# Analysis of Low-level iOS Lightning Protocols

Master Thesis proposal

Andreas Karner
andreas.karner@student.tugraz.at
Graz University of Technology, Austria

***Abstract*—The importance of mobile devices has increased over the past few years tremendously. This goes hand in hand with an increased amount of sensitive data stored on these devices, which makes leakages undesirable and data protection essential. One way to break this data protection would be Juice-Jacking, which aims to steal sensitive data from mobile devices after establishing a tethered connection, for example to charge the device battery.**

**In this thesis, we will examine how we can leverage Juice-Jacking, to leak sensitive data from Apple devices. This is accomplished by first, reimplement the proprietary IDBUS protocol on a microcontroller, which is necessary to make the USB interface available on the Lightning plug. Second, we will search for new Juice-Jacking attack vectors in the IDBUS protocol itself, by performing black-box fuzzing against the embedded chip called Tristar, part of the Apple device. Third, we assemble an evil Lightning cable, which mounts a mixture of Human Interface Device (HID) and Juice-Filming attack against an Apple device. Finally, we propose a Lightning Data Blocker, which allows mitigating all currently known kind of Juice-Jacking attacks.**

***Index Terms*—Apple, Lightning, IDBUS, Reverse Engineering, Juice-Jacking**

## I. Motivation

For most people nowadays, the smartphone is already part of their daily life, because it simplifies performing overly complex tasks, such as proving user identity and managing their financial banking account. Consequently, to enable all these highly confidential tasks, these devices store a significant amount of sensitive data, which needs to be protected from outside intruders, by encasing it with a secure system.

However, all mobile devices have one special property in common, they are battery powered, therefore recharging the battery after a certain amount of time is necessary. Charging these devices is almost entirely accomplished tethered, and to keep the device charged on the go, so called public-kiosks are deployed at airports, bus station and shopping centers, which allow the user to connect their mobile devices for charging. However, these kiosks are prone to so called charging attacks, which exploit the unawareness of the users and can be leveraged by an attacker to steal sensitive data or install hidden malware on the device. The academic research community denotes these charging attacks as Juice-Jacking, which unites various subclasses, where each subclass exploits a different attack vector. Three examples are, the classic Juice-Jacking [4], Juice-Filming [3] and injecting HID strokes [8], where each of them take advantage of the fact that the charging cables do not only convey power, but also provides access to the Universal-Serial-Bus (USB) interface. In conjunction with the USB On-The-Go (OTG) specification, an attacker is now able to decide if the mobile phone should run in host or peripheral mode, where each mode comes with different security concerns by design. In the case of Juice-Jacking, an attacker emulates the USB host device, which forces the mobile device to act as a USB peripheral. This implies that commonly USB interfaces, such as access to the internal device storage, are exposed to the USB host. To actually read storage content, access needs to be activated and granted explicitly. Therefore, this attack can be mitigated by simply denying the access request, however, this is often not the case, due to unawareness of the user and to enhance the device usability. To overcome this access boundary, different Juice-Jacking classes have emerged, such as Juice-Filming and injecting HID strokes, which do not require any user authority. For them, the attacking object acts as a USB peripheral and exploits the default trust from the host device into the peripheral. In the case of Juice-Filming, a USB peripheral device is connected to the mobile device, aiming to video-capture the victim device screen. This attack does not provide direct access to the sensitive data, nevertheless, it obtains the entire screen content during the charging phase and therefore, usernames and passwords can be extracted via state-of-the-art optical character recognition (OCR) algorithm. Conversely, HID attacks take advantage of emulating input USB peripherals, such as keyboards and mice, which allow controlling the device by injecting user interactions.

Juice-Jacking attacks have been already addressed by various security researchers in the past and the following mitigation strategies have been proposed: i) removing the data lines, so called USB-Condoms [4], ii) leverage detection tool such as SandUSB [8] iii) shutting down the device during charging iv) use the own charger. However, all these mitigation strategies assume a malicious outlet, and none of them, except shutting down the device, would be able to detect and mitigate Juice-Jacking attacks, where the malicious controller is embedded into the charging cable.

To close this gap, we will examine in this master thesis, how to mitigate Juice-Jacking attacks, where the malicious controller is embedded into the charging cable, with respect to Apple devices. Unfortunately, for Apple devices, it is not a valid mitigation strategy to simply remove the data lines, as we would do for USB, because they utilize a Apple specific connector called Lightning [12]. Lightning is an adaptive eight pin connector, which requires an initial messages preamble to synchronous the male and female port properly. Because this preamble is also mandatory for charging only cables, removing the data lines is not a legitimate option, as already stated. The female port, which is part of the Apple device, is wired single-sided and the male port double-sided. Furthermore, it is possible to map device protocols like USB, Universal asynchronous receiver-transmitter (UART) and Serial Wire Debug (SWD) to specific pins of the Lightning plug. Therefore, to detect the rotation of the male port and to commit to a pin wiring, the female and male port are equipped with an integrated circuit (IC) chip, called Tristar and Hi-Five [10] respectively. After connecting a Lightning cable to an Apple device, these two chips start communicating via the proprietary IDBUS protocol, which was classified by various security research, to be like a one-wire-protocol [7] on the physical layer and which implements a request-response pattern [1], [10] for exchanging information.

The master thesis is structured as follows. First, we empower a microcontroller to understand and interact with the IDBUS protocol, which is then leveraged to perform black box fuzzing [6] against the Tristar chip to search for new Juice-Jacking possibilities. Second, a malicious charging cable is manufactured, which mounts a mixture of Human Interface Device (HID) and Juice-Filming attack against a real iPhone device, where we aim to emulate a keyboard and a mouse as well as video-capture the screen. This is accomplished, by letting the evil cable acting as a USB OTG peripheral, which is trusted and mounted by the Apple device on default. To attack the Apple device, the cable will expose for each afore mentioned target a separate USB interface. Finally, we propose the Lightning Data Blocker, which allows mitigating all currently known kind of Juice-Jacking attacks, by reassembling the idea of a USB condom, and recover trust into the charging cable.

## II. Related Work

Numerous security researchers have already reverse engineered the Lightning connector as well as the physical layer of the embedded IDBUS protocol.

Amin [1] has successfully reverse engineered the pinout of the Lightning connector, as well as the IDBUS communication between the Tristar and Apple MFI (Hi-Five) chip. After gaining this information, he was able to access the internal Debug-UART interface and exposed with it the boot sequence of an iPhone device. His work was based on leaked documents about the Lightning connector and the Tristar chip; an already available Lightning-Serial-Cable from Redpark[1]; and brute forcing a desoldered Tristar chip. He has also shown that the Lightning connector consists of eight pins, where the male plug is not wired in the same way on both sides. In addition, his work has proven that the IDBUS protocol resembles a proprietary one-wire-protocol from Texas Instruments called SDQ, which can be captured and analyzed via logic analyzers. Additionally, he has identified that the Tristar chip communicates with other device components via the I2C protocol. Summarized, his work supplies fundamental knowledge about the Lightning connector pinout as well as how the physical layer of the IDBUS protocol works, which is needed to carry out our proposed goals. In contrast to his work, we aim to leverage the USB instead of the Debug-UART interface, for attacking the device.

Nyan Satan [10] has also elaborate vastly the same information as Amin, however, he has focused more on analyzing the IDBUS protocol and the associated messages and timing constrains. Furthermore, he has partially reverse engineered the Lightning Digital AV Adapter, which is also called Haywire [9]. His research concluded that the adapter is shipped with a very constrained firmware version, which aims to expose a simple USB peripheral interface. On connection, the operational firmware is then pushed from the Apple device to the peripheral, which includes all the missing logic to enable full video-capturing functionality of the device screen. To summarize, he has not directly attacked or exposed any sensitive data from the Apple device, instead, he supplied vital information on how data is conveyed in the IDBUS protocol. His work also aids us in carrying out our goals, because he has already shown how to interpret the various IDBUS messages.

Stacksmashing [11] proposed the Tamarin Cable, which is a low-cost Lightning debug adapter. After connecting it to an Apple device and a host machine, it allows a security researcher to either reset the device, boot into Device Firmware Update (DFU) mode or access internal device protocols, such as UART or JTAG. Requesting the desired protocol and resetting the Apple device is accomplished by replaying prior captured IDBUS messages. This is realized by reimplementing the IDBUS protocol on a Raspberry Pi Pico, leveraging its programmable I/O interface. In case an internal protocol is requested, the Tamarin Cable acts as gateway between the Apple device and the host machine after the request phase. By using the JTAG functionality of the Tamarin Cable, an attacker is capable of extracting sensitive data from the Apple device. However, to access the JTAG interface, the Apple device needs to be demoted, which requires a jailbreak beforehand. Compared to his work, we aim to explore new IDBUS messages by fuzzing and no jailbroken device is needed to mount our attack.

Billy Lau et al. [5] has shown in 2014 that it is possible, via a Juice-Jacking attack, to upload malware to a connected Apple Device. The present version of iOS (iOS 6) in 2014 facilitated that they can install their hidden malware without user interactions, capable of emulating touches and taking screenshots. Their malicious charger, denoted as Mactans, was able to compromise an Apple device, because each USB host was accepted as trustful without any approval. However, this has changed since iOS 7 [13], as already mentioned in Section 1. Compared to Mactans, where the Apple device acts as the USB peripheral, our work aims to extract sensitive information by letting the device act as USB host and therefore, bypass this newly introduced trust approval mechanism.

Furthermore, companies like Lambda-Concepts and Hak5 also sell off-the-shelf Lightning-to-USB charging cables with

[1]https://redpark.com/lightning-serial-cable-l2-db9v/

an evil implant, allowing it to infiltrate mobile devices. These cables are called Graywire[2] and O.M.G. cable[3] respectively. In the case of Graywire [2], which is tailored for Apple devices, the embedded microcontroller acquires USB access via the IDBUS and exposes a USB interface for video, keyboard, and mouse. Conversely, the O.M.G. cable is available in various variations and its main functionality is to read and inject keystrokes, but it does not support video-capturing of the device screen. Therefore, Graywire would be the ultimate fit for our needs, although it is not currently available for sale, therefore we will develop it on our own.

## III. Goal

The master thesis contains the following goals: i) explore the IDBUS protocol, used by the Lightning connector, for new weaknesses which allow mounting Juice-Jacking attacks, ii) demonstrate a mixture of HID and Juice-Filming attack against a iPhone 7, by embedding the attack logic into the Lightning cable iii) propose a Lightning Data Blocker, which allows mitigating all current known Juice-Jacking attacks related to Apple Lightning devices. To accomplish these goals, the first subgoal is to equip a microcontroller with a reimplementation of the IDBUS protocol, which allows us to emulate the Hi-Five chip and communicate with Tristar.

### A. *Black Box Fuzzing*

The ability to speak the IDBUS protocol enables us to perform black box fuzzing against the Tristar chip. This allows us to fuzz for data disclosure and system crashes in general, as well as searching for new possibilities to mount Juice-Jacking attacks within the protocol. Each of them is counted as a security issue because it either allows an attacker to obtain directly sensitive information, crash the device on demand or extract sensitive information stealthily.

### B. *Attack Phase*

The next subgoal encompasses the design and manufacturing of an evil Lightning cable, which allows mounting a mixture of Human Interface Device (HID) and Juice-Filming attack against an Apple mobile device, such as the iPhone 7. Specifically, the modified cable should be capable of video-capturing the device screen as well as injecting keyboard strokes and mouse movements by emulating an HDMI, keyboard, and mouse USB accessory, respectively. To overcome the stated mitigations in Section 1, the attack logic will be embedded into the cable itself.

### C. *Mitigation Phase*

Finally, the thesis provides a Lightning Data Blocker, which allows mitigating all known Juice-Jacking attacks, including the previously assembled evil Lightning cable. The Lightning Data Blocker will automatically mitigate all Juice-Jacking attacks, by acting as a firewall, and dropping all non-charging related IDBUS messages.

[2]https://blog.lambdaconcept.com/post/2019-09/graywire/

[3]https://shop.hak5.org/products/omg-cable

## IV. Experimental Approach

This section will exhibit and explain our experimental approach, and how we aim to accomplish the nominated goals in Section 3. To enable all these goals, the ability to intercept IDBUS messages between Apple devices and Lightning accessories, as well as injecting messages as needed, is necessary. This is achieved by first acquiring fundamental knowledge about the protocol and later endowing a microcontroller to properly speak the IDBUS protocol. Obtaining a fundamental knowledge about the protocol is achieved by capturing handshakes from off-the-shelf Lightning accessories and interpreting them by correlating it to previous work of security researchers, such as [1], [10], [11]. To expose the IDBUS messages, the Lightning accessory is connected through a pass-through breakout board to an Apple device. This allows us to capture the handshake, by hooking up the relevant pins of the connector to a logic analyzer. Leveraging a free of charge IDBUS analyzer plugin[4] for the Saleae Logic2 application[5] should facilitate the analysis of the protocol. After gaining the necessary knowledge about the protocol, we will endow a microcontroller with a reimplementation of the IDBUS protocol. This makes it possible to automatically capture and store Lightning accessory handshakes persistently, as well as responding to requests from the Tristar (Apple device) chip on the fly. At this point in time, we will be able to evaluate which microcontroller, from the list stated in Section 5, suits best for our use case. Independent of the chosen microcontroller, the implementation will utilize the I/O interface for emulating the IDBUS communication. However, we would like to emphasize that the Raspberry Pi Pico offers programmable I/O (PIO) ports, suitable for reimplementing custom peripheral protocols, such as IDBUS, as shown by Stacksmashing [11]. Furthermore, the IDBUS protocol operates at the 3.3 volt logic level, which requires a level converter, if the I/O interface of the chosen microcontroller operates at the commonly used $5V$ level.

### A. *Black Box Fuzzing*

Having the armed microcontroller in the toolbox, we are able to start performing black box fuzzing against the Tristar chip. The IDBUS protocol exchanges information based on the request-response pattern, where each message consists of a header, followed by the data, which can be empty, and terminated by a cyclic redundancy check (CRC). To inform the receiver that the current transmitted message acts as a response to a previously received request, the header value from the response is the incremented header value from the request. For example, if device A transmits a request to B with the header value 75, B's response to A contains the header value 76. Furthermore, requests maintain always even and responses odd header values, based on previous research [1], [10], [11]. To fulfill these protocol constraints and avoid damaging the device, the first fuzzing iteration will primarily

[4]https://github.com/nezza/SDQAnalyzer

[5]https://www.saleae.com/downloads/

rely on previously captured response messages with a mutated data field. To still cover a broad spectrum of possible request-response messages, several diverse types of accessories should be captured and incorporated into the generation phase. So far, we have access to charging cables manufactured by i) Apple, ii) authorized 3-rd party company and iii) unauthorized 3-rd party company; USB-OTG adapter; Apple official and unauthorized Haywire adapter. Nevertheless, the fuzzing input of this iteration will also contain a small set of static samples, with a wrong response header value, to verify if Tristar can handle them securely. After obtaining more evidence that sending bad IDBUS messages does not harm the device, we will start the second fuzzing iteration loop. Within this iteration, we will generate the header and data portion on the fly, as well as challenging the Tristar chip by hitting it with generated requests. The generation of the input data should be either done on the microcontroller itself, or on an external computer and transferred to the microcontroller on the fly through the USB interface. To evaluate which option suits better for our use case, we need first to agree on a microcontroller and verify if it offers enough space to store the input and result data. To achieve an efficient fuzzing performance, we need to consider how we can recover from a precipitated system crash or protocol starvation. Ideally, recovering from the aforementioned states is entirely accomplished autonomously. Based on the research from Nyan Satan [10], a reset of the internal IDBUS state machine is forced by sending a WAKE signal, which resolves starvation. If we now combine his research with the one from Stacksmashing [11], we can force a reset of the Apple device, by sending a static predefined response to the request we receive after the WAKE signal. The consecutive execution of sending the WAKE signal and RESET response should allow us to recover from system crashes.

### B. Attack Phase

In the attack phase, we will try to mount a combination of Juice-Filming and HID attack against an Apple Device. The attack will be triggered by connecting the Apple device to an evil Lightning cable, which is equipped with two microcontrollers. Ideally, both microcontrollers are powered by the charger, which erases the need for dedicated power lines. On connection, the microcontrollers within the evil cable try to mount the attack against the Apple device in two stages. First, one of the microcontrollers, a Raspberry Pi Pico, will request USB access from the Apple device, by sending the correct IDBUS messages through the programmable I/O pins. Second, the other microcontroller acts as a USB peripheral and exposing three USB interfaces, which represent a mouse, a keyboard, and an HDMI endpoint, respectively. This requires proper wiring of the IDBUS and USB data lines to the I/O pins of the Raspberry Pi Pico and to the USB interface of the second microcontroller, respectively. To guarantee that the Apple device mounts the emulated USB peripheral property, a supported and correct USB interface needs to be exposed by the microcontroller. Retrieving this information includes real world testing, where we connect the Haywire (Lightning-HDMI) directly and a predefined set of HID devices (keyboard and mouse) through an USB-OTG adapter to the Apple device and verify its functionality. After successful validation of the functionality, obtaining the USB interface properties will be done by following one of these approaches: i) connecting the USB peripheral to a Linux machine and execute 'lsusb', ii) verifying if the USB interface properties can be extracted from the captured IDBUS handshake, iii) make use of the USB interfaces proposed by Lambda Concepts [2]. The implementation order of the USB interfaces is based on the assumed difficulty, which is as follows: i) mouse, ii) keyboard, iii) HDMI adapter. Depending on the complexity, the implementation will be done either in a bare-metal fashion or on top of an operating system. Furthermore, the implementation is done in two iterations. The first iteration aims to inject static mouse and keyboard strokes, as well as storing the video-stream on the on-board flash memory. Due to the constrained amount of flash memory, the video-stream will be neither live nor long-lasting, however, iteration one aims as a proof-of-concept. If there is enough time left, a remote-control panel, allowing to inject specific HID strokes and watching the video-stream live, should be developed in iteration two. Additionally, our approach assumes that the USB communication between the Lightning accessory and the Apple device is neither encrypted nor wrapped by a highly sophisticated proprietary protocol. This can be verified very easily for the two HID devices (mouse and keyboard), by utilizing non-Apple devices in the real-world testing phase. If these are mounted properly by the Apple device, we can conclude with a high probability that the Apple device does not expect any customized USB interface or communication stack, instead it follows common standards. However, based on the research done by Lambda Concepts [2], this does not apply for the Haywire adapter. Instead, Apple exchanges the video packages following the Nero protocol which is running on top of the USB and encapsulated the H.264 video data into so called SDAT frames. In case there is no standardized protocol available for exchanging the video data, the Nero protocol needs to be reverse engineered and implemented on the microcontroller hosting the USB logic. To gain fundamental knowledge about the protocol, the USB communication between the Apple device and the adapter needs to be intercepted and analyzed, which perhaps requires a USB interceptor.

### C. Mitigation Phase

To enhance the security of Apple devices regarding Juice-Jacking attacks, we will also provide a Lightning Data Blocker to mitigate this class of attacks. The Data Blocker will assemble the idea of a Lightning pass-through-board, equipped with an IDBUS capable microcontroller, acting as a firewall. Specifically for this master thesis, a Raspberry Pi Pico will be leveraged. The firewall is configured to only pass-through charging related traffic to the Apple device and drop everything else coming from the original Hi-Five chip. To obtain a valid set of charging related messages, handshakes from various different charging setups needs to be captured and analyzed.

## V. Evaluation

This section illustrates how the correctness of the defined goals in Section 3 is proven. For carrying out the subsequently listed evaluation flows, access to the following utilities is needed: i) Logic Analyzer, ii) Microcontroller, iii) Apple device. To keep the cost affordable, we aim to leverage low cost utilities such as either the eLabGuy[6] or a self-assembled Lightning pass-through breakout board; a logic analyzer from AZ-Delivery[7]; a Raspberry Pi Pico[8] for the IDBUS protocol emulation; either the STM32F746ZG[9] or the Raspberry Pi 4 model B[10] for emulating the USB peripherals; and an iPhone 7[11] operating at iOS version 15.7. The proposed utilities are verified to be suitable for carrying out the experiments, although they can vary in case of unexpected circumstances. Additionally, the Experimental Campaign will provide sufficient information, if the Nero protocol, responsible for requesting the device video stream data, needs to be reverse engineered. For reverse engineering the Nero protocol, access to a USB interceptor is maybe required. Furthermore, the IDBUS protocol reimplementation, running on the Raspberry Pi Pico, needs to be verified, which is done within two stages. First, the emulated IDBUS signal is captured with the logic analyzer and compared with previous recorded traces from official Apple accessories. In the second stage, we will expose the Apple device to the emulated IDBUS signal and verify if it draws the same behavior as for the official accessory.

### A. Black Box Fuzzing

Identifying if fuzzing has found security related concerns is extremely hard within this context, because we neither have access to a list of valid responses nor to a datasheet explaining the message structure and their meaning. Therefore, printable or system relevant information, such as sensitive data, serial numbers, and IP addresses, are treated as sensitive data disclosure. Where sensitive data in this domain refers to any information which is not available if the device is locked. Additionally, we will try to construct a database of valid request-response samples, by analyzing and consolidating the prior captured IDBUS message of official Lightning accessories. By diffing the messages produced by the fuzzing loop with these from the database, we aim to find issues within the protocol. Furthermore, we need to check after each IDBUS interaction if the pin usage of the Lightning connector has changed. This is done by monitoring each individual pin for a fixed period via the GPIO port of the microcontroller and comparing it with a prior defined baseline. To notice if we face a system crash or protocol starvation, a well-known request-response sample should act as a heartbeat. However, with the current knowledge level, we cannot ensure that a crashed system implies an unresponsive Tristar chip.

### B. Attack Phase

The HID and Juice-Filming attack will be evaluated against the Apple device directly. Hence, the attack is successful if the assembled evil Lightning cable is recognized by the device, and we are able to video-capture the device screen as well as emulating a mouse and a keyboard.

### C. Mitigation Phase

The Lightning Data Blocker is also evaluated against the Apple device directly. We count it as a success if none of our available Lightning accessories receive data from the device and the device is charging properly after establishing a connection with one of the following devices: i) computer, ii) Haywire iii) charger.

## VI. Estimated Timeline

Table VI exhibits the estimated time to complete the various goals.

| **Task** | **Estimated Time** |
|---|---|
| **Analyzing IDBUS protocol** | **3 weeks** |
| * Get familiar with the protocol<br>* Work out the different protocol units (HIGH, LOW, etc.)<br>* Capture several handshakes | |
| **Re-Implementing IDBUS protocol** | **2 weeks** |
| * Evaluate previous work from strackmashing<br>* Finalize MCU protocol implementation<br>* Validate implementation | |
| **Attack Phase** | **10 weeks** |
| * Propose system architecture<br>* Implement IDBUS communication stack<br>* Implement/Expose custom USB interfaces<br>* Evaluate functionality of chosen architecture | |
| **Mitigation Phase** | **4 weeks** |
| * Collect guideline data (execute Juice-Jacking attacks)<br>* Propse Lightning Data Blocker architecutre<br>* Collect Lightning handshakes<br>* Implement firewal logic<br>* Evaluate Lightning Data Blocker functionality | |
| **Fuzzing** | **6 weeks** |
| * Prepare fuzzing architecture<br>* Run fuzzing iteration one<br>* Run fuzzing iteration two | |
| **Writing** | **17 weeks** |

## VII. Concerns

The previous sections have already mentioned some requirements, which need to be fulfilled to reach our defined goals from Section three. This section will address them, discuss the impact, and propose an alternative for each identified concern.

[6] http://elabguy.com/datasheet/APPLE-LF-LM-V2A\%20Rev1.0.pdf
[7] https://www.az-delivery.de/en/products/saleae-logic-analyzer
[8] https://www.raspberrypi.com/products/raspberry-pi-pico/
[9] https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html#products
[10] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[11] https://support.apple.com/kb/SP743?locale=en_US

### A. IDBUS encryption

Even though none of the related work has stated that the IDBUS protocol is encrypted, we would like to address this scenario as well. In case we are unable to interact with the Tristar chip via the IDBUS protocol, none of the previous defined goals can be fully accomplished. Due to the lack of official documentation, it is infeasible for us to identify if the messages are exchanged encrypted or unencrypted. But the main question for us is, if the IDBUS protocol contains a mechanism to mitigate replay attacks, which would imply that the same piece of information produces different IDBUS payloads. If this is the case, and we are not able to crack the encryption logic, we would document the protocol based on the facts we are able to extract from the physical layer in conjunction with the work of other security researchers. Furthermore, we would reimplement the IDBUS protocol on a Raspberry Pi Pico as good as possible and construct a fuzzing framework around it. Finally, we would investigate if higher level protocols were subject to some security issues, and at which point in time security mechanism, such as the lockdown dialog [13], are evaluated.

### B. IDBUS Message Interpretation

As already stated in Section five, due to the lack of official documentation, it will be incredibly challenging to identify invalid IDBUS messages. In case we are able to interact with the Tristar chip, but not able to identify misbehavior, proper labeling the fuzzing output will be infeasible. Therefore, we would see as the main goal of fuzzing, to provide the framework only. This includes equipping a Raspberry Pi Pico with a reimplementation of the IDBUS protocol and driver logic, including the input generation and communication with the Raspberry Pi Pico.

### C. Nero protocol

As already stated in Section three, emulating the Haywire adapter maybe requires reverse engineer the Nero protocol. In case no access to a USB interceptor is given, or reverse engineering the protocol occupies too much time, we would shift the main focus on the HID devices.

### D. Data Blocker Firewall

If, due to Concern A and B, it is impossible to classify IDBUS messages and extract a set of legitimate charging commands, we would propose to replace the firewall with a static preamble. This implies that the embedded Raspberry Pi Pico waits for the request to be sent by Tristar and acknowledges it with a static, predefined message, which enables charging only.

## References

[1] Ramtin Amin. Tristan. https://web.archive.org/web/20180524101120/http://ramtin-amin.fr:80/tristar.html, 2018. Accessed: 2022-09-14.

[2] Lambda Concept. Graywire lightning cable implant. http://blog.lambdaconcept.com/post/2019-09/graywire/, 2019. Accessed: 2022-09-04.

[3] Lijun Jiang, Weizhi Meng, Yu Wang, Chunhua Su, and Jin Li. Exploring energy consumption of juice filming charging attack on smartphones: a pilot study. In *International Conference on Network and System Security*, pages 199–213. Springer, 2017.

[4] Yuvraj Kumar. Juice jacking-the usb charger scam. *Available at SSRN 3580209*, 2020.

[5] Billy Lau, Yeongjin Jang, Chengyu Song, Tielei Wang, Pak Ho Chung, and Paul Royal. Mactans: Injecting malware into ios devices via malicious chargers. *Black Hat USA*, 92, 2013.

[6] Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing: a survey. *Cybersecurity*, 1(1):1–13, 2018.

[7] Bernhard Linke. Overview of 1-wire technology and its use. *MAXIM, AN1796, jun*, 2008.

[8] Edwin Lupito Loe, Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Shao-Chuan Lee, and Shin-Ming Cheng. Sandusb: An installation-free sandbox for usb peripherals. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 621–626, 2016.

[9] Nyan Satan. https://twitter.com/nyan_satan/status/1155148804892569604. Accessed: 2022-09-16.

[10] Nyan Satan. Apple lightning. https://nyansatan.github.io/lightning/. Accessed: 2022-09-14.

[11] stacksmashing. The hitchhacker's guide to iphone lightning & jtag hacking. https://media.defcon.org/DEFCON30/DEFCON30presentations/stacksmashing-ThehitchhackersguidetoiPhoneLightningJTAGhacking.pdf, 2022.

[12] The Iphone Wiki. Lightning connector. https://www.theiphonewiki.com/wiki/Lightning_Connector. Accessed: 2022-09-14.

[13] Dino Dai Zovi. ios lockdown diagnostic services. https://theta44.org/2014/07/28/ios-lockdown-diag-services.html, 2014.