				Because the Ledger Chrome [35] Application can be used with Ledger Blue, it was also vulnerable to the unprivileged Javascript code tampering.

Table 5.8: Overview of existing attacks on the five specified hardware wallets.

errors like the javascript file access exploit in the wallet [35]. Keepkey also had a C vulnerability with a format string [36].

5.5 Discussion

This security feature review did not only gave a good overview and introduction in the field of cryptocurrency hardware wallets, but also explained the companies unique approaches in protecting the user's private key. When reviewing the collected general information about the wallets, it is clearly visible that the different companies follow diverse approaches which is visible through all attestation layers. Table 5.1 shows that the Trezor and Keepkey are open platforms, whereas Ledger devices are closed hardware. Table 5.2 reinforces this impression, because the Ledger wallets are the only ones utilizing the concept of a secure element, which has to be closed-source by design.

This can also be seen by reviewing the safety measures and attestations further. To check that nothing has been manipulated, Trezor and Keepkey use hologram stickers at various points. Ledger however, state that they do not need any stickers or sealings for their device, because the software integrity check unleashing the secure element is more sufficient (see Table 5.3). This secure element approach can also be seen in Table 5.4, because Ledger devices do not use special tamperresistant casings. Table 5.5 shows that every device needs to verify the bootloader to stop malicious code loading. But also the firmware verification is critical, as seen in Table 5.6. The firmware also protects the private key access because all wallets can use a passphrase (which enables access to multiple wallets in case somebody forces the owner to reveal a password) and a PIN. Also, the transactions have to be confirmed on the device (see Table 5.7). All those attestations have been tested after unpacking and initializing the devices.

However, this study also showed that there are still flaws in the security design of hardware wallets. Table 5.8 shows that there are multiple attack vectors available, and a lot of attacks were already possible. Comparing the different attestations shows that a lot of effort was done to prevent product manipulation such as malicious code execution on the device and regulating access to the private key. Nonetheless, there are no attestation methods to verify that the hardware can be trusted, which leads us to the proof of concept in Section 6.

CHAPTER 6

Proof of Concept

Because the wallets do not verify the hardware, it was chosen to create a supply chain attack. After getting knowledge about the most prominent wallets, the decision to create a fake wallet for the Trezor Model One was made. The main reason was the openess of the platform, because there is no secure element and everything of the Trezor One is open source, including the wallet website and hardware.

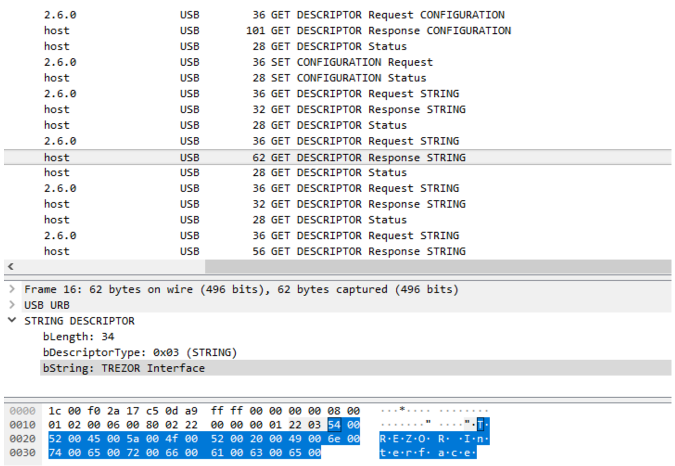
6.1 Configuring the USB Armory

The USB armory [5] is a single-board computer. The boot process involves a micro SD card, similar to the Raspberry Pi [43], and a Debian-like Linux can be flashed on the card. However, it supports, unlike a Raspberry Pi, no external peripherals, like a monitor or a network interface. After the USB-stick like device contains a micro SD card with an operating system and is connected to a host, it automatically boots and loads a Linux driver module called "g_ether".
This module is build on top of the Linux-USB stack and implements a Ethernet USB gadget. A USB gadget driver [44] gives the system the functionality of a USB device (and not a USB host), which is also a desired behaviour to emulate a Trezor USB device. On the host side, Windows PCs for example can use a Technology called RNDIS (Remote Network Driver Interface Specification) [45] to use this Ethernet gadget to emulate a network interface in order to communicate with the device.

6.2 Sniffing the USB Protocol

So with its initial configuration, we can connect with the Debian of the USB armory via ssh and the emulated network interface (over the g_ether module). But before it is possible to implement the Trezor protocol, it must first be examined further, as well as identify the USB endpoints provided by the Trezor. For this, the programs Wireshark

and UsbTreeView were used.
Wireshark [46] is a program to capture not only network but also USB traffic (via "extcap"). Different interactions with the original Trezor (like entering a PIN, doing a transaction, just connecting it to the host) were recorded and then analyzed. With Wireshark, it was not only possible to capture the Trezor-specific protocol, but also the sent descriptors (see Figure 6.1).



```
2.6.0           USB     36 GET DESCRIPTOR Request CONFIGURATION
host            USB    101 GET DESCRIPTOR Response CONFIGURATION
host            USB     28 GET DESCRIPTOR Status
2.6.0           USB     36 SET CONFIGURATION Request
host            USB     28 SET CONFIGURATION Status
2.6.0           USB     36 GET DESCRIPTOR Request STRING
host            USB     32 GET DESCRIPTOR Response STRING
host            USB     28 GET DESCRIPTOR Status
2.6.0           USB     36 GET DESCRIPTOR Request STRING
host            USB     62 GET DESCRIPTOR Response STRING
host            USB     28 GET DESCRIPTOR Status
2.6.0           USB     36 GET DESCRIPTOR Request STRING
host            USB     32 GET DESCRIPTOR Response STRING
host            USB     28 GET DESCRIPTOR Status
2.6.0           USB     36 GET DESCRIPTOR Request STRING
host            USB     56 GET DESCRIPTOR Response STRING

> Frame 16: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
> USB URB
v STRING DESCRIPTOR
    bLength: 34
    bDescriptorType: 0x03 (STRING)
    bString: TREZOR Interface

0000  1c 00 f0 2a 17 c5 0d a9  ff ff 00 00 00 00 08 00   ···*···· ········
0010  01 02 00 06 00 80 02 22  00 00 00 01 22 03 54 00   ·······" ····"·T·
0020  52 00 45 00 5a 00 4f 00  52 00 20 00 49 00 6e 00   R·E·Z·O· R· ·I·n·
0030  74 00 65 00 72 00 66 00  61 00 63 00 65 00         t·e·r·f· a·c·e·
```

Figure 6.1: Wireshark shows all the USB communication, like this USB string descriptor describing one interface as the "TREZOR Interface".

UsbTreeView [47], on the other hand, was used to identify all static information about the Trezor, like the magic numbers and interfaces. The program can print a comprehensive report about a USB device. Also, the required endpoints which were already visible in the Wireshark capture could be verified. They are illustrated as a UML communication diagram in Figure 6.2. Endpoint 0 is used to send the descriptors, while the U2FInterface enables two-factor authentication (this interface is not needed in this work).

The descriptor information is required by the Trezor Bridge to filter for wallets connected to the host. This bridge, which is a userspace driver, is used by the Trezor web wallet (over a web server from the Trezor Bridge) to access the hardware wallet. The requirements for a USB device to look like a Trezor could also be reviewed directly in the source code of the Trezor Bridge, which is written in the programming language Go [48]. This communication is shown clearly within Figure 6.3.

Regarding the protocol, when plugging in the Trezor, all of the USB descriptors are sent over endpoint 0. Because the software seems to be polling the Trezor every 0.5s, this "keepalive" sequence in Figure 6.4 is going on over endpoint 0. In fact, those messages are just USB descriptor requests and this behaviour is implemented automatically by the Linux USB stack. However, at the same time, the Trezor protocol is sent with HID over the TREZOR Interface endpoint (see Figure 6.3). When the Trezor web wallet is started and the Trezor bridge sees a "valid" Trezor connected, a preamble is sent first, as seen

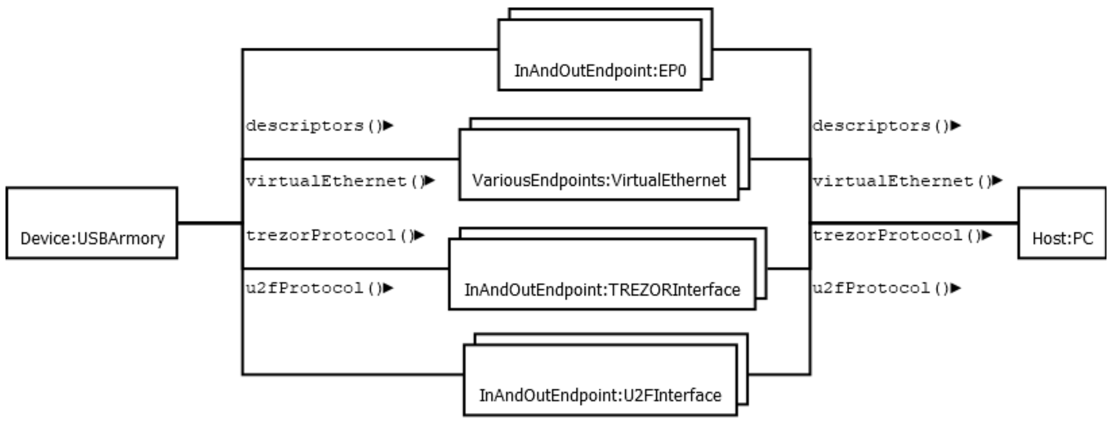


Figure 6.2: An overview of the needed USB endpoints which are behind the virtual channels.

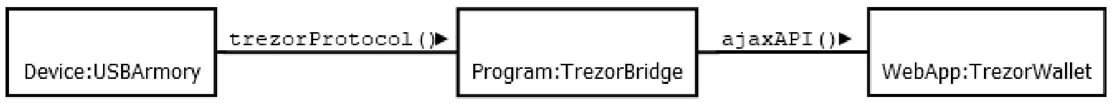


Figure 6.3: On the host side, these two important software components are interacting.

in Figure 6.5. After this, the PIN is sent (see Figure 6.6) and after that, all the other messages like the public key or a transaction signing request.

6.3 Implementation of the Fake Wallet

After we made those diagrams and understood the collected data, the investigation on how to implement the protocol could begin. The first thing to know is that the g_ether module is a legacy module and does not support multiple different functions. In Section 4.2 it was mentioned that it might be useful to simultaneously run a virtual network interface and the Trezor emulation, but those are two different USB functions. Now the question is, how this can be etablished. The g_ether module must be unloaded and another module which supports multiple functions is needed. But in fact, we do not have to write an own kernel module. The Linux USB gadget framework grew historically and in late 2003, it introduced GadgetFS, which enabled the development of userspace gadget drivers [44]. In 2010, FunctionFS was added, which basically resembles a rewrite of GadgetFS to support the combination of multiple userspace gadget functions into a single composite gadget. Finally, with Linux 3.11, ConfigFS was created, which allowed an easy interface based of file commands to compose such a composite gadget at runtime [49]. E.g. new functions can be created with ConfigFS by just creating a file with the correct name in the file system.
So, the choice was to combine several functions with ConfigFS. For this, the kernel module libcomposite had to be loaded. A shell script (see Listing 6.1) was created to initialize

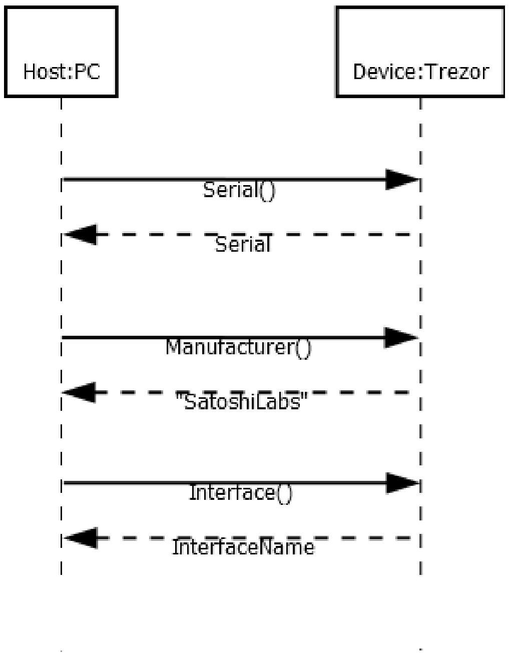


Figure 6.4: The "keepalive" messages.

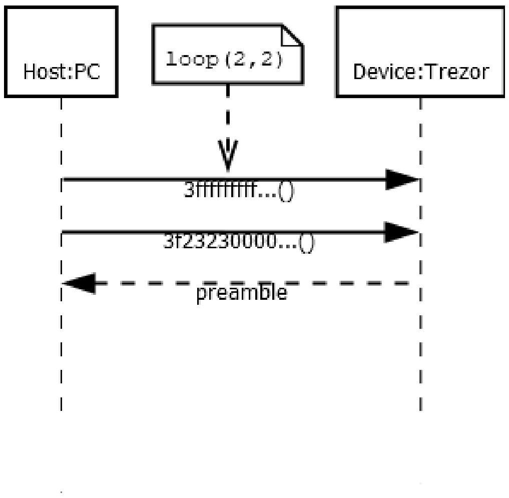


Figure 6.5: The "preamble" messages over the TREZOR Interface (two times).

the USB gadget and all the functions via ConfigFS. There was also the possibility to use libusbg which is a C library for ConfigFS, but because it did not make the configuration easier, it was not used. With ConfigFS, the composite gadget was customized to send the descriptors extracted by UsbTreeView, like setting the USB gadget's manufacturer to the string "SatoshiLabs" and the product string to "TREZOR". Now, the composable ethernet function could be added [50]. However, for the Trezor protocol, it was not

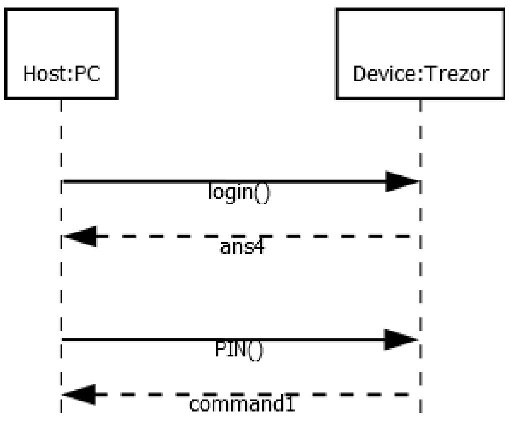


Figure 6.6: The PIN messages over the TREZOR Interface after the preamble. Note that the PIN is sent in plaintext.

necessary to create a own function with FunctionFS, because after further analysis of the Wireshark capture, it was clearly evident that the Trezor-specific protocol was build on top of the USB HID protocol.

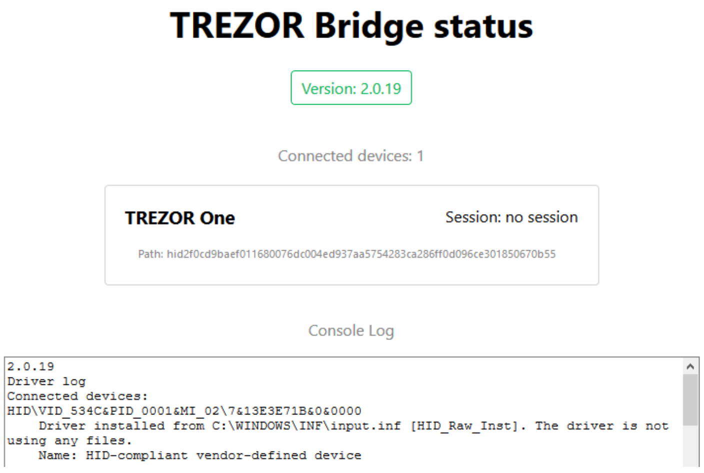
Listing 6.1: Shell script to configure the USB emulation.

```
#!/bin/bash -e
{
echo "starting␣usb␣emulation"
# remove single ether module
modprobe -r g_ether
sleep 1
# insert gadgetfs and functionfs module
modprobe libcomposite
#modprobe g_ffs functions=rndis
modprobe usb_f_fs
modprobe usb_f_hid
# create new gadget
cd /sys/kernel/config/usb_gadget/
mkdir g && cd g
# vid, pid
echo 0x534c > idVendor  # SatoshiLabs
echo 0x0001 > idProduct #
echo 0x0100 > bcdDevice # v1.0.0
echo 0x0200 > bcdUSB    # USB 2.0
```

```
# multifunction gadget
echo 0x00 > bDeviceClass
echo 0x00 > bDeviceSubClass
echo 0x00 > bDeviceProtocol
# english strings
mkdir -p strings/0x409
echo "BA386EE5B17B4DCEDF8BB89B" > strings/0x409/serialnumber
echo "Satoshi␣Labs"          > strings/0x409/manufacturer
echo "TREZOR"    > strings/0x409/product
#- general config
mkdir -p configs/c.1
echo 250 > configs/c.1/MaxPower
#-- hid
#mkdir -p functions/ffs.my_func_name
#mkdir -p /tmp/mount_point
mkdir -p functions/rndis.usb2  # network
mkdir functions/hid.usb0
echo 0 > functions/hid.usb0/protocol
echo 0 > functions/hid.usb0/subclass
echo 8 > functions/hid.usb0/report_length
cat /home/USB armory/r1.hex > functions/hid.usb0/report_desc
mkdir functions/hid.usb1
echo 0 > functions/hid.usb1/protocol
echo 0 > functions/hid.usb1/subclass
echo 8 > functions/hid.usb1/report_length
cat /home/USB armory/r2.hex > functions/hid.usb1/report_desc
#mount my_func_name -t functionfs /tmp/mount_point
#/home/USB armory/trezor /tmp/mount_point &
#ln -s functions/ffs.my_func_name configs/c.1/
ln -s functions/rndis.usb2 configs/c.1/
ln -s functions/hid.usb0 configs/c.1
ln -s functions/hid.usb1 configs/c.1
#--
# OS descriptors
echo 1       > os_desc/use
echo 0xcd    > os_desc/b_vendor_code
echo MSFT100 > os_desc/qw_sign
echo RNDIS   > \
functions/rndis.usb2/os_desc/interface.rndis/compatible_id
echo 5162001 > \
functions/rndis.usb2/os_desc/interface.rndis/sub_compatible_id
ln -s configs/c.1 os_desc
#echo "before"
```

```
#ls /dev | grep hid
udevadm settle -t 5 || :
ls /sys/class/udc/ > UDC
echo "ready"
ls /dev | grep hid
# compile the message loop
#gcc /home/USB armory/trezor.c -o /home/USB armory/trezor
#echo "starting trezor"
#/home/USB armory/trezor
#echo "fin"
} > log.txt
```

HID can be added just as another function and offers a easy interface in form of a device driver file hidgX [51] which can be read and written after initialization. But before the program which used the HID protocol via the device file could be implemented, the USB HID report descriptor had to be configured. As mentioned in Section 2.4.3, HID report descriptors contain specific information how the HID device sends its data, especially in which format. To retrieve the Trezor One report descriptors, Usbhid-dump [52] was used. The obtained binary descriptors could also be configured via ConfigFS. With this, the Trezor bridge finally accepted the USB armory as a Trezor (see Figure 6.7), but the Trezor web wallet did not see any wallet at all.


TREZOR Bridge status

Version: 2.0.19

Connected devices: 1

TREZOR One   Session: no session

Path: hid2f0cd9baef011680076dc004ed937aa5754283ca286ff0d096ce301850670b55

Console Log

```
2.0.19
Driver log
Connected devices:
HID\VID_534C&PID_0001&MI_02\7&13E3E71B&0&0000
    Driver installed from C:\WINDOWS\INF\input.inf [HID_Raw_Inst]. The driver is not
using any files.
    Name: HID-compliant vendor-defined device
```

Figure 6.7: The Trezor Bridge believing a Trezor is connected.

This was because the web wallet makes use of the bridge and therefore of the protocol build on HID, which was not implemented at this point. So the last task was to implement the Trezor-specific protocol over HID. The successful setup of the HID kernel module in the shell script created the hidg0 device file. Now, a C program to implement the Trezor protocol could be developed. This program can connect to the hidg0 file and interact with it via the file commands of the standard library. With the Wireshark capture,

identifiying the basic Trezor messages was possible.
Because all requests are issued by the web wallet via the USB host, the C program developed is basically just a message dispatcher (see Listing 6.2), which processes all retreived messages in a loop and then sends the answer. The preamble and PIN messages were the first part of the protocol which was implemented.

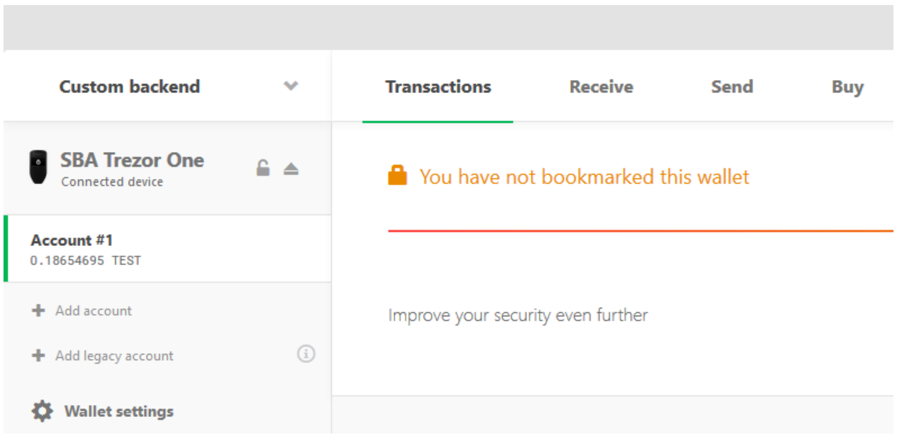
Listing 6.2: C code to process the HID messages.

```
while(1) {
        read_msg(buffer, fd);
        switch(identify_packet(buffer)) {
                case IDENTIFY_ME:
                printf("GOT IDENTIFY ME\n");
                handle_identify_me(buffer, fd);
                break;
                case TESTNET:
                printf("GOT TESTNET\n");
                handle_testnet(buffer, fd);
                break;
                case TESTNET_RECV:
                printf("GOT TESTNET RECV\n");
                handle_testnet_recv(buffer, fd);
                break;
                case OK_TESTNET:
                printf("GOT  OK  TESTNET\n");
                handle_ok_testnet(buffer, fd);
                break;
                case OK_SEND:
                printf("GOT  OK  SEND\n");
                handle_ok_send(buffer, fd);
                break;
                case OK_SEND_ADDR:
                printf("GOT  OK  SEND ADDR\n");
                handle_ok_send_addr(buffer, fd);
                handle_ok_send_confirm(buffer, fd);
                break;
                case COMMAND1:
                printf("GOT COMMAND1\n");
                handle_command1(buffer, fd);
                break;
                // .. other messages
                case COMMAND8:
                printf("GOT COMMAND8\n");
                handle_command8(buffer, fd);
                break;
```

```
                    case SEND:
                    printf("GOT␣SEND\n");
                    handle_send(buffer, fd);
                    break;
                    case NO_ANS:
                    printf("GOT␣NO␣ANS\n");
                    printf("EXPECT␣IDENTIFY␣ME␣NEXT\n");
                    break;

                    default:
                    printf("CANNOT␣IDENTIFY␣COMMAND\n");
                    print_buffer(buffer, REPORT_SIZE);
                    // triggers error msg with expected oxpub
                    write_msg(command1_1, fd);
                    write_msg(command1_2, fd);
                    write_msg(command1_3, fd);
                    write_msg(command1_4, fd);
                    break;
          }
}
```

After this, the Trezor software fully recognized the device as a Trezor (see Figure 6.8). Finally, the messages to get the account and public key were integrated. Also the initialization behaviour could be emulated, so the victim thinks that his new wallet was not preinitialized.



You have not bookmarked this wallet

Improve your security even further

Figure 6.8: The Trezor Wallet showing the account which the USB armory emulates.

6.4 Final Attack Vector

The attack with this fake wallet could happen with a supply chain attack. The victim buys the wallet by a reseller, thinking he bought the original one. It is necessary that the victim generates a new wallet, because entering a recovery seed is not addressed in this thesis. If the victim trusts the hardware, let the wallet generate a new e.g. Bitcoin address and wants to make a transaction, he will notice that (like with an original wallet) he first has to "fill" the wallet with money, because its initial balance is zero. Now if the victim wants to pay money on his "wallet", the fake wallet sends the malicious address to the web wallet and in the end, the attacker gets the money. Especially with Bitcoin, transactions are final when properly propagated.

6.5 Discussion

On the one hand, it was shown again that USB is a problematic interface, because it is known for the lack of security checks. Attacks like using an USB stick to emulate a keyboard and type critical commands are particulary easy to perform. The descriptors of the Trezor could be extracted without any problems, and faking those descriptors was enough to let the Trezor Bridge believe a real Trezor was connected. The software did not care about the virtual network interface on the device or the fact that not all string descriptors fully matched the original Trezor.

On the other hand, this proof of concept could demonstrate that the software trusts the hardware too much. The Trezor Wallet only needed a working protocol response, and the Trezor messages were not encrypted and could just be replayed by sending the binary capture of Wireshark. All for this proof of concept necessary Trezor messages could be fully implemented and if all the other messages like device name configuration etc. are emulated as well (which is technically possible), a victim is not able to distinguish the behaviour of the software between the emulation and a real Trezor.

CHAPTER 7

Future Work

The security feature market review's main purpose was to gave a good entry point in cryptocurrency hardware wallet security. As new wallets are coming to the market and new attestation methods are developed, one has to observe the future development trends. Especially the vulnerability classifcation is a snapshot in time, as new vulnerabilities are found on a regular basis, and the classification has to be updated with those. But it can be used to classify further attacks with this scheme.

The specific proof of concept introduced in this thesis could be improved in multiple ways. Firstly, it is currently needed to open a remote shell and start the script in order to start the Trezor One emulation due to debugging purposes. In a real attack, the script should be started after boot. Secondly, the transactions were only tested with the Bitcoin Testnet, so the software has to be adapted to the real Bitcoin network. The RNDIS interface must also be removed. Finally, in order to seem like a Trezor, one must change the appearance of the USB device to look like to original hardware, including the characteristic LCD display. However, this might not be a lot of work for a serious attacker. In the end, there is a good chance that multiple other wallets are also vulnerable to this mimicking attack.

CHAPTER 8

Conclusion

The first part with the security feature market review gave an overview about five prominent hardware wallets, their characteristics and their weaknesses. It is important to differentiate how the different vendors follow particularly different security approaches, all with their personal subtleties. Especially the difference openess vs. secure element usage stood out. There are multiple attestations available, however, they do not cover every situation, e.g. faking the hardware. It also showed existing attacks, which were classified with a layered approach. This gave a good understanding of important attack vectors which must be considered with hardware wallets.

The second part which presented the fake wallet attack showed that there are no attestations for the Trezor One to verify that the hardware is genuine. Therefore, it can be used as an attack vector. It was possible to emulate the device from the USB descriptors to the Trezor messages and create an attack scenario. It showed how this emulation was possible, from choosing the hardware to reverse engineering the messages. This proved that such an attack can be created with managable time and resources, and there are still missing attestations for devices like the Trezor One.

Bibliography

[1] N. Satoshi, “Bitcoin: A Peer-to-Peer Electronic Cash System.” `https://bitcoin.org/bitcoin.pdf`, 2008. Accessed: 2018-09-21.

[2] B. Marr, “A Short History Of Bitcoin And Crypto Currency Everyone Should Read.” `https://www.forbes.com/sites/bernardmarr/2017/12/06/a-short-history-of-bitcoin-and-crypto-currency-everyone-should-read/#b794c773f279`, 2007. Accessed: 2018-09-21.

[3] N. Marinoff, “Cryptocurrency Growing in Popularity: Especially Among Young People.” `https://blockonomi.com/cryptocurrency-growth-young-people/`, 2018. Accessed: 2019-02-18.

[4] Ledger, “Hardware wallets - Securing your crypto assets.” `https://www.ledger.com/`, 2018. Accessed: 2018-09-21.

[5] I. Path, “Open Source Flash-Drive Sized Computer.” `https://inversepath.com/usbarmory.html`, 2018. Accessed: 2018-09-21.

[6] Stellabelle, “Cold Wallet Vs. Hot Wallet: What’s The Difference?.” `https://medium.com/@stellabelle/cold-wallet-vs-hot-wallet-whats-the-difference-a00d872aa6b1`, 2017. Accessed: 2019-01-30.

[7] Strongcoin.com, “Highly Secure Bitcoin Wallet - Strongcoin.” `https://strongcoin.com/`, 2019. Accessed: 2019-01-30.

[8] S. Labs, “Pre-order your new TREZOR model T!.” `https://blog.trezor.io/pre-order-your-new-trezor-model-t-cf2a3426cf03`, 2019. Accessed: 2019-01-30.

[9] I. Fovino, “Secure Smart Embedded Devices, Platforms and Applications,” 2014. ISBN: 978-1-4614-7915-4.

[10] commoncriteriaportal.org, “The Common Criteria.” `https://www.commoncriteriaportal.org/`, 2018. Accessed: 2019-02-22.

[11] T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "Things, trouble, trust: on building trust in iot systems," in *Proceedings of the 53rd Annual Design Automation Conference*, p. 121, ACM, 2016.

[12] K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "Hydra: Hybrid design for remote attestation (using a formally verified microkernel)," in *Proceedings of the 10th ACM Conference on Security and Privacy in wireless and Mobile Networks*, pp. 99–110, ACM, 2017.

[13] M. Palatinus, "Mnemonic code for generating deterministic keys." `https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki`, 2013. Accessed: 2019-02-16.

[14] C. Peacock, "USB in a NutShell." `https://www.beyondlogic.org/usbnutshell/usb1.shtml`, 2018. Accessed: 2019-01-26.

[15] T. Roth, D. Nedospasov and J. Datko, "wallet.fail: Hacking the most popular cryptocurrency hardware wallets ." `https://media.ccc.de/v/35c3-9563-wallet_fail#t=113`, 2018. Accessed: 2019-02-28.

[16] A. Gkaniatsou, M. Arapinis, and A. Kiayias, "Low-level attacks in bitcoin wallets," in *International Conference on Information Security*, pp. 233–253, Springer, 2017.

[17] J. Datko, C. Quartier, and K. Belyayev, "Breaking bitcoin hardware wallets," *DEF CON 2017*, 2017. Talk: `https://www.youtube.com/watch?v=hAtoRrxFBWs`.

[18] C. Decker and R. Wattenhofer, "Bitcoin transaction malleability and mtgox," in *European Symposium on Research in Computer Security*, pp. 313–326, Springer, 2014.

[19] M. Rosenfeld, "Analysis of hashrate-based double spending," *arXiv preprint arXiv:1402.2009*, 2014.

[20] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts.," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1007, 2016.

[21] M. Dalton, H. Kannan, and C. Kozyrakis, "Deconstructing hardware architectures for security," in *5th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) at ISCA*, Citeseer, 2006.

[22] M. Roland, J. Langer, and J. Scharinger, "Practical attack scenarios on secure element-enabled mobile devices," in *2012 4th International Workshop on Near Field Communication*, pp. 19–24, IEEE, 2012.

[23] allthingsdecentral, "Wallet Overview." `https://allthingsdecentral.com/pages/compare-hardware-wallets`, 2018. Accessed: 2018-10-14.

[24] K. Kreder, “Hardware Wallet Vulnerabilities.” https://blog.gridplus.io/hardware-wallet-vulnerabilities-f20688361b88, 2018. Accessed: 2018-10-14.

[25] SatoshiLabs, “Trezor Security.” https://trezor.io/security/, 2018. Accessed: 2018-10-13.

[26] Keepkey, “Keepkey Security FAQ.” https://www.keepkey.com/keepkey/faq/security/, 2018. Accessed: 2018-10-14.

[27] Keepkey, “KeepKey under the hood.” https://medium.com/@AussieHash/keepkey-under-the-hood-3beac31e1064, 2018. Accessed: 2018-10-17.

[28] Ledger, “How to protect hardware wallets against tampering.” https://www.ledger.fr/2015/03/27/how-to-protect-hardware-wallets-against-tampering/, 2015. Accessed: 2018-10-14.

[29] Ledger, “Check hardware integrity.” https://support.ledger.com/hc/en-us/articles/115005321449-Check-hardware-integrity, 2019. Accessed: 2019-02-28.

[30] SatoshiLabs, “User manual:Running a local instance of Trezor Wallet.” https://wiki.trezor.io/User_manual:Running_a_local_instance_of_Trezor_Wallet, 2019. Accessed: 2019-02-26.

[31] J. Hoenicke, “Trezor Power Analysis.” https://jochen-hoenicke.de/trezor-power-analysis/, 2018. Accessed: 2018-10-14.

[32] D. Zero404Cool, “Frozen Trezor - Data Remanence Attacks!.” https://medium.com/@Zero404Cool/frozen-trezor-data-remanence-attacks-de4d70c9ee8c, 2017. Accessed: 2018-10-14.

[33] U. Soul, “How To Hack A Trezor or KeepKey Hardware Bitcoin Wallet - Plus Make Your Own Open Sourced Hardware Wallet!.” https://steemit.com/technology/@ura-soul/how-to-hack-a-trezor-or-keepkey-hardware-bitcoin-wallet-plus-make-your-own-open-sourced-hardware-wallet, 2018. Accessed: 2018-10-14.

[34] S. Rashid, “Breaking the Ledger Security Model.” https://saleemrashid.com/2018/03/20/breaking-ledger-security-model/, 2018. Accessed: 2018-10-14.

[35] Anonymous, “Ledger Receive Address Attack.” https://www.docdroid.net/Jug5LX3/ledger-receive-address-attack.pdf, 2018. Accessed: 2018-10-14.

[36] Keepkey, “Security Updates & Responsible Disclosure.” https://www.keepkey.com/2018/03/09/security-updates-responsible-disclosure/, 2018. Accessed: 2018-10-16.

[37] SatoshiLabs, “TREZOR One: Firmware Update 1.6.1.” https://blog.trezor.io/trezor-one-firmware-update-1-6-1-eecd0534ab95, 2018. Accessed: 2019-02-27.

[38] SatoshiLabs, “Malicious ScriptSig in transaction.” https://trezor.io/security/, 2014. Accessed: 2019-02-27.

[39] SatoshiLabs, “Details about the security updates in Trezor One firmware 1.6.2.” https://blog.trezor.io/details-about-the-security-updates-in-trezor-one-firmware-1-6-2-a3b25b668e98, 2018. Accessed: 2019-02-27.

[40] SatoshiLabs, “SpendMultisig malicious change in transaction.” https://trezor.io/security/, 2015. Accessed: 2019-02-27.

[41] Ledger, “Firmware 1.4: deep dive into three vulnerabilities which have been fixed.” https://www.ledger.fr/2018/03/20/firmware-1-4-deep-dive-security-fixes/, 2018. Accessed: 2019-02-27.

[42] Keepkey, “https://info.shapeshift.io/blog/2018/03/21/security-update-release-notes-for-v5-1-0/.” https://blog.trezor.io/trezor-one-firmware-update-1-6-1-eecd0534ab95, 2018. Accessed: 2019-02-27.

[43] R. P. Foundation, “Raspberry Pi.” https://www.raspberrypi.org/, 2019. Accessed: 2019-02-12.

[44] T. kernel development community, “USB Gadget API for Linux.” https://www.kernel.org/doc/html/v4.16/driver-api/usb/gadget.html, 2019. Accessed: 2019-02-12.

[45] Microsoft, “Overview of Remote NDIS (RNDIS).” https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-remote-ndis--rndis-, 2019. Accessed: 2019-02-12.

[46] W. Foundation, “About Wireshark.” https://www.wireshark.org/, 2019. Accessed: 2019-02-12.

[47] U. Sieber, “USB Device Tree Viewer V3.3.2.” https://www.uwe-sieber.de/usbtreeview_e.html, 2019. Accessed: 2019-02-12.

[48] SatoshiLabs, “TREZOR Communication Daemon (written in Go) .” https://github.com/trezor/trezord-go, 2019. Accessed: 2019-02-12.

[49] K. Opasiak, “Tame the USB gadgetstalkative beast.” https://elinux.org/images/1/14/Opasiak--tame_the_usb_gadgets_talkative_beast.pdf, 2014. Accessed: 2019-02-12.

[50] A. Pietrasiewicz, “USB/Linux USB Layers/Configfs Composite Gadget/Usage eq. to g ether.ko.” https://wiki.tizen.org/USB/Linux_USB_Layers/Configfs_Composite_Gadget/Usage_eq._to_g_ether.ko, 2016. Accessed: 2019-02-12.

[51] kernel.org, “Linux USB HID gadget driver.” https://www.kernel.org/doc/Documentation/usb/gadget_hid.txt, 2017. Accessed: 2019-02-12.

[52] DIGImend, “USB HID device dumping utility .” https://github.com/DIGImend/usbhid-dump, 2019. Accessed: 2019-02-12.