TECHNISCHE
UNIVERSITÄT
DARMSTADT

SECURITY AND PRIVACY FOR IOT ECOSYSTEMS

vom Fachbereich Informatik
der Technische Universität Darmstadt
genehmigte Dissertation
zur Erlangung des akademischen Grades
Doktor-Ingenieurin (Dr.-Ing.)

von JISKA DOROTHEE CLASSEN, M. SC.

geboren am 26. Februar 1990 in Kassel, Deutschland.

Erstreferent: Prof. Dr.-Ing. Matthias Hollick
Korreferent: Assoc. Prof. Paul Patras (Ph.D.)

Tag der Einreichung: 6. Dezember 2019
Tag der Disputation: 20. Januar 2020

Darmstadt 2020
Hochschulkennziffer D17

SEEMOO
SECURE MOBILE NETWORKING

Jiska Dorothee Classen, *Security and Privacy for IoT Ecosystems*, Dissertation, Technische Universität Darmstadt, 2020.

Fachbereich Sichere Mobile Netze
Fachbereich Informatik
Technische Universität Darmstadt
Jahr der Veröffentlichung: 2020
Tag der mündlichen Prüfung: January 20, 2020
URN: `urn:nbn:de:tuda-tuprints-114229`
URL: `https://tuprints.ulb.tu-darmstadt.de/id/eprint/11422`

ABSTRACT

Smart devices have become an integral part of our everyday life. In contrast to smartphones and laptops, Internet of Things (IoT) devices are typically managed by the vendor. They allow little or no user-driven customization. Users need to use and trust IoT devices as they are, including the ecosystems involved in the processing and sharing of personal data. Ensuring that an IoT device does not leak private data is imperative.

This thesis analyzes security practices in popular IoT ecosystems across several price segments. Our results show a gap between real-world implementations and state-of-the-art security measures. The process of responsible disclosure with the vendors revealed further practical challenges. Do they want to support backward compatibility with the same app and infrastructure over multiple IoT device generations? To which extent can they trust their supply chains in rolling out keys? Mature vendors have a budget for security and are aware of its demands. Despite this goodwill, developers sometimes fail at securing the concrete implementations in those complex ecosystems. Our analysis of real-world products reveals the actual efforts made by vendors to secure their products. Our responsible disclosure processes and publications of design recommendations not only increase security in existing products but also help connected ecosystem manufacturers to develop secure products.

Moreover, we enable users to take control of their connected devices with firmware binary patching. If a vendor decides to no longer offer cloud services, bootstrapping a vendor-independent ecosystem is the only way to revive bricked devices. Binary patching is not only useful in the IoT context but also opens up these devices as research platforms. We are the first to publish tools for Bluetooth firmware and lower-layer analysis and uncover a security issue in *Broadcom* chips affecting hundreds of millions of devices manufactured by *Apple*, *Samsung*, *Google*, and more. Although we informed *Broadcom* and customers of their technologies of the weaknesses identified, some of these devices no longer receive official updates. For these, our binary patching framework is capable of building vendor-independent patches and retrofit security.

Connected device vendors depend on standards; they rarely implement lower-layer communication schemes from scratch. Standards enable communication between devices of different vendors, which is crucial in many IoT setups. Secure standards help making products secure by design and, thus, need to be analyzed as early as possible. One possibility to integrate security into a lower-layer standard is Physical-Layer Security (PLS). PLS establishes security on the Physical Layer (PHY) of wireless transmissions. With new wireless technologies emerging, physical properties change. We analyze how suitable PLS techniques are in the domain of mmWave and Visible Light Communication (VLC). Despite VLC being commonly believed to be very secure due to its limited range, we show that using VLC instead for PLS is less secure than using it with Radio Frequency (RF) communication.

The work in this thesis is applied to mature products as well as upcoming standards. We consider security for the whole product life cycle to make connected devices and IoT ecosystems more secure in the long term.

ZUSAMMENFASSUNG

Intelligente Geräte sind fester Bestandteil unseres täglichen Lebens geworden. Im Gegensatz zu herkömmlichen Handys und Laptops, werden Geräte im Internet der Dinge (IdD) in der Regel vom Hersteller gesteuert. Nutzer haben meist keinerlei Möglichkeit, an solchen Geräten Anpassungen vorzunehmen. Stattdessen müssen Nutzer IdD-Geräten vertrauen und diese so nutzen, wie sie sind; inklusive allen hiermit verbundenen Ökosystemen zur Datenverarbeitung. Es muss deshalb in jedem Fall sichergestellt werden, dass ein IdD-Gerät ausspähsicher ist.

Diese Dissertation analysiert Sicherheitspraktiken in weit verbreiteten IdD-Ökosystemen über verschiedene Preissegmente hinweg. Unsere Ergebnisse zeigen eine Lücke zwischen tatsächlichen Implementierungen und dem aktuellen Stand der Technik auf. Bei der Kommunikation von Sicherheitslücken an Hersteller sind uns grundlegende praxisnahe Problemstellungen aufgefallen. Möchten Hersteller Rückwärtskompatibilität innerhalb einer Handy-Anwendung mit mehreren Gerätegenerationen ermöglichen? Können sie ihrer Lieferkette die Installation von Schlüsselmaterial anvertrauen? Hersteller, die schon länger am Markt sind, sind sich über diesen Problembereich bewusst, und haben deshalb ein Budget für Produktsicherheit. Trotz dieses Bewusstseins scheitern Entwickler manchmal daran, Sicherheit in komplexen Ökosystemen korrekt anzuwenden. Unsere Analyse diverser Produkte am Markt zeigt auf, wie Hersteller Sicherheit praktisch umsetzen. Unsere offene Kommunikation über Sicherheitsproblematiken und unsere Empfehlungen helfen sowohl, die Sicherheit der direkt betroffenen Produkte zu erhöhen, als auch den Herstellern vernetzter Ökosysteme sichere Produkte zu entwickeln.

Darüber hinaus ermöglichen wir Anwendern mit Hilfe von Binärcodeänderungen, die Firmware ihrer vernetzten Geräte zu modifizieren. Sobald ein Hersteller seine Cloud-Dienste einstellt, können hierdurch unbrauchbar gewordene Geräte nun wiederverwendet werden, indem ein alternatives herstellerunabhängiges Ökosystem aufgezogen wird. Binärcodeänderungen sind nicht nur im IdD-Kontext nützlich – sie können auch dazu genutzt werden, herkömmliche Geräte für Forschungszwecke umzufunktionieren. Wir sind die ersten, die Werkzeuge für Bluetooth-Firmware zur Untersuchung der unteren Netzwerkschichten zur Verfügung stellen. Bei der Firmwareanalyse ist uns eine Schwachstelle aufgefallen, welche mehrere Hunderte Millionen von *Broadcom*-Chips betraf, die unter anderem von *Apple*, *Samsung* und *Google* verbaut wurden. Wir haben *Broadcom* und ihre betroffenen Kunden informiert, damit diese Lücken geschlossen werden können. Allerdings erhalten einige der betroffenen Geräte keine Updates mehr. Auch bei diesen können Binärcodeänderungen eingesetzt werden, um herstellerunabhängig Sicherheitsupdates für alte Geräte herauszubringen.

Hersteller vernetzter Geräte sind von Standards abhängig. Nur selten implementieren sie neue Kommunikationsschemata für die unteren Netzwerkschichten selbst, denn Standards ermöglichen die Kommunikation zwischen Geräten unterschiedlicher Hersteller, was in IdD-Installationen häufig benötigt wird. Sichere Standards helfen dabei, Produkte von Grund auf sicher zu machen, und sollten deshalb frühestmöglich analysiert werden. Eine Methode, Sicherheit in Standards für die unteren Netzwerkschichten zu implementieren, ist Sicherheit für die physikalische Schicht. Hierbei werden physikalische Eigenschaften drahtloser Übertragungen dazu genutzt, diese abzusichern. Mit neuen drahtlosen Technologien ändern sich solche physikalischen Eigenschaften. Wir analysieren, wie gut sich Sicherheit auf der physikalischen Schicht für Millimeterwellen- und Lichtkommunikation anwenden lässt. Trotz der allgemeinen Annahme, dass Lichtkommunikation aufgrund der begrenzten Reichweite sehr sicher ist, zeigen wir auf, dass sich Lichtkommunikation wegen ihrer anderen Eigenschaften insgesamt schlechter für Sicherheit auf der physikalischen Schicht eignet als herkömmliche Radiowellenkommunikation.

Diese Dissertation beschäftigt sich sowohl mit weit verbreiteten Produkten als auch mit bekannten Standards. Wir betrachten Sicherheit im gesamten Produktlebenszyklus um Sicherheit in vernetzten Geräten und IdD-Ökosystemen langfristig sicherer zu machen.

AUTHOR'S PUBLICATIONS AND COLLABORATIONS

Research on advanced topics is often a joint effort. In the following, I will give a detailed statement on collaborations for my primary and co-authored publications. I want to thank my supervisor Matthias Hollick and co-supervisor Paul Patras; my colleagues, students and collaborators at TU-Darmstadt Robin Klose, Matthias Schulz, Daniel Steinmetzer, Fabian Ullrich, Daniel Wegemer, Max Weller, Johannes Eger, Dennis Mantz, Jan Ruge, Jan-Pascal Kwiotek, Martin Pfeiffer, Ahmad-Reza Sadeghi, Hossein Fereidooni, Markus Miettinen; Tom Spink from University of Edinburgh; and Edward Knightly and Joe Chen from Rice University. In addition to scientific papers, we published at non-academic venues. Some publications also resulted in open-source software. A detailed list and statement on my publications and work at SEEMOO are listed below.

STATEMENT ON SCIENTIFIC PUBLICATIONS

My Master thesis topic was very different from my PhD topics. When I started at SEEMOO, I was still in contact with my Master thesis supervisors Florian Volk (team of Max Mühlhäuser) and Johannes Braun (group of Johannes Buchmann). We published my results as a paper and as part of a journal article [5; 19].

The first student under my supervision, Martin Pfeiffer, did a great job in localizing Terrestrial Trunked Radio (TETRA) transmissions. Since this work turned out to be more on the PHY than expected, Robin Klose also assisted us a lot. Together with TETRA fuzzing results on the link layer by my student Jan-Pascal Kwiotek, this work was published [15].

Matthias Schulz and I investigated how to implement covert channels on off-the-shelf devices [7]. I implemented the first version using raw Orthogonal Frequency-Division Multiplexing (OFDM) frames, which he re-implemented to work on standard Wi-Fi frames anyone can receive. My additional focus was the detectability of these covert channels.

During the end of my first year, from March to June 2015, I visited Edward Knightly's group at Rice University in Houston for three months. We had working in tandem with Daniel Steinmetzer focusing on mmWave eavesdropping [16; 17; 18], Joe Chen continuing mmWave work for localization [13], and me analyzing VLC eavesdropping [6; 7].

Being interested in building security on top of new communication standards, Richard Meister, a Master's Degree student I supervised, built a VLC testbed [10]. We got support from Marcos Katz and Muhammad Saad Saud for the Universal Software Radio Peripheral (USRP)-based implementation.

Paul Patras from the University of Edinburgh visited our group in summer 2016. We started our work on the Fitbit ecosystem and analyzed hardware and web components [14]. The team of Hossein Fereidooni, Ahmad-Reza Sadeghi, Markus Miettinen, and Mauro Conti specialized in the hardware parts, whom I also assisted in soldering and reverse engineering the hardware design. The team of Paul Patras and Tom Spink analyzed web communication, and I helped them in reverse engineering essential parts of the proprietary protocol. Together with Daniel Wegemer, Paul Patras, and Tom Spink, I continued reverse-engineering the Fitbit firmware [2]. My focus was on the firmware update process, including its encryption, while Daniel Wegemer supported me in patching new functionality into the firmware.

Dennis Mantz started a thesis on *Broadcom* Bluetooth chips under the supervision of Matthias Schulz. When Matthias Schulz finished his PhD, I took over the supervision of Dennis Mantz. We published the resulting framework *InternalBlue* [9]. Based on *InternalBlue*, I reverse-engineered the *Broadcom* Bluetooth diagnostics protocol [1].

Fabian Ullrich, Johannes Eger, and I analyzed the *Neato* vacuum cleaner ecosystem [11]. My main focus was the hardware part—I bypassed secure boot and extracted firmware from various *Neato* and *Vorwerk* robots. Moreover, I did the initial analysis of the firmware image and uncovered the static log and core dump encryption key.

Due to my other work in the area of IoT, Erik Tews asked me to join him in analyzing Bluetooth finders, a work currently under submission [12]. Max Weller implemented *PrivateFind*, a secure and privacy-preserving finder protocol. Discussions on how the protocol of *PrivateFind* should look like were a joint effort. Fabian Ullrich did the security review of the *Tile* ecosystem, which he found to be insecure despite a previous analysis which did not find any security issues. I wrote the initial draft of the whole paper.

PRIMARY AND CO-FIRST AUTHOR

1 Jiska Classen and Matthias Hollick. "Inside Job: Diagnosing Bluetooth Lower Layers Using Off-the-Shelf Devices." In: *12th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec). Replicability label*. 2019. DOI: 10.1145/3317549.3319727. **Part of this thesis.**

2 Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. "Anatomy of a Vulnerable Fitness Tracking System: Dissecting the Fitbit Cloud, App, and Firmware." In: *PACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*. 2018. **Part of this thesis.**

3 Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. *Demo. Modified Fitbit Firmware: Reach your Daily Goals within Seconds*. Singapore, 2018. URL: http://ubicomp.org/ubicomp2018/program/demo-schedule.pdf.

4 Jiska Classen, Daniel Steinmetzer, and Matthias Hollick. "Opportunities and Pitfalls in Securing Visible Light Communication on the Physical Layer." In: *Proceedings of the 3rd Workshop on Visible Light Communication Systems*. ACM. 2016. **Part of this thesis.**

5 Jiska Classen, Johannes Braun, Florian Volk, Matthias Hollick, Johannes Buchmann, and Max Mühlhäuser. "A Distributed Reputation System for Certification Authority Trust Management." In: *Proceedings of IEEE TrustCom*. Vol. 1. IEEE. 2015.

6 Jiska Classen, Joe Chen, Daniel Steinmetzer, Matthias Hollick, and Edward Knightly. "The Spy Next Door: Eavesdropping on High Throughput Visible Light Communications." In: *Proceedings of the 2nd International Workshop on Visible Light Communications Systems*. ACM. 2015. **Part of this thesis.**

7 Jiska Classen, Matthias Schulz, and Matthias Hollick. “Practical Covert Channels for WiFi Systems.” In: *IEEE Conference on Communications and Network Security*. IEEE. 2015.

CO-AUTHOR

8 Michael Spörk, Jiska Classen, Carlo Alberto Boano, Matthias Hollick, and Kay Römer. “Improving the Reliability of Bluetooth Low Energy Connections.” In: *International Conference on Embedded Wireless Systems and Networks (EWSN)*. 2020.

9 Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. “InternalBlue - Bluetooth Binary Patching and Experimentation Framework.” In: *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2019. DOI: 10.1145/3307334.3326089. **Part of this thesis.**

10 Richard Meister, Jiska Classen, Muhammad Saad Saud, Marcos Katz, and Matthias Hollick. “Practical VLC to WiFi Handover Mechanisms.” In: *CoWireless*. 2019.

11 Fabian Ullrich, Jiska Classen, Johannes Eger, and Matthias Hollick. “Vacuums in the Cloud: Analyzing Security in a Hardened IoT Ecosystem.” In: *The 13th USENIX Workshop on Offensive Technologies (WOOT)*. 2019. **Part of this thesis.**

12 Max Weller, Jiska Classen, Fabian Ullrich, Erik Tews, and Matthias Hollick. “Lost and Found: Stopping Bluetooth Finders from Leaking Private Information.” In: *Under submission*. 2019. **Part of this thesis.**

13 Joe Chen, Daniel Steinmetzer, Jiska Classen, Edward Knightly, and Matthias Hollick. “Pseudo Lateration: Millimeter-Wave Localization Using a Single RF Chain.” In: *Wireless Communications and Networking Conference*. IEEE. 2017.

14 Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. “Breaking Fitness Records without Moving: Reverse Engineering and Spoofing Fitbit.” In: *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. Springer, Cham. 2017.

15 Martin Pfeiffer, Jan-Pascal Kwiotek, Jiska Classen, Robin Klose, and Matthias Hollick. “Analyzing TETRA Location Privacy and Network Availability.” In: *Proceedings of the 6th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM. 2016.

16 Daniel Steinmetzer, Jiska Classen, and Matthias Hollick. “Exploring Millimeter-Wave Network Scenarios with Ray-tracing based Simulations in mmTrace.” In: *IEEE Infocom 2016 Poster Presentation*. IEEE. 2016.

17 Daniel Steinmetzer, Jiska Classen, and Matthias Hollick. “mmTrace: Modeling Millimeter-wave Indoor Propagation with Image-based Ray-tracing.” In: *Millimeter-wave Networking Workshop*. IEEE. 2016.

18 Daniel Steinmetzer, Joe Chen, Jiska Classen, Edward Knightly, and Matthias Hollick. “Eavesdropping with Periscopes: Experimental Security Analysis of Highly Directional Millimeter Waves.” In: *IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2015.

19 Johannes Braun, Florian Volk, Jiska Classen, Johannes Buchmann, and Max Mühlhäuser. "CA Trust Management for the Web PKI." In: *Journal of Computer Security* 22.6 (2014).

NON-ACADEMIC PUBLICATIONS AND TALKS

Academic publications are significant in a scientific context. However, many of these fail in publishing results with open access and recording videos. Scientific papers are often invisible to the press and the public.

Thus, I often decided to publish work in progress as a result of public events. I especially want to thank the Chaos Communication Congress (C3) organization teams for supporting the growth of the events and allowing me to talk in lecture halls as big as 3000 seats in Hamburg [37] and 5000 seats in Leipzig [31]. The C3 Video Operation Center (C3VOC) did a fantastic job in recording the talks, streaming them live, and publishing recordings within hours. Despite not being scientifically recognized, 35C3 had over 600 submissions, of which only 150 were accepted. 36C3 had 690 submissions, 194 of those in security, and only 39 of the security talks were taken [12]. DEF CON 27 in Las Vegas had an acceptance rate of less than 10 %.

20 Jiska Classen. *All Wireless Communication Stacks are Equally Broken*. Leipzig: 36. Chaos Communication Congress, 2019.

21 Jiska Classen. *Vacuums in the Cloud*. Karlsruhe: OWASP, 2019.

22 Jiska Classen and Johannes Eger. *Smart Vacuum Cleaners as Remote Wiretapping Devices*. Wien: Easterhegg, 2019. URL: `https://media.ccc.de/v/eh19-157-smart-vacuum-cleaners-as-remote-wiretapping-devices`.

23 Jiska Classen and Dennis Mantz. *Bluetooth, Does it Spark Joy?* Heidelberg: TROOPERS, 2019. URL: `https://www.troopers.de/troopers19/agenda/smsm3l/`.

24 Jiska Classen and Dennis Mantz. *Playing with Bluetooth*. Darmstadt: MRMCD, 2019. URL: `https://media.ccc.de/v/2019-185-playing-with-bluetooth`.

25 Jiska Classen and Dennis Mantz. *Reversing and Exploiting Broadcom Bluetooth*. Montreal: REcon, 2019. URL: `https://cfp.recon.cx/reconmtl2019/talk/EQTRGU/`.

26 Fabian Ullrich and Jiska Classen. *Vacuum Cleaning Security—Pinky and the Brain Edition*. Las Vegas: DEF CON 27, 2019. URL: `https://www.defcon.org/html/defcon-27/dc-27-speakers.html#jiska`.

27 Daniel Wegemer, Carolin Groß, and Jiska Classen. *A Security Researchers Guide into the Fitbit Ecosystem*. Las Vegas: IoT Village at DEF CON 27, 2019. URL: `https://www.iotvillage.org/`.

28 Jiska Classen. *Pinky & Brain are Taking Over the World with Vacuum Cleaners*. Darmstadt: MRMCD, 2018. URL: `https://media.ccc.de/v/2018-124-pinky-brain-are-taking-over-the-world-with-vacuum-cleaners`.

29 Jiska Classen. *Praktische IoT-Sicherheit am Beispiel von Wearables und Smart Home*. Frankfurt: Deka Bank, 2018.

30 Jiska Classen. *ma1lm4n.mp4*. Leipzig: 35. Chaos Communication Congress, 2018. URL: `https://media.ccc.de/v/35c3-9566-lightning_talks_day_2#t=219`.

31 Jiska Classen and Dennis Mantz. *Dissecting Broadcom Bluetooth*. Leipzig: 35. Chaos Communication Congress, 2018. URL: https://media.ccc.de/v/35c3-9498-dissecting_broadcom_bluetooth. **Talk in a lecture hall with 5000 seats and Bluetooth live demos.**

32 Jiska Classen and Daniel Wegemer. *Create your own Fitness Tracker Firmware*. Montreal: REcon, 2018. URL: https://recon.cx/2018/montreal/schedule/events/118.html.

33 Jiska Classen and Daniel Wegemer. *Hacking your Fitbit*. Würzburg: Easterhegg, 2018. URL: https://media.ccc.de/v/TNYPFB.

34 Jiska Classen. *Leaking and Modifying Fitbit Data*. Frankfurt: Continental AG, 2017.

35 Jiska Classen and Daniel Wegemer. *Doping your Fitbit*. Leipzig: 34. Chaos Communication Congress, 2017. DOI: 10.5446/34791. URL: https://media.ccc.de/v/34c3-8908-doping_your_fitbit.

36 Jiska Classen and Daniel Wegemer. *Leaking and Modifying Fitbit Data*. Darmstadt: MRMCD, 2017. URL: https://media.ccc.de/v/3T9E8Y.

37 Jiska Classen. *Building and Breaking Wireless Security*. Hamburg: 32. Chaos Communication Congress, 2015. URL: https://media.ccc.de/v/32c3-7119-building_and_breaking_wireless_security. **Talk in a lecture hall with 3000 seats.**

38 Jiska Classen. *Wireless Physical Layer Security*. Darmstadt: MRMCD, 2015. URL: https://media.ccc.de/v/MRMCD15-7011-wireless_physical_layer_security.

SOFTWARE

A lot of work in the scope of this thesis focuses on real-world IoT ecosystems. To enable users to take control of their devices and add custom behavior, we published various software. This software supports modified and privacy-preserving usage of *Fitbit* fitness trackers [43; 41], as well as experimentation with *Broadcom* Bluetooth chips [39; 42; 40]. In contrast to the original software provided by those vendors, our software is open source.

39 Jiska Classen. *Bluetooth H4 Broadcom Wireshark plugin from the InternalBlue project*. https://github.com/seemoo-lab/h4bcm_wireshark_dissector. 2019.

40 Jiska Classen and Kristoffer Schneider. *Nexmon for Bluetooth*. https://github.com/seemoo-lab/nexmon/tree/bluetooth-wip. 2019.

41 Jiska Classen and Daniel Wegemer. *Fitbit Firmware Modifications*. https://github.com/seemoo-lab/fitness-firmware. 2019.

42 Dennis Mantz and Jiska Classen. *InternalBlue Broadcom Bluetooth Experimentation Framework*. https://github.com/seemoo-lab/internalblue. 2019.

43 Steffen Kreis, Johannes Riedel, Tobias Krichel, Jiska Classen. *Fitbit Open Source Android App*. https://github.com/seemoo-lab/fitness-app. 2019.

The practical work with deployed systems and off-the-self-hardware uncovered several vulnerabilities. All vulnerabilities were responsibly disclosed. Some of these were submitted as Common Vulnerabilities and Exposure (CVE) or got public acknowledgments by the vendor. I want to thank *Fitbit*, *Neato*, *Broadcom*, *Cypress*, *Google*, and *Apple* for properly handling my disclosure requests and allowing me to publish results.

CVE-2019-18614 A buffer size misconfiguration in the *CYW20735* evaluation board firmware leads to a heap overflow that can be triggered as a local user and over the air.

CVE-2019-15063 On *Broadcom* Bluetooth Wi-Fi combo chips, an attacker who has execution within the Bluetooth chip can crash Wi-Fi. This crash is persistent until the complete smartphone/device reboots; without a reboot, the user cannot re-enable Wi-Fi. Some *Android* and *iOS* systems produce a kernel panic due to this and immediately reboot on their own. To trigger this behavior, the attacker needs to write to the global coexistence registers. This vulnerability shows that Bluetooth and Wi-Fi are not adequately isolated, despite running on different ARM cores.

CVE-2019-6994 An issue was discovered on *Broadcom BCM4335C0* chips as used with the Nexus 5 phone. The attacker initiates Secure Simple Pairing (SSP) on classic Bluetooth. The device under attack does nothing, or at least the user does not immediately accept/reject the pairing request. The device under attack has no established pairing to another device. While the confirmation display for the pairing is still open, the attacker sends an `LMP_start_encryption_req`, which causes the Bluetooth stack of the device under attack to crash. The crash happens within the `bignum_xormod` calculation.

CVE-2018-19860 *Broadcom* Bluetooth firmware built before summer 2014, as well as some later builds, does not properly restrict Link Manager Protocol (LMP) commands with opcode 0 and executes particular memory contents upon receiving an LMP command, as demonstrated by executing an Host Controller Interface (HCI) command.

Broadcom did not further specify which devices are affected, but we can confirm at least the following devices: MacBook Pro early 2011 *BCM4331*, MacBook Pro mid 2012 *BCM20702A3*, MacBook Pro early 2015, MacBook Pro 2016, iPhone 5, iPhone 5s *BCM43342*, iPhone 6 *BCM4345*, Nexus 5 *BCM4335C0*, Xperia Z3 Compact *BCM4335C0*, Xperia Z5 *BCM4356A2*, Raspberry Pi 3 *BCM43438A1*, Huawei Honor 8, Samsung Galaxy Note 3 *BCM4335C0*. iOS update 12.1.3 fixed iPhones.

CVE-2018-20785 A secure boot bypass and memory extraction exists in *Neato Botvac Connected* 2.2.0 devices and later. Firmware is stored in a signed and encrypted format on a flash chip. Regular firmware updates contain the same secure format. During startup, *AM335x* secure boot decrypts and executes firmware. Secure boot can be bypassed by connecting to the *Neato* USB serial and entering `TestMode On` followed by `SetSystemMode PowerCycle`. This power cycle does not completely reset the chip—memory contents are still in place. Yet, it restarts into a boot menu that enables XMODEM upload and execution of an unsigned QNX Image File System (IFS), thereby bypassing secure boot. Moreover, the attacker can craft a custom QNX IFS and locate it to unused memory to extract all memory contents that were present before. Memory contents include the original firmware and sensitive information, such as Wi-Fi credentials. Boot menu and system upload are accessible via a secondary unlabeled serial interface with three wide pins in the back on the right-hand side.

CVE-2018-17177 An issue was discovered on *Neato Botvac Connected* 2.2.0 and *Botvac* 85 1.2.1 devices. Static encryption is used for the copying of so-called "black box" logs (event logs and core dumps) to a USB stick. These logs are RC4-encrypted with a 9-character password of `*^JEd4W!I` that is obfuscated by hiding it within a custom `/bin/rc4_crypt` binary.

IOS AND ANDROID The *iOS* 13 updates additionally recognize us for our work on Bluetooth [App19a]. Moreover, the *Android* updates recognize our Bluetooth CVEs [And19a].

FITBIT Fitbit acknowledges our results [Cla+18a; Fer+17a] and cooperation with the following statement: "Thank you to the teams from the University of Padua, Technische Universität Darmstadt, and the University of Edinburgh for their assistance." [Fit18]

Fitness tracker firmware updates with this acknowledgment: Alta 21.40.2, Alta HR 26.63.2, Blaze 17.8.402.1, Charge 8.124, Charge 2 22.55.2, Charge HR 18.128, Flex 2 24.30.2, Flex 7.88, One 6.64, Surge 16.34.6.1.

TEACHING AND PRESS ACTIVITY

Besides publications, this work also reached out to students and public media. In the following, an overview of supervised theses, courses, and lectures, as well as press activity, is provided. In total, I supervised or co-supervised 21 successfully finished Bachelor and Master theses.

SUPERVISED STUDENTS

I want to thank all students for their excellent work and collaboration. They gave me new perspectives and enabled me to work on multiple topics in parallel.

Some of the theses and works of students were published in a scientific context. Max Weller's work as HiWi on Bluetooth finders is under review [12]. Fabian Ullrich continued to work at SEEMOO after his thesis, co-supervised Johannes Eger, and we published our work on *Neato* vacuum cleaners at WOOT [11]. Dennis Mantz' thesis resulted in a MobiSys publication [9]. Moreover, I continued his research, leading to a WiSec publication [1], and further collaborations with the University of Brescia and TU Graz are ongoing. His great work was awarded the *CAST Förderpreis* and the *Datenlotsenpreis*. The results of Richard Meister, Jan-Pascal Kwiotek and Martin Pfeiffer were published at workshops [10; 15].

To give outstanding results appropriate to the audience, I also supported my students in presenting their work on public events. Johannes Eger and I presented work in progress results on the *Neato* vacuum cleaning robots in Vienna [22]. Dennis Mantz and I gave a talk, including a Bluetooth live demo in a lecture hall with 5000 seats [31]. We were invited to give this talk a second time at TROOPERS [31] and presented an updated talk at REcon [25]. Fabian Ullrich presented his work on Nello door openers in Darmstadt [Ull18b]. Daniel Wegemer and I presented our Fitbit results along with further findings of Matthias Hanreich at REcon in Montréal [32].

Some of my students even went abroad for collaborations and research experiences. Jan Ruge got funding to attend DEF CON 27 in Las Vegas. Carolin Groß got a *CROSSING Female Student Mentoring and Networking* travel grant and joined me for the Ubicomp conference in Singapore. Richard Meister visited the University of Oulu in Finland during his thesis. Sven Neubauer and Jannik Jürgens both used equipment in Köln provided by Dr. Ralf Reckenfelderbäumer, until we got our TETRA base station.

44 Patrick Dworski. "A Study on Proprietary Communication Protocols Used in TETRA Hardware Components." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2019.

45 Johannes Eger. "Analyzing Firmware and Cloud Security of a Premium IoT Ecosystem." Supervised by Fabian Ullrich and Jiska Classen. Master thesis. TU Darmstadt, 2019.

46 Carolin Gross. "A researcher's guide to the Fitbit Ionic smartwatch." Supervised by Jiska Classen and Daniel Wegemer. Master thesis. TU Darmstadt, 2019.

47 Uwe Müller. "PowerPC Binary Patching and dissecting of TETRA Base Station." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2019.

48 Florentin Putz. "Secure Device Pairing Using Short-Range Acoustic Communication." Supervised by Flor Álvarez and Jiska Classen. Master thesis. TU Darmstadt, 2019.

49 Jan Ruge. "Dynamic Bluetooth Firmware Analysis." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2019.

50 Tim Walter. "Fuzzing the Linux Bluetooth Stack." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2019.

51 Matthias Hanreich. "Security Analysis and Firmware Modification of Fitbit Fitness Trackers." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2018.

52 Dennis Mantz. "InternalBlue - A Bluetooth Experimentation Framework Based on Mobile Device Reverse Engineering." Supervised by Matthias Schulz and Jiska Classen. Master thesis. TU Darmstadt, 2018.

53 Sven Neubauer. "Angriffsanalyse einer TETRA-Basisstation." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2018.

54 Marco Plaue. "Sicherheit funkferngesteuerter Rangierlokomotiven." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2018.

55 Fabian Ullrich. "Analysing and Evaluating Interface, Communication, and Web Security in Productive IoT Ecosystems." Supervised by Jiska Classen and Max Maass. Master thesis. TU Darmstadt, 2018.

56 Muneeb Ahmed. "Improving a Linux Device Driver for Visible Light Communication." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

57 Serafettin Ay. "Detecting WiFi Covert Channels." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

58 Jannik Jürgens. "TETRA Security Analysis by Fuzzing." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

59 Tim Kornhuber. "Implementation of a physical layer for visible light communication using the OpenVLC platform." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2017.

60 Felix Kosterhon. "Absicherung von SCADA-Protokollen." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2017.

61 Michael Kümpel. "Implementierung des unteren MAC-Layers für die OpenVLC-Hardware." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2017.

62 Richard Meister. "Design and Evaluation of a Hybrid SDR Testbed For Visible Light Communication and Wi-Fi." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

63 Jan-Pascal Kwiotek. "TETRA Fuzzing." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2016.

64 Martin Pfeiffer. "Location Privacy of Digital Trunked Radio." Supervised by Jiska Classen and Robin Klose. Master thesis. TU Darmstadt, 2016.

own master thesis

65 Jiska Classen. “Reputation Systems for Trust Management in the Web PKI.” Supervised by Johannes Braun and Florian Volk. Master thesis. TU Darmstadt, 2014.

teaching

SEEMOO does an exceptional amount of teaching and is well-known for this among students. When joining SEEMOO, I decided to start a new course from scratch, which enables computer scientists to understand the basics of wireless communication, building simple radio circuits, and experimenting with Software-Defined Radios (SDRs). Designing a new course and providing all the materials was very time-consuming, especially during my first year.

wireless network for emergency response: fundamentals, design, and build-up from scratch Full course on crisis communication each winter term (WS14/15, WS15/16, WS16/17, WS17/18, WS18/19) including teaching with teaching permit, organization of labs and HiWis, and excursions.

physical-layer security in wireless systems Two lectures on jamming, integrity, and authentication on the PHY each winter term (since WS14/15).

network security One lecture on IoT security each summer term (since SS18).

secure mobile systems One lecture on VLC security each summer term (since SS15).

seminars and labs Each semester I offered multiple seminar and lab topics.

public television and media

IoT devices are used by a lot of users who care about their data. Thus, the media has a high interest in IoT. *Hackaday* covered our findings on *Fitbit* fitness trackers [74], and *ZDF* also featured these in television [72]. *MDR* had a more general show on IoT espionage in everyday living and interviewed us on Chinese vacuum cleaners [69]. For a television feature on emergency communication, *Hessenschau* also included these vacuum cleaners in their show [68]. *Neato* and *Vorwerk* vacuum cleaners are more relevant to the German market, and we were interviewed for the podcast *Netzagent* by *SWR* [67] as well as by the local newspaper *Darmstädter Echo* [66].

Despite being more technical, our work on Bluetooth also got media attention, because it affects hundreds of millions of devices. We were interviewed by *Golem*, *Hackaday*, and *dpa* [70; 71; 73].

66 Sabine Schiner. *Hacker im Wohnzimmer - Wissenschaftler der TU Darmstadt decken Schwachstellen bei Vorwerk-Saugroboter auf*. 2019. url: https://www.echo-online.

de/panorama/wissenschaft/wissenschaft/wissenschaftler-der-tu-darmstadt-decken-schwachstellen-bei-vorwerk-saugroboter-auf_20434863.

67 Kirsten Tromnau. *Die Spione im eigenen Haus*. 2019. URL: https://www.swr.de/swraktuell/radio/netzagent/Die-Spione-im-eigenen-Haus,av-o1150171-100.html.

68 Hessenschau. *Kommunikation trotz Netzausfall*. 2018. URL: https://www.hessenschau.de/tv-sendung/hessenschau---ganze-sendung,video-77832.html.

69 Wolfram Huke. *Spion im Wohnzimmer*. 2018. URL: https://www.mdr.de/video/mdr-videos/c/video-175524.html.

70 Moritz Tremmel. *DoS-Angriff auf Bluetooth-Chips von Broadcom*. 2018. URL: https://www.golem.de/news/sicherheitsluecke-dos-angriff-auf-bluetooth-chips-von-broadcom-1901-138454.html.

71 Elliot Williams. *35C3: Finding Bugs In Bluetooth*. 2018. URL: https://hackaday.com/2018/12/30/finding-bugs-in-bluetooth/.

72 Markus Wolsiffer. *Datenschutz bei Wearables - Wie sicher sind meine Daten bei Smartwatch und Co.?* 2018. URL: https://www.zdf.de/verbraucher/volle-kanne/datenschutz-bei-wearables-102.html.

73 Peter Zschunke. *Forscher warnen vor Bluetooth auf älteren Smartphones*. 2018.

74 Brian McEvoy. *34C3: Fitbit Sniffing and Firmware Hacking*. 2017. URL: https://hackaday.com/2017/12/29/34c3-fitbit-sniffing-and-firmware-hacking/.

ACKNOWLEDGMENTS

Special thanks for giving helpful advice while writing this thesis goes to Matthias Hollick and Paul Patras.

I also want to thank my colleagues, who supported me during my research. Our secretary Doris Müller helped me with the almost infinite amounts of paperwork caused by my travels and hardware orders. Max Maass was proofreading most of my papers. Lars Almon was always there for technical support. Despite not yet resulting in publications, we were discussing a lot of cryptographic concepts with the Cryptoplexity group.

I want to thank Matthew Lloyd, Flor Alvarez, Milan Stute, Philipp Tiesel, and Tim Walter for proofreading my thesis. Moreover, I want to thank Oliver Pöllny and Bianca Mix for proofreading my papers and the interesting discussions on Bluetooth and fitness trackers.

Last but not least, my black unicorn was always traveling with me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ADB	Android Debug Bridge
AFH	Adaptive Frequency Hopping
AoA	Angle of Arrival
API	Application Programming Interface
ARM	Advanced RISC Machine
BER	Bit Error Rate
BLE	Bluetooth Low Energy
BPCS	Broadcom Proprietary Control Signaling
BSI	Bundesamt für Sicherheit in der Informationstechnik
C3	Chaos Communication Congress
C3VOC	C3 Video Operation Center
CCC	Chaos Computer Club
CERT	Computer Emergency Response Team
CLI	Command Line Interface
CMAC	Cipher-based Message Authentication Code
CSI	Channel State Information
CSK	Color-Shift Keying
CVE	Common Vulnerabilities and Exposure
DCO	Direct Current Offset
DoS	Denial of Service
EAX	Encrypt-then-Authenticiate-then-Translate
ECDH	Elliptic Curve Diffie-Hellman
EEPROM	Electrically Erasable Programmable Read-Only Memory
FPGA	Field-Programmable Gate Array
GATT	Generic Attribute Profile
GDB	GNU Debugger
GDPR	EU General Data Protection Regulation
GPIO	General-Purpose Input/Output

GPS	Global Positioning System
HCI	Host Controller Interface
IdD	Internet der Dinge
IFS	Image File System
IM	Intensity Modulation
IO	Input-Output
IoT	Internet of Things
IPL	Initial Program Load
IQ	In-phase and Quadrature
ISM	Industrial, Scientific and Medical
JSON	JavaScript Object Notation
KNOB	Key Negotiation of Bluetooth
LCP	Link Control Protocol
LED	Light-Emitting Diode
LE SC	LE Secure Connections
LMP	Link Manager Protocol
LOS	Line-of-Sight
LTE	Long-Term Evolution
MAC	Media Access Control
MITM	Machine-in-the-Middle
NAT	Network Address Translation
NDA	Non-Disclosure Agreement
NFC	Near-Field Communication
NLOS	Non-Line-of-Sight
OB	Orthogonal Blinding
OFDM	Orthogonal Frequency-Division Multiplexing
OOK	On-Off Keying
OTP	One-Time Pad
PCIe	Peripheral Component Interconnect Express

PD	Photo Diode
PGP	Pretty Good Privacy
PHY	Physical Layer
PLS	Physical-Layer Security
PoC	Proof of Concept
PPC	PowerPC
PSIRT	Product Security Incident Response Team
RAM	Random Access Memory
RF	Radio Frequency
RFCOMM	Radio Frequency Communications
ROM	Read-Only Memory
RSA	Rivest–Shamir–Adleman
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
SDR	Software-Defined Radio
SNR	Signal-to-Noise Ratio
SRAM	Static RAM
SSP	Secure Simple Pairing
SWD	Serial Wire Debug
TETRA	Terrestrial Trunked Radio
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
UUID	Universal Unique Identifier
VLC	Visible Light Communication
WARP	Wireless Open-Access Research Platform
XOR	eXclusive OR
XTEA	eXtended Tiny Encryption Algorithm
ZFBF	Zero-Forcing Beamforming

ZLL	Zigbee Light Link

Part I

INTRODUCTION

Chapter 1 introduces this thesis by motivating the key research challenges. Connections between the investigated topics are drawn, and practical relevance is highlighted. It also details the structure of the thesis.

1 INTRODUCTION

1.1 PROBLEM STATEMENT

Recent smart devices require connectivity to external services for basic functionality. The leading Internet of Things (IoT) devices we analyzed all depend on an external cloud [Cla+18a; Ull+19]. Cloud dependency poses various threats to privacy and security. On average, a smartphone connects to tracking-enabled services every 2 seconds [Elv18]. Even devices as simple as connected television are known to be used for governmental espionage [Wik14]. Opting out from connected services comes with huge drawbacks for users, if possible at all. Despite the growing importance of IoT and high user dependency on IoT devices, there is almost no regulation. Customers buy interesting gadgets and entrust them with highly privacy-sensitive information such as health data, details about their social life, and habits. As smart and connected devices have become an integral part of our everyday life, their security matters to everyone.

As of now, there is too little security research on connected devices for the common good. Intelligence services do a lot of security research, and there are popular platforms like *Zerodium* that sell zero-days [Zer19]. These zero-days might never become public. Despite a lot of security analysis being non-public, some research also gets intentionally published as part of responsible disclosure with the vendor, and privacy breaches or security incidents due to malicious attacks get media attention. Only this fraction of published security issues reaches regular users. While some users might not know about the risks, most users will ignore them and continue using connected devices. There remain open questions: What can research do to improve the situation? Are there tools that will even help the users? Can researchers push vendors to make their products more secure?

In the following, we outline why the current situation is alarming from a customer perspective, as well as how new standards need to be considered when designing secure solutions. We go through the whole life cycle of a device from a vendor and customer perspective, as illustrated in Figure 1, and show where security is relevant and can be improved. We consider three significant phases in a product's life cycle, which are product design (Section 1.1.1), release (Section 1.1.2), and end of life (Section 1.1.3). Throughout this thesis we then address pertinent challenges to all phases of this life cycle.

1.1.1 *Design*

The design phase is the most fundamental phase to guarantee a decent level of privacy and security in a product. The fact that fixing bugs later costs a lot is well-known, and a study by the *IBM System Science Institute* dating back to 2010 estimates costs to be 100 times higher when fixing a bug after release than during design [Daw+10].

When security and privacy breaches become public, this is considered as a marketing disaster from a vendor perspective, which might lead to dramatic reputation loss or severe financial consequences [Cox19]. Depending on the leaked data's sensitivity, users of the product might experience economic and social damage, such as rising health insurance rates or love affairs becoming public [Tho15].

Still, the egregious state of IoT security in practice due to fundamental security issues indicates that the design phase and initial testing are often underestimated by device vendors. Possible explanations include the cost of security considered to be prohibitive while designing a product on a budget, as well as security concerns trailing feature requests in the race to be first in the market. Even though many issues can be fixed in software after the product release, it is hard to impossible to change the overall ecosystem and trust assumptions for a connected device later. Even specifications and regulations sometimes fail to ensure a basic level of privacy and security.

ARCHITECTURAL DECISIONS AND FAILURES Communication paradigms and protocols, including trust anchors, should be designed carefully. The rollout of secure keys and certificates can be done in the factory. Trusted keys assist device and software update verification after product release. Moreover, depending on the radio equipment within a product's hardware, only specific communication standards can be used at the lower layers. If a vendor decides to use a technology that is not commonly available in smartphones, an additional gateway has to be part of the ecosystem (Chapter 2).

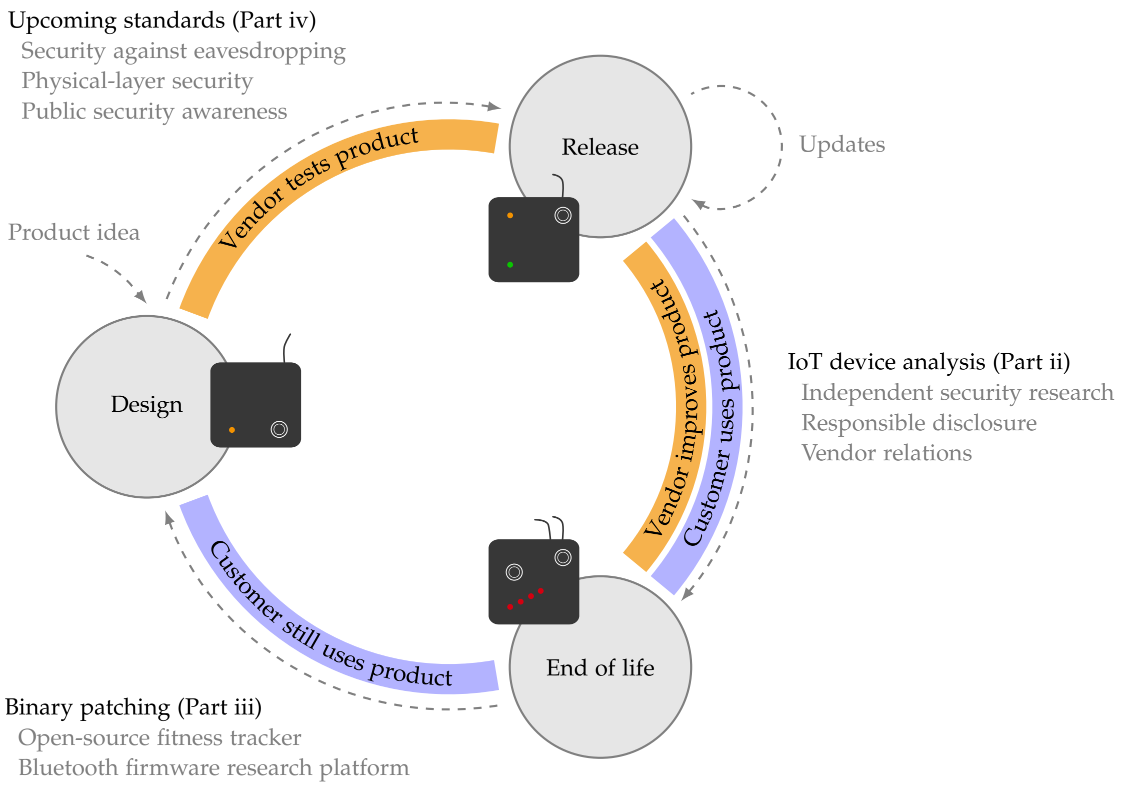

Upcoming standards (Part iv)
Security against eavesdropping
Physical-layer security
Public security awareness
Vendor tests product
Release
Updates
Product idea
Design
Vendor improves product
Customer uses product
IoT device analysis (Part ii)
Independent security research
Responsible disclosure
Vendor relations
Customer still uses product
End of life
Binary patching (Part iii)
Open-source fitness tracker
Bluetooth firmware research platform

Figure 1: Product life cycle from a security perspective aligned with the contributions of this thesis.

Note that some architectural decisions incur additional costs in infrastructure and during rollout. While a high price does not guarantee the security of a product, low price products cannot provide costly security mechanisms. Especially mechanisms based on hardware such as secure boot and trusted elements increase costs.

As the time to market is vital for many IoT products, and the initial development investment is low, the basic functionality is usually prioritized over a secure architecture. Poor architectural decisions in a connected ecosystem can lead to almost impossible to fix legacy issues, as detailed in the following.

POOR MITIGATION OF LEGACY ISSUES Throughout the work on multiple connected products within this thesis, we noticed that even mature products suffer from legacy issues introduced during the design of an early release. Fixing trust assumptions in devices that cannot be trusted is almost impossible.

For example, *Fitbit* failed to enforce encryption on older trackers, despite the rollout of proper encryption keys in the factory. To mitigate this issue after product shipment, they invented a trial encryption scheme that probes if trust can be established, yet relies on the assumption that the tracker and its Internet connection have not already been compromised (Chapter 3). *Neato* vacuum cleaning robots use secure boot and encrypted firmware updates to prevent attackers from firmware extraction. Still, once an attacker obtained a plaintext image from one robot, she can impersonate arbitrary robots (Chapter 4). *Broadcom* Bluetooth chips trust the host and accept malicious firmware patches, even in the most recent devices (Chapter 8). We noticed that *Apple* introduced an additional signature for *Broadcom* patch files in *macOS*, but as the chips themselves do not employ any checks, this signature can only be checked on the host side and thus be bypassed. Such issues as present in *Fitbit*, *Neato*, and *Broadcom* devices can only be fully fixed by releasing a new generation of devices that break with the old ecosystem.

ISSUES IN STANDARDS AND REGULATIONS Vendors have to deal with official regulations, which means that they must give statements on those aspects of their products covered by these regulations [Cou16]. Moreover, if there are best practices or well-designed specifications and frameworks, they are likely to adopt those [Eur19]. These guidelines are emerging and are a good step, but as of now, they are insufficient and not legally binding. Thus, it is significant to test upcoming technologies before they are widely deployed.

Even mature specifications can be error-prone, e.g., the Bluetooth specification had various vulnerabilities that enabled eavesdropping throughout multiple versions (Chapter 8). We specifically looked into upcoming specifications on the lower layers that have valuable Physical Layer (PHY) properties for IoT applications. We found that mmWave and Visible Light Communication (VLC) were claimed to be secure against eavesdropping by design, due to their physical properties, although not being the case in practice (Chapter 9). Besides, various schemes for Physical-Layer Security (PLS) in VLC were designed by other researchers, but lack security properties against common attacks (Chapter 10).

1.1.2 *Release*

When a product becomes publicly available, anyone can analyze its security—for good and bad purposes. As researchers, it is our responsibility to use our knowledge for product improvements and customer protection. Products might not fulfill regulations and lack internal vulnerability testing prior to release. Issues found by external parties can stay unpatched for a long time, and in some scenarios, security updates are infeasible or cannot be enforced.

DISCREPANCY BETWEEN PRODUCTS AND REGULATIONS The issues originating from the design phase occurred despite various existing recommendations, specifications, standards, and regulations. Our practical experience shows that working on the specification and regulation side is indeed necessary—but does not guarantee privacy and security in released products.

Laws in the form of EU General Data Protection Regulation (GDPR) were put into practice to counteract data collection and processing without customer knowledge. However, this is only a legal restriction and might differ a lot from the technical reality. When analyzing Bluetooth finders and informing vendors (Chapter 5), we found one of them making false claims about how often position data is sent to their servers. They did not claim this on purpose—they outsourced smartphone app and server daemon development, and they never received the smartphone app's source code.

This example emphasizes that even if there are official regulations, connected products need to be analyzed after release to identify security and privacy flaws. There is no legal guarantee that a product only does what it is supposed to do and implements this functionality securely.

THE URGENCY OF EXTERNAL TESTS IoT vendors are allowed to decide their security investments on their own. There are no market restrictions for IoT, which differs from the regulated banking and medical sectors [Cou93; Org15]. Customers will not know how thoroughly a product was tested and by whom. So far, there is no precise testing and labeling scheme to make products comparable.

As there is no effective regulation during the design phase, it becomes necessary that external parties test products after release. Our analysis of various IoT related products shows that they indeed lack security (Chapters 3, 4, 5, 8). However, most users cannot perform security testing on their own. Users have to trust data protection within cloud services, the security of the smartphone app, as well as the security of devices they install in their smart home Wi-Fi. They need to rely on the vendor and security researchers who publish their work.

SECURITY BUGFIXING DELAYS Identifying security issues is only the first step. After a vendor is informed about a problem, they must take the report seriously and act upon the findings.

During responsible disclosure, we encountered positive and negative experiences. Overall, in practice, responsible disclosure is no easy task. The worst example is a Bluetooth finder vendor, who did not react to contact by email, mail, Chinese email to a developer, various social media, telephone, and relayed communication via CERT/CC.

Customer data is still leaking through their servers, one year after our first contact attempt (Chapter 5). Moreover, although disclosing to the *Neato* developers about security issues with their vacuum cleaning robots on March 12 2018, it took repeated contact attempts until they answered on November 27 2018 (Chapter 4).

Even if a company provides a security contact and has a dedicated team working on security, responsible disclosure can take a while. *Broadcom* has a dedicated Product Security Incident Response Team (PSIRT) but it took them one month to reply to our description of the first vulnerability we found within their Bluetooth firmware (Chapter 8). They argued that the malicious payload was not standard compliant and had forwarded the initially Pretty Good Privacy (PGP) encrypted mail in plaintext internally. It took them another two weeks to set up a phone call, and even longer until they confirmed that the vulnerability is genuine. While the *Cypress* PSIRT was faster in answering our emails, they also forwarded them in plaintext, including attachments. Interestingly, even the *Apple* security team who usually encrypts and signs their mails was asking us in plaintext about Bluetooth related slides of an upcoming conference.

IMPRACTICAL SECURITY UPDATES The update process should be as simple as possible. Otherwise, devices might stay unpatched despite updates.

One example of poor usability in software updates is the first generation of connected *Neato* vacuum cleaners (Chapter 4). Despite app support and cloud connection, the only way to update them is to download a software update to a Universal Serial Bus (USB) stick, use a special connector cable, remove the trash bin, and start the update via the few buttons on the vacuum cleaner. We assume that most owners of this robot series never updated their devices. *Neato* could enforce updates by blocking robots with legacy software for communicating with their cloud. However, many users might lack technical tools and skills to perform an update.

1.1.3 *End of Life*

Once a vendor decides to stop support of a product or goes bankrupt, IoT products that rely on external Internet services are bricked. For example, when the GDPR became effective, *Yeelight* decided to shut down services in Europe instead of fixing their privacy issues, leaving smart light bulbs in many homes dysfunctional [Liu18]. While these are rather extreme edge cases, vendors usually decide to stop software updates after the warranty ends. The software update period is comparably short, considering a product's actual lifetime—when users replace the battery of a device, it can last for many years. *Neato* releases software updates only for two years, and *Apple* provides updates for a comparably long period of guaranteed four years, with some models even reaching six years [Zep19]. For devices that are still somewhat functional and connected to the Internet, missing updates become a security risk.

BRICKED FOREVER Devices are locked to the vendor's ecosystem in terms of availability. Since each vendor invents their own proprietary IoT ecosystem, the central cloud component within it cannot easily be replaced. Usually, there is no alternate ecosystem for a connected device (Chapters 3, 4). How much an IoT product depends on external infrastructure and apps is not advertised by most vendors and hard to figure out for the user.

If a vendor goes bankrupt or decides to restrict certain features to paying the premium accounts, the user has to invest money despite owning a technically functional product. Moreover, old devices might no longer get software updates, which is not only a problem for the functionality of the device itself but also interoperability with smartphone apps and ecosystems. A smart lighting system, fridge, or washing machine could stop working from one day to the next. Short-term outages of the vendor's ecosystem can already be critical, i.e., smart door locks could neither open nor close apartments for a couple of hours.

INSECURE ZOMBIES Some devices continue to work without further maintenance for years. They are still in use and connected to other devices, but do no longer receive security updates (Chapter 8). These devices are not fully bricked but very dangerous. They might run malware without their users taking any notice of it. While users will immediately notice a non-working device, it is hard to determine when a device becomes highly insecure due to missing updates for well-known security issues.

1.2 GOALS, APPROACHES, AND CONTRIBUTIONS

The previous section introduced several problems that this thesis addresses. Users, vendors, governmental institutions, and standardization committees must work together to make IoT devices more secure. Thus, we identify the following four topics that need action:

1. Increasing public security awareness of all involved parties,
2. enhancing vendor support and bugfixes,
3. enabling users to take over control of their devices, and
4. working on regulations, specifications, and standards.

In the following, the goals and approaches of this thesis are outlined. We then describe how we contributed to these goals.

1.2.1 *Public Security Awareness*

Goals and Approaches

Public security awareness is our first goal. Without knowledge, users will not demand increased security, and vendors will not harden their systems and products. While security awareness is rather generic at first sight, it has individual aspects outlined in the following.

Users must demand security and privacy. Only if security and privacy become integral product requirements, they will already be considered by the vendor during the product design phase. However, many users are not aware of what exactly security and privacy in a product imply. As of now, there still is a market for insecure IoT products.

While vendors might not be interested in public disclosure and press releases, this is required to raise public security awareness. Moreover, public disclosure forces vendors to fix their issues within products that are released and actively used. Vendors having similar problems in their products that have not been publicly disclosed might nonetheless choose to fix them. Being in the news for commonly known security issues meanwhile is considered a marketing disaster [Th015; Cox19]. Data leakage due to non-compliance to the GDPR can cost a company up to several millions of € [Bra19].

Ideally, the vendor is responsibly informed about the issues before they become public. Responsible disclosure reduces the risk of attacks being exploited in the wild. When a security issue is discovered by a researcher, it might have been previously known to less responsible parties. Especially popular products are also popular targets and exposed to non-public security analysis. While public disclosure is discussed controversially [Dul19; May19], users who demand security against advanced attackers and regularly update their devices are more secure after responsible disclosure took place.

Contributions

Public security awareness requires outreach to non-scientific media, such as news articles, radio stations, public television, and collaborations with companies. During our research, it was not just us contacting journalists, but even the press who contacted us on their initiative. Their interaction shows that there is a demand for information about the current state of security for the devices we use every day.

TALKS During the work on this thesis, I contributed to 19 presentations at non-scientific venues. The talks "Building and Breaking Wireless Security" (2015), "Doping your Fitbit" (2017), and "Dissecting Broadcom Bluetooth" (2018) were presented on large stages at the Chaos Communication Congress and recorded. Recordings are available on `https://media.ccc.de` and `https://youtube.com`. Combining views of the English recordings in December 2019, the oldest talk on wireless PLS has 77 089 views, the presentation on *Fitbit* fitness trackers has 8066 views, and the *Broadcom* Bluetooth vulnerability demonstration has 7463 views. Topic-wise, "Building and Breaking Wireless Security" is the most fundamental talk relevant to a product's design phase. The view statistics indicate a high interest in wireless and IoT security topics within a more technical community. Based on these talks, I was invited for press interviews and presentations at companies, which enabled me discussing these topics with a broader, less technical audience.

NEWS ARTICLES Our work on *Fitbit* fitness trackers got published on *Hackaday* [McE17], while a local newspaper covered our work on *Neato* and *Vorwerk* vacuum cleaners, as well as a radio podcast [Sch19; Tro19]. Issues in *Broadcom* Bluetooth chips got even more media attention. We gave interviews to *Golem* and *Hackaday* before our talk and got a great news report from a journalist at *dpa* [Wil18; Tre18; Zsc18]. Many newspapers copied the *dpa* press release and translated it into multiple languages. We cannot provide any statistics on view numbers, as they are not listed within the corresponding articles. However, we estimate the audience of news articles, especially if published by *dpa*, to go way beyond the impact of talk recordings and scientific publications.

PUBLIC TELEVISION Public television is very interested in IoT and consumer electronics. *ZDF* made a full feature on fitness trackers, including an interview with me on their security [Wol18]. *MDR* published a short clip, including some vacuum cleaning security information and an interview with me [Huk18]. *Hessenschau* published a story about emergency communication and filmed within our university, also briefly mentioning our vacuum cleaning robots [Hes18].

PRESENTATIONS AT COMPANIES A major bank decided to do *Security Awareness Days* for their staff. I was invited to give two talks and a live demonstration about IoT security, with a non-expert audience and unconcerned IoT users [Cla18b]. Moreover, I gave a talk at an automotive company about a connected IoT device with similar properties as they face during development [Cla17]. While these companies do not provide insights on their internal security processes without Non-Disclosure Agreement (NDA), their invitations show nonetheless that our work is of high relevance to them.

1.2.2 *Vendor Support and Fixes*

Goals and Approaches

Responsible disclosure raises public security awareness, but also leads to the second goal, which is coordination with vendors to provide proper fixes. We consider vendor support and fixes as a separate goal, as public disclosure of a bug does not automatically lead to good software quality. Often, a security researcher's perspective on a product is very different from a developer's perspective. This difference leads to varying patch quality.

Starting at the design phase and until the end of life, it is the vendor's responsibility to patch products. During the design phase, the vendor will potentially follow specifications and regulations but has a lot of freedom. This freedom leads to highly customized and complex ecosystems that are prone to bugs. Thus, typical bugs especially in regarding IoT ecosystem design should become well-known.

Many security problems discovered after release are architectural and originate from the design phase. Specific issues can be fixed quite easily, i.e., by introducing a length check to prevent a buffer overflow as *Neato* did to fix an authentication bypass. However, the exploitability of such bugs depends highly on the overall system architecture. In connected systems, it is vital to review trust assumptions between components. Some interfaces may be required to support substantial product features, while others might not be necessary or can be hardened.

During responsible disclosure, it is essential to highlight the overall architectural issues and, if possible, review changes with the vendor. Vendors might silently close issues or accidentally re-introduce them [New19]. Thus, regular testing after reporting a bug is required. Moreover, vendors who depend on third-party components might not be adequately equipped to perform tests themselves, e.g., one of *Broadcom's* customers claimed they thought they had been patched against *CVE-2018-19860*.

Contributions

In the following, we cover how we decided which products to test for security, how we informed the vendors, and how we still stay in contact with them.

INDEPENDENT SECURITY RESEARCH We select mature devices that were not yet publicly analyzed for security and lack a bug bounty program. While this does not mean that such products are not tested internally, there is little or no open knowledge about these products. In contrast to most security analysts, we are independent of the companies whose products we test. We can spend as much time on a product as we think is necessary. Even if our analysis would not return any vulnerabilities, our results enable users to make a more informed decision when purchasing a product.

Our pre-selection enables us to spot widely used IoT devices, using recent and multifaceted technologies. This selection for testing is very different from vendors actively considering testing their products. The usual approach is that they hire an external company to test specific interfaces and provide them with documentation. Some of their components might also go through a certification process. We appreciate that vendors do this kind of testing. Yet, some vendors choose not to test their products and save money in the short run.

Most of the time, security testing and improvements are kept private. For example, we saw that *Broadcom* added security measures to their Bluetooth chips, despite there being only a few documented incidents on their Wi-Fi chips. Our research pushed them to add even more security to their newer chips.

Some vendors have a bug bounty program but exclude interesting components within their ecosystem. *Fitbit* has a bug bounty program, but they exclude fitness tracker firmware vulnerabilities from it. Their newer smartwatches became part of their bug bounty program later. In their initial threat model, they never considered the owners of trackers to have malicious intents, such as spoofing a step count.

Within the work on this thesis and the Bachelor and Master thesis supervision, we identified six Common Vulnerabilities and Exposures (CVEs) in *Broadcom* and *Cypress* Bluetooth chips (Chapter 8), four CVEs on *Neato* and *Vorwerk* vacuum cleaning robots (Chapter 4), and one issue in the QNX operating system running on those robots [Ege19]. We reported one incident that affected critical infrastructure across Europe [Pla18]. Moreover, we identified various issues in Bluetooth finders of that one got assigned a CVE (Chapter 5), and were acknowledged by *Fitbit* for our work on their fitness trackers (Chapter 3). The *iOS* 13 security updates additionally recognized us for our work on Bluetooth [App19a], and our *Broadcom* related CVEs appear in the *Google* and *Samsung* security advisories [And19a]. Responsible disclosure with *Apple* on further security issues is still ongoing, as well as the responsible disclosure with three vendors of intercom door opening systems. These issues would likely persist without our research and possibly be sold on the black market.

RESPONSIBLE DISCLOSURE IMPROVEMENTS AND BENEFITS Since our selection does not include asking or informing the vendor that we are performing tests in advance, responsible disclosure is not straightforward. Especially for smaller companies, responsible disclosure was new, and they often did not have a security contact and process for this. Despite these struggles during initial contact, we do have personal contacts at *Neato*, *Cypress*, and *Broadcom* by now, with short response times.

Along with contacting the vendors, we also provide them a timeline of when we aim to publish our findings. This timeline allows them a reasonable time to fix vulnerabilities but also pushes them to move forward. In our papers and within this thesis, we also publish bugfix and disclosure timelines. These timelines are valuable for other researchers who want to disclosure bugs to similar companies and to find reasonable deadlines in disclosure processes. Anguelkov also published a responsible disclosure timeline with *Broadcom* [Ang19]. Similar to our experience, replies by *Broadcom* and their customer *Apple* took a long time, and we both were asked by *Broadcom* if we would be willing to sign an NDA. In contrast to Anguelkov, we directly communicated the venues on that we are going to present the vulnerabilities and did not follow any requests to extend these deadlines.

VENDOR RELATIONS Within our work, security awareness at the vendors increased. For example, the new *Fitbit* smartwatches have a different security architecture, and except compatibility features, none of the issues the trackers had applied [Gr019]. Also, *Neato* is considering a redesign of their security architecture and adding new protocols to their ecosystem. Overall, reactions by the vendors were very positive, and their customers will benefit from the increased product security.

Neato invited us to visit their campus while we were around for presenting our paper about their ecosystems. Our security contact from *Fitbit* visited Paul Patras while he was in Europe, and we met them at *DEF CON 27* in Las Vegas. Detecting vulnerabilities in *Broadcom* chips got us in contact with *Apple*, *Cypress*, *Google*, and *Samsung*. We additionally contacted them because they use *Broadcom* chips and have affected products that are still receiving security updates. Moreover, we have a local contact at *Cypress* in Langen.

1.2.3 *User Support and Fixes*

Goals and Approaches

Another aspect of security awareness leads to the third goal, which is user support. Users need the possibility to act and make decisions. Solely informing them that the product they are currently using is insecure does not provide them with any solution.

From an economic perspective, users can primarily act by avoiding specific products. To support the users in decision making, press releases about security issues and privacy leaks enable them to compare products by incidents. Products that have been under public analysis and review will usually be improved afterward. While this is no guarantee, users can still make decisions on the hardware capabilities, i.e., if they buy a vacuum cleaning robot with or without a camera module. Besides, it might also make a difference to users in which country their private data is hosted. Since IoT products typically require support from the vendor, users can also decide on past software updates and security fixes to estimate if the vendor will provide good support in the future. The absence of updates and security patches is usually a bad sign because it does not imply that a product never had any flaws.

Surveys show that users demand security and privacy in a product when they are given the choice. They are even willing to pay a limited amount for this. However, the actual costs to ensure features like encrypted storage and certifications can be higher for the vendor [MWZH17].

While monetary aspects are important for newly released products, users might want to customize devices and keep them over a long time. Missing features and support can especially become relevant in later stages of a product's life cycle, especially at the end of life. Users can customize and re-purpose devices themselves if provided with proper tools. However, most IoT ecosystems are closed-source with undefined Application Programming Interfaces (APIs). As researchers, we can open up these systems and provide alternate ecosystems. Advanced users might decide to flash an unofficial research firmware if this helps them to keep their data private instead of sharing it to the vendor. While such tools and their open-source releases require additional efforts that go beyond bug hunting in products, they are of high value for the community. Off-the-shelf devices might even be re-purposed for research.

Contributions

Opening up closed-source firmware is an integral part of providing control over devices to their users. We integrate *Nexmon* [Sch18], a binary patching framework, into two very different types of devices: *Fitbit* fitness trackers (Chapter 7), and *Broadcom* and *Cypress* Bluetooth chips (Chapter 8). Since both of them are Advanced RISC Machine (ARM) based, most parts of *Nexmon* can be used as they are.

While we had access to the firmware of various IoT devices throughout our work, we decided to patch these two. *Fitbit* fitness trackers have a large user base and are embedded into a complex IoT ecosystem; thus, we considered it valuable to make them cloud-independent and their sensors accessible. Back in 2017, *Fitbit* was amongst the few vendors who applied end-to-end encryption on fitness data [Fer+17b], which made reverse-engineering their technologies interesting. The Bluetooth firmware is part of hundreds of millions of devices, and our main motivation was to make it modifiable for other researchers due to the lack of proper Bluetooth analysis tools.

OPEN SOURCE FITBIT SMARTPHONE APP AND FIRMWARE First of all, we use *Nexmon* for the *Fitbit Flex* and *Fitbit Charge HR* fitness trackers. This firmware is overall more high-level but lacks symbols. With binary patching, we can change step counts on both devices, while keeping the remaining firmware functional. Moreover, on the *Fitbit Flex*, we can re-enable debugging at runtime and readout raw accelerometer data. We also build an *Android* app to flash pre-built firmwares and interact with our new features.

We publish the firmware patching frameworks along with exemplary patches. Programmers without reverse engineering knowledge can extend these patches in C with *Nexmon*. For example, the *Fitbit* accelerometer patch currently only copies readings to an output buffer, but the readings could also be interpreted locally to recognize gestures.

BLUETOOTH FIRMWARE RESEARCH PLATFORM Second, we integrate *Nexmon* for various *Broadcom* Bluetooth chips. Patches can be either generated to be executed at runtime with *InternalBlue* or as operating system level `.hcd` and `.bin` files loaded during driver initialization. Both patching formats are different from rewriting a complete binary blob; thus, adjustments to the *Nexmon* patching framework are required.

For the Bluetooth firmware, we provide patches that enable users to test their devices against publicly known attacks against pairing [ATR19; Eli18]. Thus, users can now check for themselves if their car or smartwatch uses a secure and up-to-date pairing implementation. Moreover, we implement a patch against a vulnerability we found on a device that is no longer receiving updates, as well as a Media Access Control (MAC) address filter that protects from more generic attacks.

1.2.4 *Regulations, Specifications, and Standardization Improvements*

Goals and Approaches

The last goal is to go beyond specific products. Regulations, specifications, and standardization can achieve this. Any findings and proposals on this level apply to the early design phase. Thus, research in this area has a high potential to make contributions that will later be integrated into a variety of products.

Many issues arise because the underlying specifications and standards make false security assumptions. Vendors will follow these specifications, meaning that any vulnerability on this level is likely to affect the entire device base utilizing the standard in question. The time it takes from specification to actual products allows testing upcoming technologies before they get widely applied.

Furthermore, governmental regulations can force vendors to provide a certain level of security and privacy. Before bringing a product to the market, vendors need to provide statements about compliance with those, and sometimes even undergo additional testing. However, due to the sheer quantity, IoT devices cannot be tested in detail. While regulations provide a legal framework to claim what is wrong with a product, they do not replace public security research.

Contributions

Most solutions for short-range IoT connections operate in the 2.4 GHz Industrial, Scientific and Medical (ISM) band, such as Wi-Fi, Bluetooth, and Zigbee. However, technologies that are based on different PHYs enable new communication paradigms and possibilities. For example, mmWave communication in the unlicensed 60 GHz band offers high throughput that can replace cables. Moreover, VLC uses ambient light sources and, thus, is an ideal candidate for IoT smart home applications. We take a deeper look into those upcoming technologies and analyze claims on their security properties.

SECURITY AGAINST EAVESDROPPING With the first VLC and mmWave development kits available, it is possible to test these upcoming technologies. One of the most common claims is that their range is much easier to control, thereby making eavesdropping harder [Pov11; Fre13; Yan+15]. We experimentally show that eavesdropping is still possible despite the different propagation characteristics (Chapter 9).

WIRELESS PHYSICAL-LAYER SECURITY ASSUMPTIONS Furthermore, the first wireless PLS approaches have been transferred to the domain of VLC [LMGGB15; ML14;

Zha+14]. We show that these approaches are as broken as in existing Radio Frequency (RF) and Wi-Fi implementations, and sometimes even worse, because visible light does not contain phase information (Chapter 10).

1.3 OUTLINE

This thesis is makes contributions within the whole product life cycle, and thus, is also structured by this life cycle. In Part ii, we cover IoT products and their ecosystems after their release. Specifically, we analyze *Fitbit* fitness trackers (Chapter 3), *Neato* and *Vorwerk* vacuum cleaners (Chapter 4), and various Bluetooth finders (Chapter 5). When a product reaches the end of life and the vendor stops support, the only possibility to modify it is binary patching, which we cover in Part iii. After explaining the concept of binary patching (Chapter 6), we change the firmware of *Fitbit* fitness trackers for cloud independence (Chapter 7), and enable monitoring and injection capabilities on *Broadcom* Bluetooth chips (Chapter 8). Finally, Part iv provides an outlook on how future wireless standards can be used to design secure wireless connections. We experimentally evaluate if it is hard to eavesdrop on VLC connections (Chapter 9), and then expand the scope to PLS for VLC applications in general (Chapter 10). Results are discussed in Part v, where we also conclude this work.

Part II

PRACTICAL IOT SECURITY

In this part, existing IoT ecosystems are analyzed regarding security. A general overview of those ecosystems is provided in Chapter 2. Practical studies include *Fitbit* fitness trackers (Chapter 3), *Neato* and *Vorwerk* vacuum cleaners (Chapter 4), and a large selection of Bluetooth finders (Chapter 5). Despite finding very diverse implementations, all ecosystems share common concepts. Findings are generalized, and applicability to all kinds of distributed systems is shown.

2 IOT ECOSYSTEMS

This chapter covers structural and technical similarities throughout different real-world Internet of Things (IoT) ecosystems. Section 2.1 motivates our approach to analyze existing systems instead of designing new and perfect solutions from scratch. In Section 2.2, common concepts of all IoT ecosystems are introduced. Sections 2.3 to 2.5 show where these concepts exist in real-world IoT ecosystems. A detailed analysis of these systems, as outlined in Section 2.6, follows in Chapters 3 to 5.

2.1 MOTIVATION

IoT devices have become far more than just fancy gadgets. They are part of our everyday lives. In the past, the IoT component of devices has been optional, i.e., a smart fridge with a built-in camera would still work without the Internet. Nowadays, more and more IoT devices on the market must always be online. *Fitbit* fitness trackers can store data for up to a week but need to be synchronized with the cloud before the data is visible to the user. *Neato* and *Vorwerk* vacuum cleaners work quite well without the cloud, but some features require a cloud connection. For example, the user can draw no-go lines on the plan view of her apartment using the smartphone app, as an alternative to installing magnetic stripes. Bluetooth finders with crowd search functionality require the user to be always online and connected to the finder, a simple empty smartphone battery would mark a finder as lost.

The requirement of an Internet connection to make devices smart comes with two significant drawbacks. First, the device is no longer functional if an IoT vendor goes bankrupt, decides to no longer support an old device, has server or network infrastructure issues, or legal requirements such as the EU General Data Protection Regulation (GDPR) no longer allow operation. Second, user privacy is at risk because data is uploaded to the cloud. For example, fitness tracker users trust the vendor not to abuse their health data. Vendors who sell this data to third parties or even give incentives to their users to share their data with other services are not the only risk. Security issues in the product and cloud implementation often lead to data breaches in practice.

The first issue, vendor infrastructure dependency, raises the question if smart devices can be built in a cloud-minimal or even cloud-independent fashion. If this requirement is part of the system design, this is possible, as we have shown with a reference design for Bluetooth finders [Wel+19]. Removing the cloud from a system becomes more challenging for products where the vendor did not have this in mind. Cloud-independence often requires binary patching of the device firmware and modifying or rewriting the smartphone app. Such techniques are related to IoT ecosystems, and the unique methods they require will be covered in more detail in Part iii, with *Fitbit* fitness trackers being one example.

The second issue, user privacy and security, is also a major concern. Trusting the vendor not to leak customer data is something that can be covered by the law. Moreover, this is a decision the user might still have in mind when choosing between vendors of a similar device. In contrast, security is often advertised but hard to confirm for the user. If a device is very low-cost, this hints to the user's data being sold or that there was no money spent on the device's security. However, there is no guarantee that this is not the case for a high-end product.

Analyzing real-world systems shows restrictions during development and deployment. Results represent common practice in the industry. Low-cost products have more restrictions than high-cost products. Security in IoT systems ranges from basic user authentication in the cloud to hardened hardware. Underlying problems can be found across all classes of devices.

In the following, after explaining basic IoT ecosystem concepts, we compare the design of three real-world systems. Bluetooth finders are the most low-end; some of them are available for just 2€. *Fitbit* fitness trackers are in the range of 50€–100€. The most sophisticated devices are vacuum cleaning robots. Even though some connected robots are as cheap as 250€, we decided to analyze the high-end *Neato* and *Vorwerk* robots, which are in the range of 600€–950 €.

2.2 ECOSYSTEM CONCEPTS

Figure 2 depicts a generic IoT ecosystem overview. Data can be transferred using various technologies, as explained in Section 2.2.1. Data from the IoT gadget can either be passively relayed by an access point or forwarding smartphone app to a server, or the gadget can actively communicate with a smartphone app. We categorize the first as a *transparent* gateway (Section 2.2.2), and the second as a *data-modifying* gateway (Section 2.2.3).

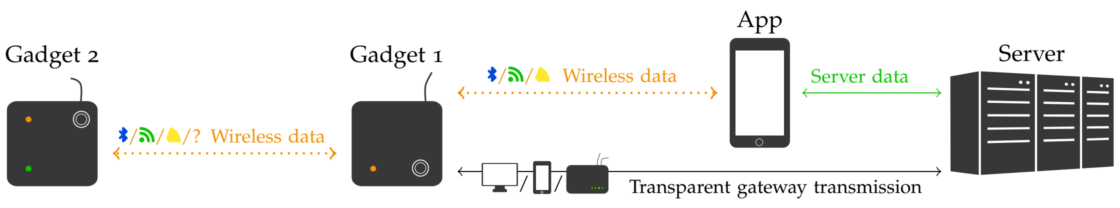


Figure 2: Generic ecosystem overview.

2.2.1 *Wireless Data Transfer*

There are plenty of wireless standards and sub-protocols present in IoT products. Most gadgets use technologies available on off-the-shelf smartphones: *Fitbit* fitness trackers use Bluetooth (ᛒ), *Neato* vacuum cleaners use Wi-Fi (📶), and *Nello* door openers use Visible Light Communication (VLC) (🔔) with the smartphone's flashlight [Ull18a].

Note that gadgets might also communicate using a separate network using a different protocol, only supported by the gateway. This can be seen in many smart home appliances. Lighting systems, such as *IKEA TRÅDFRI*, use Zigbee Light Link (ZLL) on 2.4 GHz to communicate between components. Some vendors even develop their wireless physical layer protocol. A setup with a protocol not supported by smartphones requires all gadgets to communicate with the gateway. Setups with a unique wireless gateway do not necessarily require a remote server. The *IKEA TRÅDFRI* gateway supports direct communication with Wi-Fi to the smartphone app and can optionally connect to a remote server via Ethernet.

2.2.2 *Transparent Gateway*

A transparent gateway is required when data needs to be transferred over the Internet, and information is not modified on its way. An IoT device might not have the capability to connect directly to the Internet if it is using wireless protocols such as Bluetooth. Due to energy constraints, limiting IoT device connectivity to Bluetooth Low Energy (BLE) is a common practice. In such a scenario, a Wi-Fi access point or smartphone can be a transparent gateway. IoT devices with built-in Long-Term Evolution (LTE) modules (📶) bring their own transparent gateway. A user will not be aware of data passing transparent gateways. By design, transparent gateway transmission requires either a separate server data channel or an additional wireless data channel to display data to the user.

In the *Fitbit* ecosystem, the smartphone app or a *Windows* program connects via Bluetooth to the tracker, collects its data, and forwards it to the server. The server data to the app contains fitness records for the user. A minimal subset of data can also be retrieved over the wireless data channel in a so-called live mode. *Neato* has fewer power constraints, and their robots directly connect to the servers using Wi-Fi. Only initial configuration and a very restricted manual drive mode are available over the local wireless data connection. Afterward, even if the user's smartphone and the robot are within the same Wi-Fi, command and status messages are always forwarded to and processed by an external server.

2.2.3 *Data-Modifying Gateway*

A data-modifying gateway takes information from the IoT device and adds new information before forwarding it to the server. This additional information is required by the ecosystem to provide a functional service, just taking the data extracted from the IoT device is not sufficient.

This kind of setup is required for Bluetooth finders: Identification information is on the finder itself, but only the smartphone app can add Global Positioning System (GPS) information to this. Devices such as the Bluetooth finder can also be implemented without a cloud connection if the gadget is not encrypting or signing its data. The smartphone app can store the last GPS position data and inform the user where the finder has been seen last. However, this restricts the user to use her smartphone to find the last location of an item. Moreover, crowd search by multiple users for a lost item does not work without cloud connectivity.

In general, data-modifying gateways are very interesting from an attacker perspective. They add information that will be interpreted and stored on the cloud component of an IoT ecosystem.

2.3 FITBIT FITNESS TRACKERS

The *Fitbit* fitness tracker ecosystem shown in Figure 3 is the most similar to the generic system.

Wireless data between the app and *Fitbit* is exchanged using Bluetooth. Security is based on a symmetric end-to-end encryption key, which is unique for each device and setup in the factory. Any *Fitbit*↔server message that leaves either the server or the *Fitbit* is encrypted with this key. Two **transparent gateway** implementations exist: *Fitbits* can be synchronized using a smartphone app or a *Windows* program. A user typically only uploads her own *Fitbit's* messages to the server, but in practice, the server accepts any *Fitbit* message without additional user authentication.

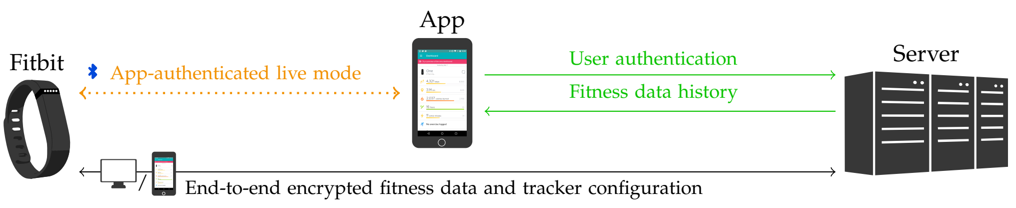

App
Fitbit
App-authenticated live mode
User authentication
Fitness data history
Server
/
End-to-end encrypted fitness data and tracker configuration

Figure 3: *Fitbit* fitness tracker ecosystem overview.

During initial pairing with a *Fitbit*, the user proves ownership to the server. A user can retrieve all fitness data from the server by using the app or logging in to the *Fitbit* website. Even for tracker models with a display, connecting to the server is the only supported way to get sleeping statistics, detailed step counts, and heart rate history.

For live monitoring of the heart rate, the transparent gateway requires requesting the current fitness date from the *Fitbit*, sending data over the app to the server, decrypting it, saving it, issuing another request from the app to get the heart rate and finally displaying it to the user. Since this would be a lot of overhead, *Fitbit* decided to implement a so-called *live mode*. It requires an authentication token issued by the server. With this token, the app can request live data, which includes the step summary of the day and current heart rate.

It is essential to add that the new *Fitbit* smartwatch series introduces a lot of components to this infrastructure [Gro19]. The *Fitbit Ionic* smartwatch has Bluetooth, Wi-Fi, and Near-Field Communication (NFC). While NFC is only used for payments, Wi-Fi adds a whole new smartwatch ecosystem. With Wi-Fi, apps can be downloaded directly from the *Fitbit* servers that run as JavaScript on the smartwatch. The *Fitbit* smartphone app runs a separate companion app that communicates with the smartwatch. Nonetheless, the Bluetooth interface is still used to transmit fitness data and also features a live mode.

2.4 NEATO AND VORWERK VACUUM CLEANERS

The *Neato* and *Vorwerk* ecosystem depicted in Figure 4 shares many concepts with the *Fitbit* ecosystem. The two significant structural differences are that (1) the robot has a Wi-Fi interface to communicate directly with servers and (2) there are two separate cloud infrastructures for account information and robot control.

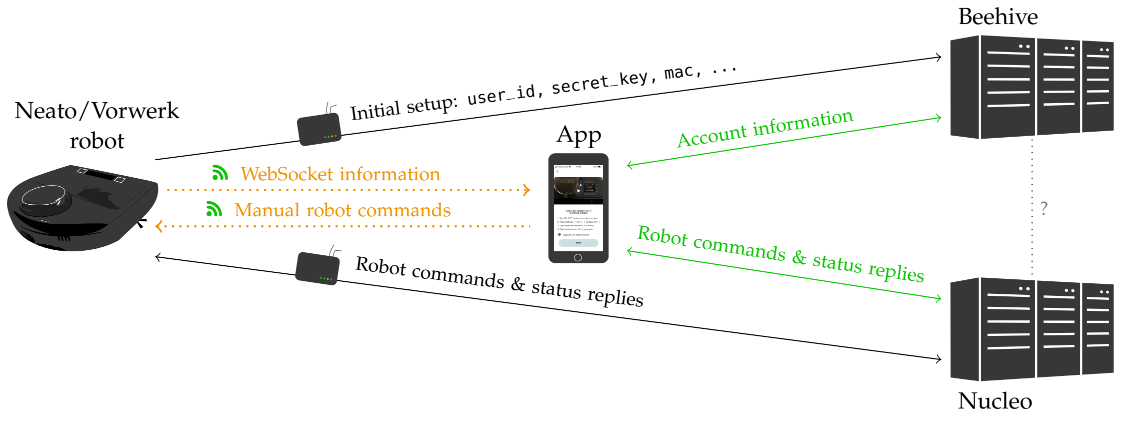

Beehive
Initial setup: user_id, secret_key, mac, ...
Neato/Vorwerk robot
App
Account information
WebSocket information
Manual robot commands
?
Robot commands & status replies
Robot commands & status replies
Nucleo

Figure 4: *Neato* and *Vorwerk* vacuum cleaning robots ecosystem overview.

During the initial setup, the robot spawns a separate Wi-Fi to which the user connects. The user creates a new account. The robot sends a list of visible Wi-Fis to the app, and the user selects one and enters the Wi-Fi password. The app sends the user's user_id and Wi-Fi configuration to the robot. From this point on, the Wi-Fi access point is used as **transparent gateway**. The robot generates a fresh secret_key and then initiates a connection with the *Beehive* cloud. After this setup, the robot is linked to the user's account.

Robot commands can be sent from the app via the *Nucleo* cloud transparently over the Wi-Fi access point to the robot. Commands include starting a new cleaning cycle, configuring cleaning schedules, and setting no-go lines on an apartment map.

Similar to the *Fitbit* live mode, robots have an authenticated *manual drive mode* without any server involved. This mode is very restricted and allows users to set the direction of the robot manually and to start and stop the motor. Except from demos and tests, this mode is not useful.

2.5 BLUETOOTH LOCATION FINDERS

Figure 5 shows a generalized Bluetooth finder architecture that can be found across multiple vendors, i.e., the market leaders *Nut* and *Tile*.

Bluetooth finders help to locate lost items. They require a permanent Bluetooth connection to a smartphone and alarm the owner when the connection is lost. An application for this is not to leave the house without a purse. Moreover, the last position with connectivity can be stored in the app or on the server. If the item is able to move or get moved, it might no longer be at the last known location. In this case, the owner can mark an item as lost and ask other users with the app to search for the finder. Other users will then report the item's position to the server once found.

Bluetooth finders are very cheap devices. Their owner will typically buy multiple finders and attach them to all essential items. In contrast to Fitness trackers and vacuum cleaners, they are even more battery-constrained. Ideally, they can be powered with a button cell battery for more than a year. Thus, they cannot rely on an on-board GPS module. Instead, the smartphone adds its GPS position to the finder's information,

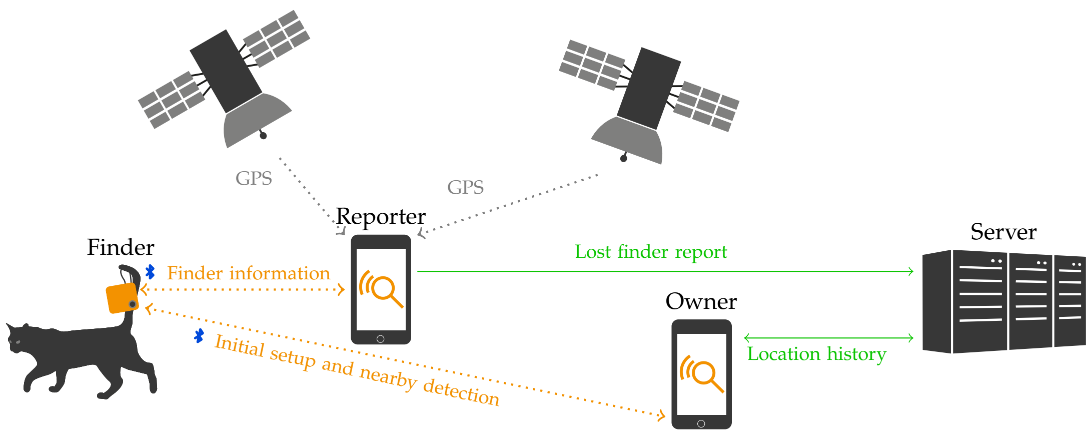


Figure 5: Bluetooth finder ecosystem overview.

thereby being a **data-modifying gateway**. The only wireless data exchanged over BLE usually is a unique identifier. Depending on the concrete implementation, this identifier can be additionally secured or bound to a user account.

2.6 ANALYSIS OF PRACTICAL SYSTEMS

Apart from the common concepts shown in this chapter, IoT ecosystems are very diverse. They need to be tested individually to answer if user data is appropriately processed. While testing can provide impressive results, it needs to be done carefully. Otherwise, as we look into production systems, data of actual users could leak, and the infrastructure could experience downtimes.

For an initial analysis, a Machine-in-the-Middle (MITM) attack can be launched on the various gateways, if not prevented by proper encryption. Even with encryption enabled, data can often be ex-filtrated from the IoT device itself. Results from such initial mostly passive analysis already give a clear picture of the infrastructure similar to Figures 3–5.

What follows is an in-depth analysis of the underlying protocols and logic. The remaining chapters of this part will cover this for the *Fitbit* fitness trackers, *Neato* and *Vorwerk* vacuum cleaners, and various Bluetooth finders.

MY CONTRIBUTION

I completely wrote this chapter clarifying IoT ecosystem structures.

Figure 4 is based on Figure 1 from [Ull+19], Figure 5 is based on Figure 1 from [Wel+19]. Both were adapted to represent information flow better.

3 FITBIT FITNESS TRACKERS

We demonstrate why looking into the *Fitbit* ecosystem offers exciting insights in Section 3.1. Possible attack scenarios based on these vulnerabilities are discussed in Section 3.2. Client-side analysis of the tracker with a focus on changing its firmware behavior is described in Section 3.3, as my main focus within this work was re-implementation of the *Fitbit* firmware update process. The actual means of binary patching to modify the firmware will be shown later in Chapter 7. We refer to related work in Section 3.4. Section 3.5 provides insights to the responsible disclosure process.

3.1 MOTIVATION

When we looked into *Fitbit* in 2017, *Apple*, *Xiaomi*, and *Fitbit* dominated the wearables market [IDC17]. In 2019, *Fitbit* has sold more than 89 million devices since their inception [Fit19b].

We decided to analyze *Fitbit* in more detail because they turned out to be secure at first sight in a study comparing 17 different fitness trackers [Fer+17b]. While *Xiaomi* was still sending plaintext data over Bluetooth in 2017, *Fitbit* already had a competitively secure product on the market.

Compared to a smartwatch, such as the *Apple Watch*, a fitness tracker is smaller, cheaper, and thus more constrained in hardware and software. It is interesting to analyze what is doable in such a limited product. *Fitbit* employs a rather complex ecosystem with exchangeable components. That means that they support different tracker models, and then synchronize either using a smartphone app or a *Windows* desktop program.

While *Fitbit* has a public bug bounty program, they exclude the trackers from it. They only consider the server components to be critical. Incidents they experienced before 2017 were mainly data leaks on their cloud components. Thus, we assumed that the hardware might be less secure, which turned out to be accurate, as shown in the following sections. Initially, they never considered the user of a fitness tracker to have cheating

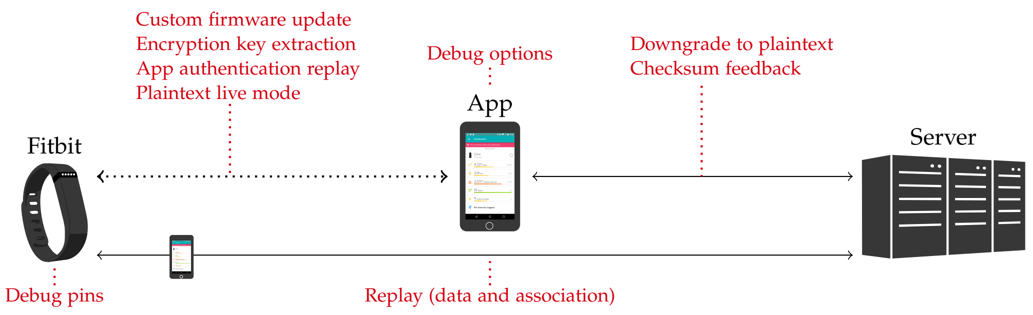


Figure 6: *Fitbit* vulnerability analysis results.

incentives. With social network add-ons and health insurance discounts, attacker incentives changed, but their security model stayed the same.

Figure 6 shows an overview of all vulnerabilities found throughout our analysis. They affect all sorts of components of the system, starting from the firmware on the tracker itself, going to the smartphone app, and even reaching to all kinds of communication between the ecosystem's components.

3.2 ATTACK SCENARIOS

We categorize adversary goals into **(1) spying** targeting a victim's privacy, **(2) financial interests** such as ransomware or monetary rewards, **(3) denial of service** affecting service availability any of the parties involved, and **(4) cloud independence** from *Fitbit*'s official services. We also discuss **(5) non-adversarial applications** such as the exploitation of fitness data for medical research. The following paragraphs describe the concrete attack scenarios that can be achieved based on our findings, as well as their implications to both users and the *Fitbit* business model.

(1) SPYING: The spectrum of spying attackers interested in fitness data is broad since the information that trackers collect could be exploited for private, political, or financial motives. Attack scenarios range from stalking to health analysis and user profiling.

- *Sniffing Ongoing Communication* — Spies can sniff ongoing traffic during routine tracker and app interaction. Local wireless Bluetooth Low Energy (BLE) communication is plaintext, and only selected data types transmitted over BLE are encrypted end-to-end. Remote Transport Layer Security (TLS) connections can be tampered with if any part of the network connection used is under an attacker's control. We observe that the smartphone app does not employ certificate pinning, and the login credentials transferred over TLS connections are not further secured. Usage of non-hashed login credentials enables later re-using intercepted credentials for espionage purposes.
- *Proximate Espionage* — *Fitbit* does not currently recognize proximate wireless espionage, though it is possible to infer, for instance, if someone was jogging outside. We find that eavesdroppers can connect to any tracker in wireless range and request stored data. Typically, this data is end-to-end encrypted, but encrypted dumps contain metadata revealing the number of synchronizations performed, user activity levels, and the tracker's serial number. In many scenarios, a spy can issue authenticated commands to obtain information in plaintext live mode, or sniff live mode data currently being exchanged with the victim. We further observe that some trackers allow full reading of their memory.
- *Remote Espionage* — Spies can obtain all locally encrypted data that trackers store by associating them to a controlled *Fitbit* account and then triggering synchronization. The server then decrypts user activity dumps while the attacker can be located virtually anywhere on the Internet. In the case of trackers in plaintext mode, the association is only limited to knowing a valid serial number. Trackers that implement encryption must initially be either within wireless or physical range (depending on the model) for an attacker to capture and replay an association.

The victim can re-associate her tracker to a personal account, limiting the espionage period between the last legitimate data synchronization and re-association. Full-featured, persistent espionage can be mounted by configuring an alternative server address inside a victim's smartphone app. This is a more sophisticated Machine-in-the-Middle (MITM) attack that does not require any control over the victim's network.

- *Spyware* — Modified firmware can also enable full-featured, persistent espionage on trackers that have been in the attacker's wireless range for minutes. Indeed, we find that spies can flash malware without the victim's interaction, compromising critical security and privacy functions. Trackers that work in plaintext mode accept plaintext firmware updates, which are neither signed nor device-specific. For encrypted trackers, the spy requires a valid device-specific encryption key, which we have been able to extract wirelessly on *Fitbit Flex* and *One*.

(2) FINANCIAL INTERESTS: Next, we consider attacks that can directly generate monetary value.

- *Selling Fitness Data* — Attackers can sell activity records that falsely certify "healthy" lifestyles, as third parties, including insurance companies, offer rewards to users who prove physically fit by sharing *Fitbit* trackers data [PwC16]. Flashing a firmware that randomly multiplies all step counts or reports a healthier heart rate produces enhanced fitness records. Consumers interested in fake fitness services would allow the attacker to analyze or tear-down their tracker to extract all tracker-specific information and automatically generate plausible activity records.
- *Ransomware* — Victims can be extorted to have a compromised tracker working again by exploiting one of the following vulnerabilities. While in wireless range, an attacker can disrupt tracker functionality via an unauthenticated "set date" command, which can make fitness records appear erratic and potentially alert an employer of unusual employee behavior. Likewise, we find that an attacker can set alarms with prior authentication to wake up the victim at arbitrary times. Repeated remote association with the attacker's account stops the victim from collecting any activity summaries. Similar to spyware, ransomware that disables official *Fitbit* firmware updates and limits tracker functionality can be flashed without user consent.

(3) DENIAL OF SERVICE: Attacks resulting in Denial of Service (DoS) not only frustrate users but also harm *Fitbit*'s reputation, if such attacks can develop at scale.

- *Local Wireless Commands* — Beyond ransomware, dates and alarms on trackers can be manipulated to deny service to *Fitbit* end-users. For instance, an attacker can exploit authentication replay and local commands to turn a reliable fitness tracker into a vexing random alarm clock.
- *False Firmware Updates* — A malicious firmware update can be used to permanently make trackers non-working by removing the wireless flashing option and other

useful functions. It is also possible to harm the *Fitbit* cloud by flashing mischievous firmware that sends randomized tracker data to the backend servers, which might be hard to filter automatically.

- *Virtually Stealing Trackers* — Vulnerabilities we identify concerning the remote association procedure can be leveraged to "steal" trackers from the currently associated accounts, thereby disabling synchronization until the victim performs re-association. Such attacks can be performed at scale, since an attacker can re-associate a tracker multiple times using previously recorded traces, as long as the tracker is temporally within wireless range. Besides, association information for plaintext trackers can be deduced from the original packing, or attackers could bruteforce plaintext associations.

(4) CLOUD INDEPENDENCE: Making a device independent of the *Fitbit* cloud can be regarded as an attack on the brand's business model since users would no longer synchronize their data to the official servers.

- *Full-Featured Separate Infrastructure* — Currently, *Fitbit* users are constrained to upload their data to the official cloud to be able to monitor their activity. This is not intrinsic to the *Fitbit* business model; it is a general practice in the fitness tracker industry [Fer+17b]. A custom firmware could feature different encryption functions and keys, hence enabling the user to synchronize data with other custom services different from *Fitbit*. We also observe that the official *Android* smartphone app has a hidden option that can allow configuring alternative servers. In such a scenario, the look and feel of the app will not change, while older trackers could be migrated to solutions that provide superior encryption solutions. Users could also run such servers themselves or allow someone they trust to do so.
- *Cloud Independence with Reduced Function Set* — Authentication credentials, which are valid for an unlimited time, enable an attacker to issue a subset of commands, including live mode operation. This can be used to extract daily activity summaries and average heart rates, and redirect such statistics to an independent cloud while preserving some interaction with the *Fitbit* cloud. Users of the newest tracker models, which are not yet wholly reverse engineered, could still prefer this reduced feature cloud independence over synchronizing all their data with *Fitbit*.

(5) NON-ADVERSARIAL APPLICATIONS: Finally, we consider scenarios where there is not necessarily a malicious intent or a personal incentive.

- *Medical Research Sensors* — Trackers have an interesting selection of sensors, ranging from unsophisticated accelerometers to fancier heart rate monitors and GPS receivers. Activity records contain only *interpreted* sensor data, for example, the accelerometer only saves aggregated data that includes step counts and floors climbed. While medical research is not an attack vector *per se*, it can benefit from inexpensive platforms that can record and transmit raw personal sensor data. In the case of fitness trackers this, could be achieved using firmware modifications. Raw tracker sensor data is more verbose than the output of the public sensor interface of even the most recent *Fitbit* Ionic smartwatch, which for example only provides the overall heart rate but not the pulse curve [Fit19a].

- *IoT Security Research* — Given *Fitbit*'s market lead and the reputation of their devices for being relatively secure, we expect other industry players and academic researchers have incentives to continue investigating their security and privacy properties. On the one hand, competitors could learn from *Fitbit*'s mistakes (if identified) to strengthen the robustness of their products from initial designs and enlarge the customer base. On the other hand, researchers may seek to discover vulnerabilities that can be exploited in different gadgets and expand the body of knowledge in the area of Internet of Things (IoT).

3.3 FIRMWARE UPDATE PROCESS

Fitbit trackers implement most logic internally. They already compute step counts and sleep states from the accelerometer locally. Fitness trackers are required to do this by design—transferring raw accelerometer data over the air would not be energy-efficient. Getting control over the firmware, therefore, enables to open up the whole system and understanding major parts of the protocols.

The encrypted communication between server and tracker needs to be reversed to extract and install arbitrary firmware. Back in 2017, when initially taking a look at the *Fitbit Flex* fitness trackers, they had a security bug that allowed attackers to extract the encryption key over the air. We briefly describe this bug in Section 3.3.1. The remaining part explained in Section 3.3.2 is about the firmware update process itself.

3.3.1 *Understanding Encryption*

The encryption key can be extracted by using an authenticated command for memory extraction. Authenticated commands are a subset of commands the app can issue directly without contacting the server. Authenticated memory readout was found by *Schellevis et al.* when comparing updates for the *Fitbit* Charge HR [Sch+16].

Steps required to issue authenticated commands are shown in Figure 7. The user (1) logs into the server, (2) connects to her *Fitbit*, and (3) during setup confirms to own the tracker by tapping it physically. Trackers with a display require the user to enter a number from the screen instead of tapping. We call this step *association* since it initiates the link between a *Fitbit* and a user account. Only one user account can be associated with a *Fitbit*, linking a new account will remove it from the previous account. After association, (4) any app, including third-party apps, can request authentication credentials. These

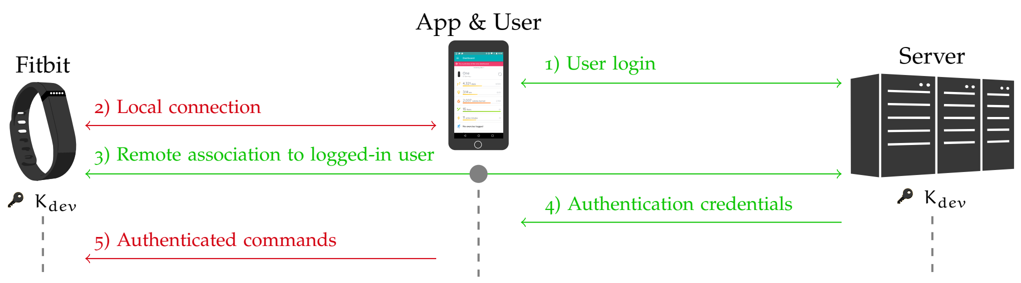

App & User
Fitbit
Server
1) User login
2) Local connection
3) Remote association to logged-in user
4) Authentication credentials
5) Authenticated commands
K_{dev}
K_{dev}

Figure 7: Authentication model: local wireless connection vs. association of a tracker to a user account. Local BLE commands (marked in red) are not cryptographically secured.

credentials are stored in the app and reusable forever since they are only bound to the device-specific encryption key 🔑 K_{dev}. Anyone with authentication credentials for a particular *Fitbit* can run authenticated commands on it forever.

Authenticated live mode is partially by design, as *Fitbit* supports a serverless live mode for current statistics. Usually, fitness records would be sent encrypted with 🔑 K_{dev} to the server, which decrypts and stores them. The app can then request this data over an encrypted connection for the currently logged in user. This approach would not scale for displaying current data, such as the heart rate, and therefore, the live mode was invented. Since the live mode is there to reduce traffic to and load on the server, making authentication credentials reusable is intended.

The non-intended use case, however, is to issue other authenticated commands. On older firmware versions, these include a memory readout, which should only be available for development purposes. The memory range is not restricted. Thus, data that can be requested include the encryption key and all firmware.

3.3.2 *Firmware Update Process*

We reverse engineer and implement flashing custom firmware for the *Fitbit Flex* and *Fitbit Charge HR*. Our Android app, which performs firmware updates, is publicly available [Ste19]. Flashing custom firmware over the air requires the tracker either to be in plaintext mode or to have the encryption key extracted, as described in the previous section. The latter requires the tracker not to be patched with the security updates from October 2017. Note that in 2019, these trackers are still shipped without any patches. Hence, any "fresh" tracker can go through the steps shown in Figure 8, which are explained in the following. If there is neither plaintext mode nor encryption key available, i.e., because the fitness tracker is up to date, firmware updates can still be installed by opening the case, connecting a wire, and flashing via Serial Wire Debug (SWD).

original app update procedure Available firmware updates are shown in the smartphone app. The original smartphone app requires the user to initiate an update. User interaction ensures that the tracker stays in range and is charged during the process since flashing firmware over BLE onto the tracker takes a few minutes. Firmware updates are requested by the tracker from the server with a microdump payload, such that only firmware for this specific tracker can be retrieved. A firmware update is encapsulated into a (typically encrypted) microdump response, which can be sent to the tracker directly after local pairing without authentication.

Initiating firmware updates does not require authentication or user interaction. Plaintext updates and downgrades can be retransmitted to any tracker of the same model that is also in plaintext mode. We use this in our open implementation to provide original firmware images for any tracker in plaintext mode.

custom firmware installation *Fitbit* employs a two-stage update process shown in Figure 8, which we reverse-engineered from the firmware binary itself. In the first stage, the `BSL` is flashed, which contains enough functionality to enable the main application update procedure. In the second stage, the tracker reboots to the `BSL`

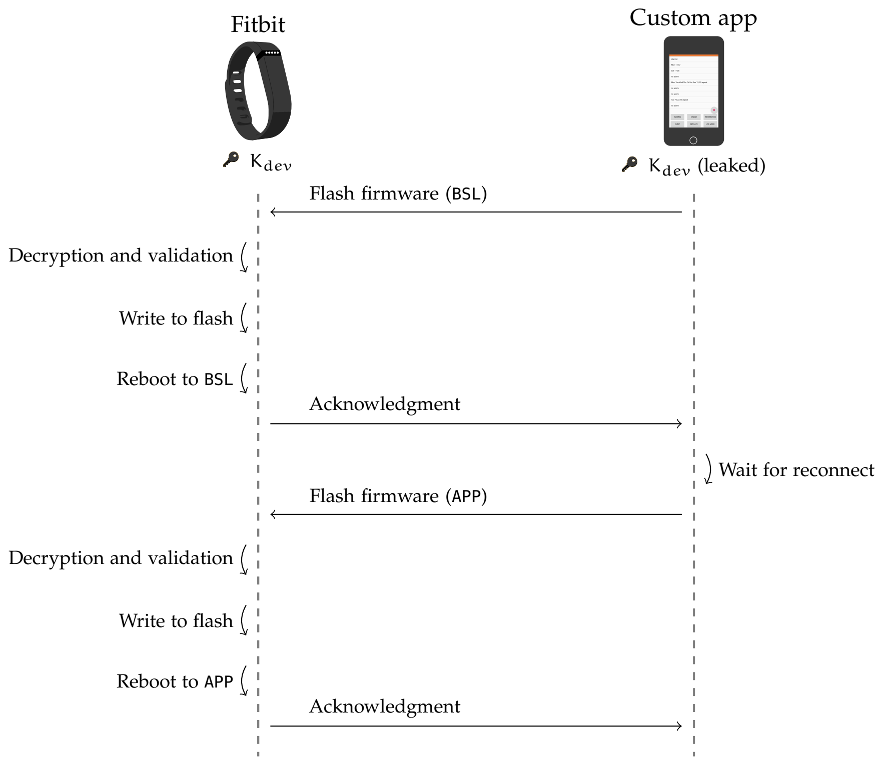


Figure 8: Client-side firmware update process.

and flashes the `APP`, which provides the full functionality of the tracker.[1] The update process is finished by rebooting into the `APP`.

A firmware update must pass several validity and integrity checks before being written to the persistent flash memory. First, generating valid firmware is difficult since a checksum is contained in an intermediate field inside the firmware, and before creating the checksum, a bit in the firmware must be flipped. This basic check prevents one from simply exchanging strings in the sniffed firmware update. Second, the firmware needs to be transferred in the firmware dump format—which was first observed in [Sch+16], but with vital information to reproduce firmware flashing missing. Notably, the first header contains the firmware length and destination memory address; the firmware segments start with the tracker's model number, continue with the action (`rebootToBSL`, `writeToFlash`, etc.) and end with segment length and checksum. Reboot actions have length and checksum set to zero, and they can be issued stand-alone. Third, if the tracker is in encrypted mode, the firmware needs to be encrypted with eXtended Tiny Encryption Algorithm (XTEA) in Encrypt-then-Authenticiate-then-Translate (EAX) mode.

Firmware updates have three fail-safe features. First, the separate update stages prevent flashing non-functioning firmware into both sections—as long as one of them is still working, the broken one can be repaired. Second, each firmware update segment starts with the first byte of the serial number, which is the tracker model number, to prevent from flashing incompatible firmware, in case the smartphone app would mix up connected trackers. Last, the firmware checks if the address range of the new firmware

1 The *Fitbit* internal naming scheme distinguishes between firmware `APP` and smartphone app, which interact with each other.

is within the flash region. Fail-safe features can be circumvented by two rounds of flashing, where the first round is malware disabling them. `BSL` and `APP` separation can be removed to make more room for malicious functionality.

Despite these fail-safe features, it is still possible to soft-brick a *Fitbit* fitness tracker within one round of firmware flashing. A bricked firmware can be the result when the flashed firmware crashes immediately due to invalid register access, which causes a reboot loop that always stays within the `APP` firmware.

The BLE communication setup only allows for a 20 byte transmission length per frame, which are acknowledged by the tracker individually, making the firmware update process last several minutes. Faster flashing can be enabled by only flashing the `APP`, or by reducing firmware size at the cost of functionality.

Attackers can wirelessly flash malware on trackers that are in plaintext mode, or that are vulnerable to authenticated memory readout.

3.4 RELATED WORK

We confine consideration to related work that delves deeper into the *Fitbit* ecosystem, and omit research on fitness trackers and IoT ecosystems in general. To the best of our knowledge, there is no scientific analysis of big real-world IoT ecosystems, including all their heterogeneous components. *Fitbit* puts many security measures in place where others employ no security at all, as a comprehensive study looking into 17 fitness trackers from various vendors except *Fitbit* shows [Fer+17b].

Cyr *et al.* were the first to research *Fitbit* security mechanisms by reverse-engineering the smartphone app [Cyr+14]. They found that cryptographic mechanisms are indeed used to secure the communication between tracker and server but did not reveal an in-depth understanding of these mechanisms and whether they may be compromised. The closest work to our contributions is by Schellevis *et al.*, who used tracker$\leftrightarrow$app authentication to extract memory at chosen addresses and undertook app and firmware analysis to understand in detail the encryption mechanisms implemented [Sch+16]. The memory readout vulnerability was found by comparing *Fitbit* Charge firmware updates, though this is fixed in the current firmware version. However, we still encountered this issue in the most current *Fitbit One* and *Flex* firmware. Despite understanding the basics of the firmware update, important gaps exist in their findings, which would need to be covered to generate custom updates. Schellevis recently extended the open-source Galileo software with authentication and megadump XTEA decryption features [Sch17], but the correct EAX Cipher-based Message Authentication Code (CMAC) mode is missing. A different firmware analysis also found the test command menu and integrated it in a *Unicorn/Radare2*-based emulation, which is an interesting toolchain for further investigations, but this analysis has not yet been continued [Rei16].

On the Galileo mailing list, it was claimed that trackers in encrypted mode could be switched back to plaintext by sending a valid plaintext server response [Dan17]. We tried this for the same tracker model, Flex, with both firmware versions. Still, it did not work, because the encryption options are not only "enable encryption" and "disable encryption". There is also a "trial encryption" when the tracker tries to initiate an end-to-end encrypted communication with the server and still switches back to plaintext if the server response is plaintext. The observed attack works for trackers in "trial encryption" but not for trackers in the wild with persistently enabled encryption.

AV-Test discovered that with older *Fitbit* Charge firmware versions, the live mode could be accessed by any device nearby after a recent authentication from a smartphone, obfuscating their finding by only listing the associated UUID [CSM16]. Goyal *et al.* revealed that a denial of service attack on a selected tracker is possible via BLE [GDS16]. Galileo recently implemented an experimental BLE synchronization, supporting local pairing and dump retrieval. The community does not yet know our commands to use the live mode, and set alarms and date.

In [Apv17], Apvrille summarizes everything recently found about the *Fitbit* protocols. Note that most of the commands documented are already implemented in the open Galileo tools or her `fittools`. Apvrille proposed to use random numbers generated by the tracker for the authentication challenges and published `fittools` implement this feature [Apv15], certainly authentication challenges are not random but cryptographically generated from partially predictable parameters and should not be used for such purposes. Moreover, Apvrille found a simple 20 byte echo command and claimed that this could already be an attack vector for firmware modifications—however, *Fitbit* stated this is not possible with this command, which we confirm with our firmware reverse-engineering [For15].

Earlier work focused on the security of old *Fitbit Ultra* models, which use an ANT protocol stack instead of BLE, and thus are not supported by the smartphone app. Instead, synchronization requires a Windows tool that was last updated in 2012 [Fit17]. The Ultra security model is fragile, as demonstrated by Rahman *et al.*, who documented the security and privacy issues encountered and proposed a FitLock tool to add authentication and encryption [RCB13]. Zhou *et al.* later highlighted security issues pertaining to the FitLock [ZP14], which prompted Rahman *et al.* to revisit *Fitbit* Ultra security and propose SensCrypt as a remedy [RCT16]. The *Fitbit* Ultra system is now only supported by the manufacturer for legacy reasons, and the current *Fitbit* infrastructure already employs encryption and authentication.

3.5 RESPONSIBLE DISCLOSURE AND OUTLOOK

In retrospect: two years after our responsible disclosure process with *Fitbit*, we still have a good working relationship with them. They have a team working on security and communicate encrypted. Whenever we report issues to them, they respond with a comprehensive list of findings in their own words and with their criticality levels assigned.

Any issue that can be fixed by changing the server's behavior was fixed immediately. *Fitbit* employed strict server-side filters, that block anyone upon invalid requests for increasing periods. Externally exploiting their infrastructure got very hard. Downgrades from encrypted communication to plaintext and checksum feedback are no longer possible.

Fitbit supports various generations of fitness trackers and smartwatches within the same smartphone app. Thus, the Bluetooth based communication is similar throughout all models, including the *Fitbit Ionic*. While this is great in terms of usability and requires *Fitbit* to only maintain one app, this also introduces legacy issues in new devices.

With recent firmware updates, users can decide if they want to disable live mode. Live mode transmits a summary of current activity in plaintext between app and tracker. It is based on the replayable app authentication. Even the recent *Fitbit Ionic* smartwatch supports live mode. We assume this feature is kept for performance reasons. Further authenticated commands, such as the memory readout, were consequently removed.

Hardware-related issues are harder to fix. Debug pins are part of the hardware layout and cannot be removed. Indeed, *Fitbit* did not update firmware on trackers coming out of the factory. Thus, even in 2019, they can still be flashed over the air with custom firmware. Assuming that users have installed security updates since 2017, this can only be exploited on purpose. However, health insurers are still offering discounts for customers with high step counts. To the best of our knowledge, health insurers do not force customers to have the most recent fitness tracker models.

MY CONTRIBUTION

My main contribution was the analysis of the firmware update process and firmware itself. The results of this chapter were already published in [Fer+17a] and [Cla+18a]. Text for Section 3.2 is already published in [Cla+18a, Sec. 3.2]. Figures Figure 7 and Figure 8 are adapted versions from [Cla+18a, Fig. 2, Fig. 6]. Section 3.3 based on [Cla+18a, Sec. 4.5]. Section 3.4 is taken from [Cla+18a, Sec. 8].

4 NEATO AND VORWERK VACUUM CLEANERS

We outline how *Neato* vacuum cleaning robots are different from many Internet of Things (IoT) devices from a technological perspective and why they are interesting to analyze in Section 4.1. Section 4.2 describes attack scenarios based on the vulnerabilities we found. Local attacks, which include bypassing secure boot to extract the firmware and then analyzing this firmware, are described in detail in Section 4.3 and Section 4.4. We reference related work in Section 4.5. Findings and bug-fixing are concluded in Section 4.6.

4.1 MOTIVATION

Vorwerk sold their first vacuum cleaner in 1930 in Germany, and since then is leading in vacuum cleaning technology in Germany [Vor19]. *Neato* released their first vacuum cleaning robot *XV-11* in 2010, followed by an enhanced version of this model for *Vorwerk* in 2011, the *VR100*. To the best of our knowledge, *Neato*'s *XV-11* was the first vacuum cleaning robot with a laser scanner for environmental mapping. This concept was copied later by other vendors such as *Xiaomi*. *Vorwerk*'s *VR200* and *VR300* both won test comparisons in Germany [Sti19]. In 2016, *Vorwerk* acquired *Neato*. Nonetheless, the marketing strategy remains similar: there are still cheaper *Neato* models and the more mature, expensive *Vorwerk* models for sale.

Feature-wise *Vorwerk* and *Neato* were leading vacuum cleaning robots in 2018 and still are as of now. They run very similar firmware on slightly different hardware. Yet, their cloud infrastructures are hosted separately. Features are usually released to *Neato* a little bit earlier for testing purposes and then pushed to *Vorwerk*. In the following, we will only refer to *Neato*, but this also includes *Vorwerk*.

From a technological perspective, *Neato* is very different from most IoT vendors. *Neato*'s first models were running on Linux, which is common amongst many vacuum cleaning robots, including *Xiaomi* [Gie19]. However, all of *Neato*'s connected models are based on the QNX operating system. The operating system change allows them to fit the whole uncompressed firmware, including autonomous cleaning logic and user interface, into just 20 MB, while *Xiaomi* comes with a full *Ubuntu* installation. *Neato* integrates a lot of security in their firmware. Firmware updates can be downloaded without owning a vacuum cleaner from their website, but they are encrypted and signed. To gain initial access to the QNX operating system and the vacuum cleaning services running on it, we used a secure boot bypass on the physical device. Overall, *Neato* sticks out on the connected device market for a rare software combination that is more common in industrial setups.

In contrast to *Fitbit*, *Neato* had the hardware attacker in mind when designing their product. We think that their threat model additionally includes the protection of their intellectual property since there is a lot of advanced logic in the autonomous vacuum cleaning process. However, they never considered an attacker would succeed in breaking their protection mechanisms and gain full control over just one of their robots. They also never considered their robots might be fake devices that could attack their cloud.

An overview of all vulnerabilities found in the *Neato* ecosystem is shown in Figure 9.

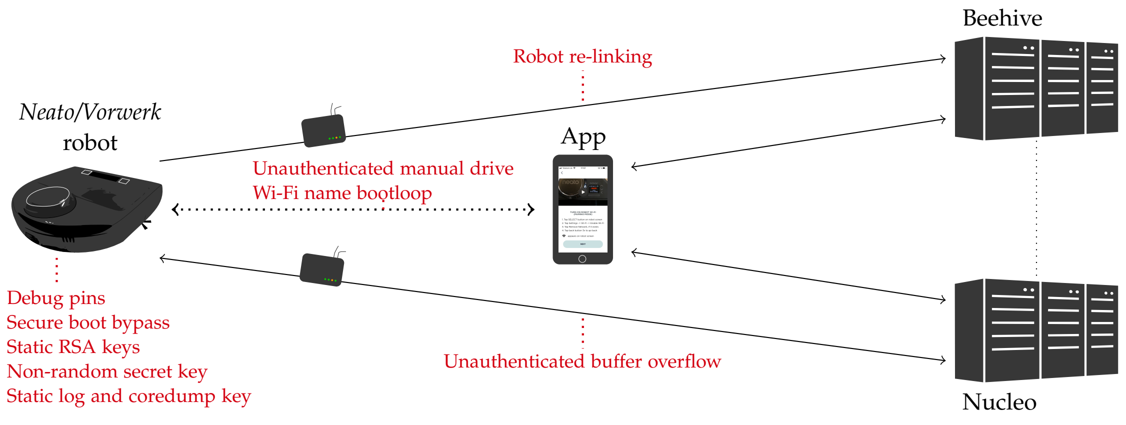


Figure 9: *Neato/Vorwerk* vulnerability analysis results.

4.2 ATTACK SCENARIOS

We follow the same categorization of attack scenarios as for *Fitbit* in Section 3.2.

(1) SPYING: The Neato ecosystem opens many attack vectors to espionage. An attacker might be motivated to spy on a user due to personal, political, or financial reasons.

- *Sniffing Ongoing Communication* — Communication between the vacuum cleaner and both cloud services is secured with Transport Layer Security (TLS). All robots have the same Rivest–Shamir–Adleman (RSA) keys for authentication. This makes it impossible to sniff this communication as an outside attacker. However, a security analyst with debugging access can access the session key to see the plaintext data.

- *Proximate Espionage* — An attacker located nearby the same Wi-Fi can always infer metadata, even if traffic is encrypted. Specific actions, such as sending a cleaning report map, are unique. Moreover, the direct mode between the robot and the smartphone app is not encrypted adequately encrypted and authenticated. Someone with physical access can use a secure boot bypass to extract memory, which contains current account and Wi-Fi credentials. Less complicated than the secure boot bypass is the possibility of obtaining the debug logs and core dumps, which is a functionality of the original firmware, and only secured with a static encryption key that has been the same for years. Depending on the crashed process. Also, core dumps may contain sensible data.

- *Remote Espionage* — A particular risk with Wi-Fi usage is that most users will probably use the same Wi-Fi for all their other devices. This means that an attacker who can get access to a robot over the cloud can sniff traffic and take over other machines within the same network. This is feasible even if there are a firewall or Network Address Translation (NAT) in place.

- *Cloud Espionage* — Access to *Nucleo* cloud data, such as maps, requires knowledge of a robot's identifier and secret key. Due to missing randomness during secret key generation, an attacker can infer the secret key based on the robot's identifier and cloud linking date.

- *Spyware* — For an attacker with physical access, it is possible to boot a temporary spyware image on a robot. To make spyware on the robot persistent, an attacker would have to break secure boot and not just bypass it temporarily. A hardware add-on that always secretly boots an alternate firmware image would be feasible to program and install.

(2) FINANCIAL INTERESTS: Users themselves cannot gain any financial rewards for their data in the current *Neato* business model. Nonetheless, data recorded by vacuum cleaning robots might be of interest to external companies for advertising purposes. Criminals might even misuse data against users.

Monetary value can potentially not be used for the advantage of the user because there are currently no business models to reward them. Companies and criminals still have several financial interests.

- *Selling Map Data* — Vacuum cleaner data is precious for marketing purposes. From the map, the size of the apartment and kind of furniture can be inferred. Data also includes information about moving objects, thereby giving hints on how many persons live in a household and when they are home. The latter is also valuable information for burglars.

- *Ransomware* — Assuming that an attacker cannot permanently install ransomware, a user can always disable her Wi-Fi access point and reboot the robot. The robot can still vacuum without the cloud, but cannot report maps or respect virtual no-go lines. A user who does not know about this technical background might still be tricked by ransomware. This ransomware would need to be rolled out over the cloud, meaning that *Neato* could easily filter it out. So, in principle, ransomware is a possible scenario, but hard to be put into practice.

(3) DENIAL OF SERVICE: In the *Neato* ecosystem, Denial of Service (DoS) is more critical to the servers than to the vacuum cleaners themselves. Robots without Wi-Fi connection still provide basic functionality, unless they get permanently bricked by an attacker.

- *Local Wireless Commands* — An attacker in the same Wi-Fi can control the robot in direct mode. Even though an authentication header is sent from the legitimate user's smartphone, this is not checked by the robot. This can be used to move the robot towards the wrong direction.

- *Virtually Stealing Vacuum Cleaners* — Currently, secret keys are generated locally on the robot each time it is linked to an account. The only security mechanism that ensures a robot is authentic is an RSA key, which is the same on all robots. An attacker can link any robot identifier to her account, which removes the link to the legitimate account. *Neato* plans to replace the current secret key mechanism to no longer only depend on information generated by the robot.

- *Abusing Cloud Infrastructure* — Anyone in possession of the static robot RSA key used among all robots can connect to the cloud services and exchange data using them. This makes the *Neato* infrastructure susceptible to being misused by botnets, as well as being exposed to high amounts of traffic that slow down services.
- *Wi-Fi Name Brick* — A robot can be bricked permanently by configuring it to use a malicious Wi-Fi name. This issue can only be fixed by sending in the robot to customer service.

(4) CLOUD INDEPENDENCE: Due to secure boot, cloud independence is hard to achieve. Proper malware protection leads to high user dependence in this case.

- *Full-Featured Separate Infrastructure* — Data exchanged with the cloud is encoded in JavaScript Object Notation (JSON) in both directions, which makes it easy to reverse engineer. Secure boot makes it nevertheless very hard to become independent from the cloud infrastructure. Each time the robot reboots, it needs to be exploited again. If a robot has a vulnerability that allows exploitation over the local network and the user does not install updates, this vulnerability can be used to exploit the robot over and over again, such that a separate infrastructure can be used.
- *Cloud Independence with Reduced Function Set* — The features provided in direct mode are not very useful. They can be used to turn on the vacuum cleaning robot's engine and move it into directions. However, there is no possibility of getting sensor feedback to drive it autonomously. The robot would need to use random strategies similar to those that can be found in low-cost robots without sensors.

(5) NON-ADVERSARIAL APPLICATIONS: Opening up the *Neato* ecosystem can have further applications that are not for personal advantage.

- *Laser and Ultrasonic Sensors* — The sensors in vacuum cleaners are very interesting. People even dissect *Neato* robots to analyze their laser system. A robot can not only be used for vacuum cleaning but also to sense its environment in general and move around. Additional sensors or actuators can be attached to a robot over the Universal Serial Bus (USB) interface, to customize it further.

The following Sections 4.3 and 4.4 focus on local attacks, as these were my main contribution to this topic. These local attacks were the entry point for advanced analysis of cloud components.

4.3 BYPASSING SECURE BOOT

Access to the firmware running on the robot is required for more advanced security research. This includes but is not limited to the analysis of mutually authenticated

Table 1: Neato/Vorwerk firmware comparison.

Product Name	Firmware Version	Build Date
Neato BotVac Connected	2.2.0-296	Dec 12 2016
Neato BotVac Connected D7	4.2.0-102	Jul 12 2018
	4.4.0-72	Dec 22 2018
Vorwerk Kobold VR300	4.2.2-137	Aug 30 2018
	4.2.5-166	Jan 11 2019

TLS connections and debugging applications while they are running. *Neato* is using a custom *AM335x* chip with secure boot enabled [MG18]. This means that the flash chip only contains an encrypted and signed system image; a physical attacker cannot merely unsolder the chip to obtain it. Firmware updates use the same encrypted and signed format. Changing and debugging firmware on a robot, therefore, requires bypassing secure boot.

We start with a hardware analysis in Section 4.3.1, which leads to a hidden boot menu in Section 4.3.2. We exploit this to extract *Neato*'s system image (see Section 4.3.3). The described security issue, which was assigned *CVE-2018-20785*, has been fixed in the *Neato BotVac Connected D7* firmware version 4.4.0-72, all other versions listed in Table 1 are vulnerable. The *Vorwerk* firmware has a more recent release date, but features are tested first on *Neato* and rolled out later for *Vorwerk*.

4.3.1 *Hardware Analysis*

After teardown of the *Neato BotVac Connected*, further analysis of all components is required. The motherboard has many testing points and undocumented chips, with essential parts depicted in Figure 10. We connected all testing points to a logic analyzer, and rebooted the robot multiple times. Most of the testing points do not show any exciting output. Some might be for sensors and other features not relevant for system access. In addition to test points, the right front of the motherboard contains three extra-large unlabeled pins, which are a serial interface with clock, read, and write. The ground needs to be taken from another source, such as the internal USB port. It prints the following text during startup and remains silent afterward:

```
BotVac Connected "<Fast Memory>"(84038)
```

The *Neato BotVac Connected D7* features a similar serial port, it is located on the motherboard's left and has four connection pins. Access to the *Vorwerk Kobold VR300* serial port does not require disassembling it; it is located next to the USB port. By default, these serial interfaces do not enable any access. Moreover, they do not leak any information about the main chip or operating system.

The main chip on the *Neato Botvac Connected* is the *SNI20952608ZCE* in the center of Figure 10. Since it is a custom chip, there is no information or documentation available. The form factor *962B* leads to the *AM335x* chip family manufactured by *Texas Instruments*. The *Neato BotVac Connected D7* is using the same form factor and chip family, but the chip is labeled *SNI5095260BZCZ*.

4.3.2 *Hidden Boot Menu*

During startup, QNX first loads an Initial Program Load (IPL), which then loads an Image File System (IFS) containing the operating system. Instead of a QNX IPL, Linux typically uses the more conventional setup of MLO and U-Boot on *AM335x*. Either way, these setups enable booting an operating system. Neato is using secure boot, which means that encryption keys configured on the *AM335x* chip decrypt and verify both, IPL and IFS, prior to execution.

Neato's standard setup is to boot the IFS containing the firmware from the external flash chip without showing any options. While testing the partially documented *Neato* USB command line [Nea15], we found that `TestMode On` followed by `SetSystemMode PowerCycle` enables a hidden IPL boot menu on the serial interface shown in Figure 11. After entering these commands, it shows the following boot options:

```
BotVac Connected "<Fast Memory>"(84038)
Press enter twice within the next 2 seconds for boot menu
```

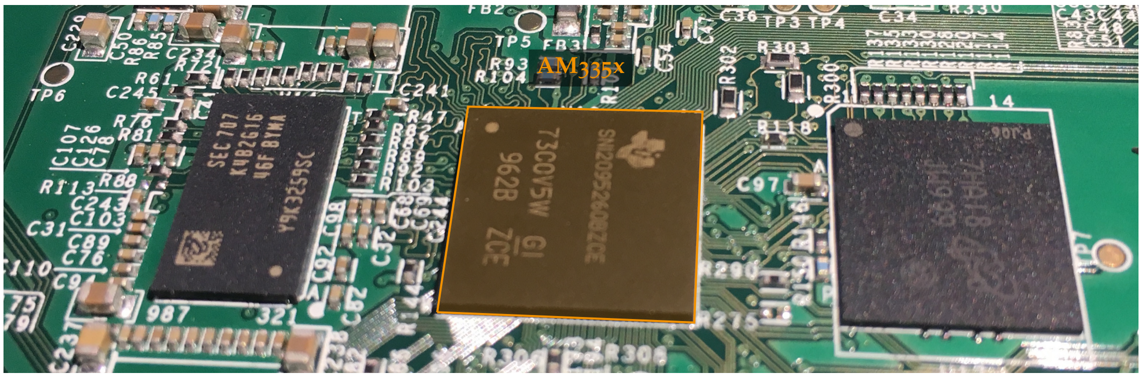


Figure 10: Part of the *Neato BotVac Connected* motherboard.

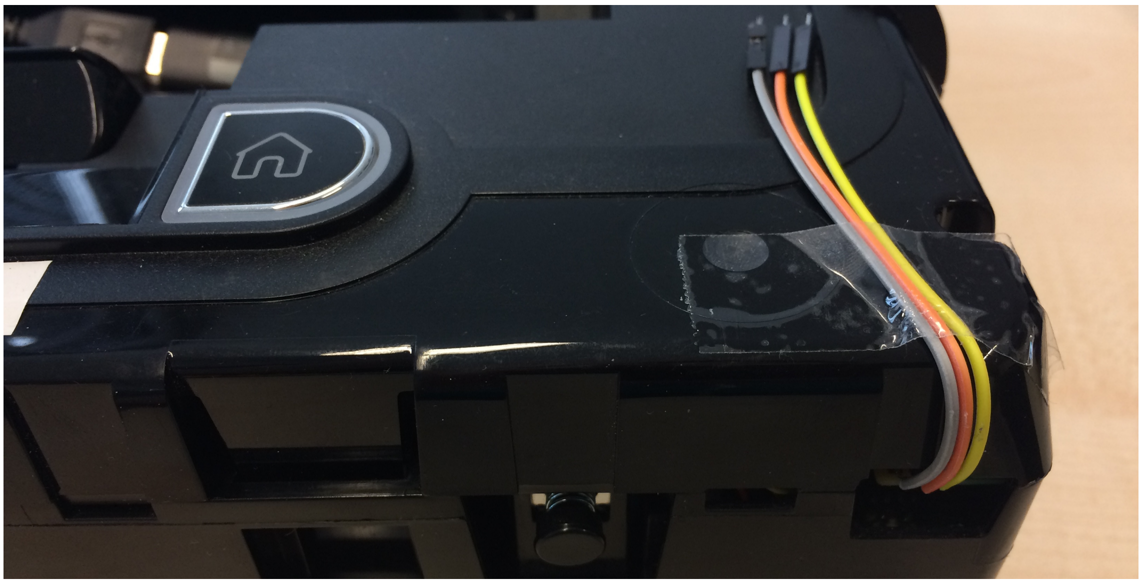


Figure 11: Unlabeled serial interface.

```
**Commands:
Press 'M' to load IFS from main image flash
Press 'X' for serial download, using XModem-1k
Press '1' for XModem-1k download at 1Mb/s
Press 'S' to scan existing memory without download
```

Note that `XModem-1k download` makes the robot download a system image from the serial port and boot it. This does not enable system image extraction. On uploading a valid QNX IFS no secure boot checks are performed, representing a bypass. Loading a *BeagleBone* QNX IFS from *Foundry27* for *AM335x* works in principle, it prints some information on the serial interface but crashes when initializing processors.

The original *AM335x Foundry27* board support package includes a very similar IPL procedure in `src/hardware/ipl/ boards/am335x/main.c`. It enables sending an IFS over the QNX serial protocol `sendnto`. Neato's implementation represents a variation, and more *AM335x*-based products might be affected by the secure boot bypass. The more common setup of MLO and U-Boot also allows loading arbitrary images by default. In general, vendors relying on secure boot should carefully check for such contradicting setups.

4.3.3 *Memory Extraction*

We exploit the secure boot bypass to launch a cold boot attack, which is possible as neither the *AM335x* chip nor the IPL reset memory contents during startup.

With QNX SDP 6.5 the *Foundry27 AM335x* image can be altered. The main issue here is that hardware cannot be initialized. This is since memory is mapped to different addresses on the *Neato* chip variant, and any invalid memory access freezes the chip. The address space is 32 bit (4 GB), and therefore it is not possible to guess all addresses required to boot a fully functional QNX system.

At this stage, an attacker can print memory contents until an invalid memory access is made. The serial interface is very slow in default settings (baud rate 115200), upgrading it to $1\,\mathrm{Mbit\,s^{-1}}$ worked in our setup. Despite the slow speed and partially unknown memory layout, this poses a large attack surface for cold boot attacks.

By default, QNX is located at address `0x81000000`. First memory dumps show that despite booting our own QNX IFS this memory region still contains some of the original Neato IFS contents. We relocate our custom QNX IFS to an address between `0x80000000` and `0x81000000` and perform a full memory dump of the upper 2 GB of the address space. This area contains the same 256 MB, which are mapped 8 times to the memory region. Each mapping contains the full *Neato* IFS and any Random Access Memory (RAM) contents that were present before the reboot, including Wi-Fi credentials.

4.4 STATIC FIRMWARE ANALYSIS

Neato's IFS is a rich source for information and enables static firmware analysis. It contains all important binaries and the shell scripts coordinating them. For example `/bin/robot` is called by a script `/etc/brain.sh` and the watchdog `/bin/pinky` is started via `/etc/pinky.sh`. An overview of the main tasks and their most important interactions is shown in Figure 12. `robot` is the largest binary and responsible for the main features, starting at the interactive command line and ending at all cleaning logic. It receives *Nucleo* cloud commands from `astro` and generates replies. The manual drive mode, which is issued locally via the smartphone app, is first redirected from `webserver` to `astro` and then processed similarly. `conman` manages Wi-Fi connections. Messages are exchanged via proprietary `libNeatoIPC` calls, as well as standard QNX message queues. In the following, we focus on those components relevant for further findings.

Testing the robot's wireless interfaces produced core dumps that are separate files in the so-called "Blackbox Logs", which can be copied to a USB stick connected to the *Neato* USB interface. Such a core dump is the entry point for our buffer overflow analysis that allows unauthenticated privileged remote access on any robot [Ull+19].

For security reasons, these core dumps are encrypted. A key question for further research is how this encryption works. Shell scripts in the *Neato* IFS copy and rotate these logs, encryption is performed by `/bin/rc4_crypt` called without arguments. Reverse engineering `rc4_crypt` reveals the hard-coded password `*^JEd4W!I`. This password is valid for each of the three tested robots. We assume the same password is used throughout all firmware versions. *CVE-2018-17177* was assigned to this issue.

The robot also features a number of encrypted private RSA keys, which are located in `/var/keys/`. One of the keys is named `vendorPrivateKeyProduction` (or `serverPrivateKeyProduction` in the most recent robots) and is used for authenticating the robot against the cloud components. By examining the key decryption procedure in the robot binaries, we were able to decrypt the keys ourselves and launched several tests. Due to the high confidentiality of these keys, we agreed not to disclose details on the decryption process.

4.5 RELATED WORK

Even though vacuum cleaning robots are a target of interest, not much security research has been published on *Neato*, except for an exploit chain for attackers in the same local network [Kie18a; Kie18b]. For *Vorwerk* robots, which are re-branded *Neato* robots with minor hardware and settings upgrades, not a single Common Vulnerabilities and Exposure (CVE) number is known.

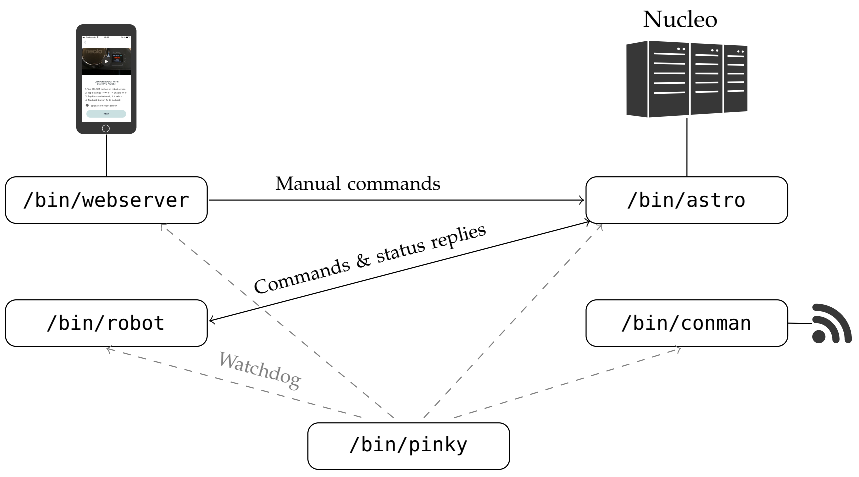


Figure 12: Core tasks running on *Neato BotVac Connected* and their most important actions.

4.6 RESPONSIBLE DISCLOSURE AND OUTLOOK

Neato vacuum cleaners are high-cost, but customers get proper hardware and software in return. IoT security analysis often focuses on cheap products, which do not have any budget for security. Analysts tend to uncover low-hanging vulnerabilities in those products since externals like press and social media often cannot determine the difference between product types and complexity. However, high-cost products with a security budget have many more interesting internals, that are valuable for IoT developers.

It took us quite some time to get into contact with the actual product developers since customer care ignored our requests for months. After establishing initial contact, we had excellent cooperation with *Neato*. They claim to have fixed all security issues we found in the *Vorwerk* 4.4.1 software, as well as the end of June software release for *Neato* robots. We can only confirm that they fixed the secure boot issue, because we would need the secure boot bypass to extract the firmware for further tests. However, we assume that they also fixed the remaining issues.

MY CONTRIBUTION

My contribution was analyzing hardware and bypassing secure boot. I did the initial static firmware analysis and revealed how core dumps are encrypted. Section 4.3 is based on [Ull+19, Sec. 3.1]. Section 4.4 is already published in [Ull+19, Sec. 3.2]. Section 4.5 was taken from [Ull+19, Sec. 1].

5 BLUETOOTH LOCATION FINDERS

Section 5.1 explains how Bluetooth location finders are different from the previously shown fitness tracking and vacuum cleaning ecosystems, and explains why they are interesting to analyze. Attacks based on the vulnerabilities we found are discussed in Section 5.2. Functions implemented by all Bluetooth finders listed in Section 5.3 can be found in reverse-engineered firmware, as explained in Section 5.4. Section 5.5 concludes our findings and the responsible disclosure process.

5.1 MOTIVATION

Bluetooth finders are connected to a smartphone. Once the Bluetooth Low Energy (BLE) connection is lost, an app on the smartphone will issue a sound. While the connection is established, the app regularly updates the last seen location of the finder to the smartphone's current Global Positioning System (GPS) position.

As the minimum requirement for a functional Bluetooth finder is to maintain a connection, these devices are relatively cheap. Some are available for less than 2€, advanced finders from well-known vendors start at around 15€. Finders that include a GPS receiver are conceptionally different and more expensive and thus are excluded in the following.

Their simple technology enables Bluetooth finders to run for a year powered by a standard button cell. This restricts their technology a lot compared to other finder systems such as *Apple*'s upcoming *FindMy*, which runs on smartphones and laptops that are powered by a large battery that is charged regularly [Gre19].

Many Bluetooth finders come with an integrated Internet of Things (IoT) ecosystem. Their last seen position is not only saved locally in the app but also reported to a server. The app does not only maintain a connection to the user's finders but also scans for

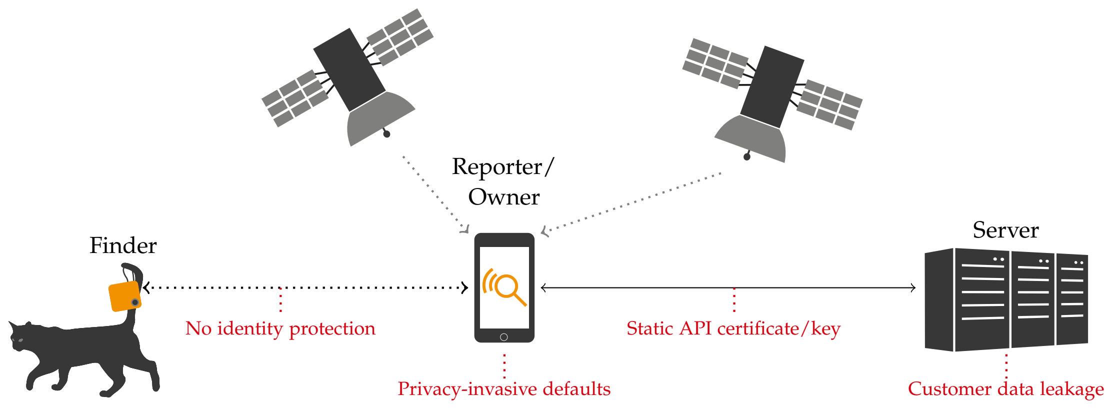


Figure 13: Bluetooth finder vulnerability analysis results in various products.

currently disconnected and thus potentially lost finders. Last known position of lost finders are reported to the server, which in turn alerts the owner of the lost finder. This feature is sold under various names, such as *crowd search*.

Bluetooth finders became famous in 2013, when *Tile* raised $ 2.6 million with a crowd-funding campaign [Lom13]. Since then, various finders appeared on the market. We analyzed top-ranked finders sold online worldwide: *Nut Find3*, *Tile Mate*, *Tile Pro*, *musegear finder*, *Pearl Callstel Key Finder*, *Gigaset g-tag*, and a couple of no-name finders based on the *ST17H26* chip that can be managed over various smartphone apps including *iTracing*, *iSearching*, and *FindELFI*. An overview of generic vulnerabilities found across all kinds of Bluetooth finders is shown in Figure 13.

5.2 ATTACK SCENARIOS

We follow the same categorization of attack scenarios as for *Fitbit* in Section 3.2 and *Neato* in Section 4.2.

(1) SPYING: Bluetooth finders are predestined for spying since they broadcast their identity and are associated with GPS locations. Because their firmware is very simplistic and does not know privacy-sensitive information itself, it is immune to spyware, sniffing, and similar attacks.

- *Proximate Espionage* — Most finders reveal their identity to anyone nearby. The owner of the item a finder is attached to can be tracked. Besides continuous tracking, it might also be interesting for the attacker to know if the victim is at home or at work.
- *Cloud Espionage* — Data leakage in the cloud is a high risk for Bluetooth finders. Usually, the cloud not only keeps the last location but also saves a location history. Cloud-based Bluetooth finders require a large user base for crowd search; thus, a leak in a popular finder system will concern a lot of data.

(2) FINANCIAL INTERESTS: As of now, Bluetooth finders do not expose location histories of items to third parties. Thus, there is no business model where a user can take economic advantage from uploading fake histories. However, companies or criminals might still have financial interests in this data. Bluetooth finders are very cheap. Thus, ransomware is not considered to be a threat.

- *Selling Position Data* — Finders are very cheap and often only work with a cloud infrastructure. It might be that especially the more affordable finders on the market can only survive if they make money out of position data.

(3) DENIAL OF SERVICE: Loss detection depends on an always-on Bluetooth connection, and ideally, the smartphone should continuously save and maybe even report the last GPS position to the cloud. This makes this hardware susceptible to denial of service attacks.

- *Wireless Jamming and Injection* — BLE jamming is supported by the cheap `btlejack` open source solution. This can be potentially used to disable selected Bluetooth finders to inject false commands and identifiers.

- *Finder Forgery* — Finders without security properties can easily be duplicated by copying their Generic Attribute Profile (GATT) services and values. This might confuse other users or cause misleading location reports.
- *Abusing Cloud Infrastructure* — Assuming that the validity of finder identifiers is not checked or new identifiers can easily be generated, and if the finder does not have proof of belonging to the vendor, an attacker can spawn and register new finders and cause a high load on the cloud infrastructure. Another way to slow down the infrastructure is the generation of false lost finder location reports.

(4) CLOUD INDEPENDENCE: Due to the privacy-sensitive location information, cloud independence is vital for users.

- *Full-Featured Separate Infrastructure* — Some finders do not employ encryption and simply broadcast an identifier. They could be ported to any other infrastructure. However, trackers without security are, by default, not privacy-sensitive anyway.

(5) NON-ADVERSARIAL APPLICATIONS: Bluetooth finders do not have any interesting sensors, so there are no applications that would not already be available on various platforms. Due to their beaconing functionality, they could be used for non-malicious applications such as indoor navigation.

5.3 BASIC BLUETOOTH FINDER

In general, a Bluetooth finder is straightforward to implement. We assume this to be one of the reasons why there are so many finders on the market. Without security, functions found across various finders are:

1. Playing a sound on the finder once the Bluetooth connection is lost,
2. playing a sound on the finder when the user pages for it via the smartphone app,
3. playing a sound in the smartphone app when the user presses a button on the finder, and
4. sending BLE advertisements, including an identifier once the connection to the smartphone is lost so that it can be found by anyone.

Most of these functions are standard GATT services [Blu19b]. *(1)* is defined as *Link Loss* service, *(2)* as *Immediate Alert* service, and *(4)* is the generic behavior of any BLE device. Many Software Development Kits (SDKs) for BLE devices already contain those services. Only the functionality to page the smartphone itself *(3)* needs to be implemented in the app, as well as all additional cloud features.

5.4 REVERSE ENGINEERING OF THE NUT FINDER FIRMWARE

During our analysis of *Nut*, we found that firmware for all finder models can be downloaded from their server. Firmware images are encrypted, but with a static encryption key. This enables us to analyze their firmware without a finder teardown.

Their newest model, which is not yet available on the market, includes GPS and a baseband modem. It works without smartphone connection. We do not further analyze this firmware, because there is no available hardware, and the communication paradigm is substantially different.

The firmware of the *Nut find 3* contains no symbols. Its structure is still comparably straight forward to reverse engineer. From the firmware file name, it is already known that the underlying hardware is an *nRF51422* chip. The finder implements multiple BLE GATT services [Blu19b]. These GATT services can be listed over the air and are identified by Universal Unique Identifiers (UUIDs). Functions implementing a service can be found in the firmware by searching for these UUIDs. The only non-standard UUIDs are for playing a custom sound upon loss and getting the finder's identifier.

The SDK for the *nRF51* series already comes with several examples. These include a basic Bluetooth finder, that triggers an action on connection loss [Nor19a]. Most UUIDs are simply implemented the same as in the example. We assume similar examples exist across various Bluetooth SDKs, because checking a connection status is a substantial functionality.

Since there is development hardware and an SDK with a Bluetooth finder example, we do not consider binary patching a proper method for experimentation and modification. Porting a Bluetooth finder firmware to be compatible with binary patching is more effort than implementing the missing services in an *nRF51* chip.

5.5 RESPONSIBLE DISCLOSURE AND OUTLOOK

The *Nut* firmware example shows that there is only a little logic in the finder. All finders had little or no security, which is reasonable regarding the hardware power and price constraints. Most features are implemented in the smartphone app and cloud. During our research, we found issues in various cloud implementations and informed the vendors.

The worst customer data leakage was found in the *Nut* infrastructure. As of now, we did not publish the vulnerability itself. We and Computer Emergency Response Team (CERT)/CC attempted to reach the vendor, to get the vulnerability fixed, for more than a year.

We wrote *Nut* an email on October 7 2018. On January 18 2019 we sent a letter to their mailbox in California. We also sent them a *Twitter* direct message on February 4 2019. Due to our unsuccessful attempts, we reported the issue to Bundesamt für Sicherheit in der Informationstechnik (BSI) CERT on March 5 2019 who then forwarded it to US CERT without further response. Thus, BSI forwarded the information to CERT/CC on April 30 2019. We contacted *Nut* again over *Facebook* on April 3 2019 and gave them a phone call to the number listed on their *Facebook* site but were not able to contact them. We were also able to obtain a mail address from one of the developers—we sent him an email in Chinese on April 5 2019. CERT/CC replied on August 21 2019 that they also could not reach *Nut*, and that we should obtain Common Vulnerabilities and Exposure (CVE) numbers to publish our findings. However, only missing encryption in the app got assigned a CVE on September 12 2019, as the remaining issues are *Nut* specific programming bugs on the server-side.

Bluetooth finders are an excellent example of "deploy and forget" IoT devices. Due to their simple firmware and hardware, they remain functional for a long time. They are potentially insecure by design, and their primary purpose is to collect privacy-sensitive data.

MY CONTRIBUTION

In this project, my contribution was firmware reverse engineering of *Nut* and communicating our findings to various vendors. The original idea came from Erik Tews. Max Weller designed and implemented *PrivateFind*. Max Weller did most of the security analysis, and Fabian Ullrich assisted him with analyzing the more sophisticated *Tile* ecosystem.

Part III

MODIFYING OFF-THE-SHELF DEVICES

The previous part on IoT outlined many examples where the user is dependent on the software provided by vendors. Many devices can be unleashed by reverse-engineering the update process and firmware itself to add new functionality. Since proprietary products ship without source code, binary patching is required to modify off-the-shelf devices (Chapter 6).

Fitbit fitness trackers are one example of an IoT device where binary patching has power over most functionality. It can be used for malicious purposes such as arbitrary step count generation, as well as non-malicious research applications such as raw accelerometer data access (Chapter 7).

Modification of firmware that is closer to hardware is typically even more powerful, but harder to apply in practice. This is the case for Bluetooth firmware. Only a few open solutions in the area of Bluetooth analysis exist, and they do not implement the full feature set of the Bluetooth specification. Modified proprietary Bluetooth firmware can be used to monitor and inject link-layer traffic, i.e., to analyze communication in an IoT scenario. We repurpose firmware of the wide-spread *Broadcom* and *Cypress* Bluetooth chips in Chapter 8.

6 BINARY PATCHING

Binary patching is a technique that enables the modification of proprietary firmware images. It does not require any source code or symbols, even though these help to locate functions that are to be patched. Section 6.1 compares the standard approach of patching firmware with known source code to binary patching. Platform requirements are listed in Section 6.2. How to identify functions is briefly explained in Section 6.3, as this also depends a lot on the specific firmware. The following Chapters 7 and 8 cover practical examples in more detail.

6.1 SOURCE CODE PATCHING VS. BINARY PATCHING

A vendor with the source code of a firmware can add arbitrary changes, re-build the whole firmware and install it on a target device. Under the assumption that a build is reproducible, it will always generate the same binary from the same source code. Note that this is not necessarily the case in practice. However, just changing a small detail and recompiling the firmware, such as adding a new branch condition, will typically rearrange most function locations in the resulting firmware image. Without access to the source code, performing the same kind of optimization and rearrangement as a compiler is practically impossible.

Most firmware is not signed and the target device has free space, this allows for binary modification using a different process. A typical use case is to add debug print statements to get a better understanding of the firmware. Assuming there is a function call to `weird_function()`, this call can be replaced by a call to a custom function `debug_weird_function()`. Both function calls require the same length on architectures like Advanced RISC Machine (ARM), i.e., replacing one function call with another only changes 4 B within the proprietary part of the firmware binary. The new instructions for the custom function `debug_weird_function()` are inserted at the end of the firmware binary. This new function could print information to a debug port first and then call the original `weird_function()`. This way, binary patching provides a mechanism to introduce changes into a binary, while keeping the modifications to the original firmware image minimal.

Binary patches can be written in C if the whole patching process is properly integrated into a compiler. Thus, the patches become readable, and a previously closed-source firmware can be extended with open-source code.

6.2 PLATFORM REQUIREMENTS

Within our group, *Nexmon* was developed to support the patching of ARM binaries [Sch18]. While the focus of the initial project was to patch *Broadcom* Wi-Fi firmware, we ported it to the *Fitbit Flex* and *Charge HR* (Chapter 7) and the *Broadcom* and *Cypress* Bluetooth firmware (Chapter 8).

The *Broadcom* Bluetooth firmware has a fundamentally different patching mechanism compared to the Wi-Fi firmware. In principle, it is possible to compile patches with *Nexmon* for both of them. Significant parts of the Wi-Fi firmware are loaded to Random Access Memory (RAM) during driver initialization, which allows rewriting most of it. The Bluetooth patching mechanism does not allow rewriting the whole firmware binary, as the firmware is located in Read-Only Memory (ROM). However, it has a transparent overlay, called *Patchram*, that allows for patching a limited number of 4 B slots—which correspond to one branch instruction each. On the one hand, since the Bluetooth patching mechanism is constrained in the number of patches that can be applied to the firmware, the quantity of function calls that get overwritten needs to be reduced compared to the standard *Nexmon* approach. On the other hand, the Bluetooth chip can even be patched during runtime, which allows for self-modifying firmware even within the ROM. One example of this is the tracepoint mechanism that we implemented within the *InternalBlue* framework, which adds a hook that first prints out debug information upon a function call and then deletes itself (see Section 8.6.2.1).

In principle, binary patching also works on platforms that are significantly different from ARM, such as PowerPC (PPC) [Mü19]. However, different calling and branching conventions on PPC require a lot of the patching logic to be redefined. In the PPC firmware, we patched one restriction was the maximum length of a jump, which is too short to reach a free memory area containing further instructions. Thus, an intermediate jump location must be added to the firmware. A binary patcher must be aware of such individual cases and handle them properly.

6.3 IDENTIFYING FUNCTIONS

Prior to patching a function, it must be identified correct. Most desktop and smartphone applications ship with symbols, or at least definitions of function and memory locations. This is rarely the case for embedded firmware. The only advantage in embedded firmware is that the environment they typically run in does not have sufficient hardware capabilities to handle sophisticated obfuscation techniques.

The first step when obtaining a firmware image is to identify the correct memory location where it is loaded and executed. Function calls and variable references are a mix of relative and absolute calls, and thus will break if the firmware is loaded to a false location. Getting the firmware image into a state where disassembly looks correctly is a fundamental requirement to do further analysis and function matching.

If the firmware still contains some debug prints, these can be used to name functions initially. Furthermore there might be constants that hint to encryption algorithms or other publicly available source code and specifications. Moreover, tools like *BinDiff* and *Diaphora* can be used to compare unknown functions to other binaries with known symbols [Zyn19; Kor19].

The *Broadcom* Bluetooth firmware has around 11 000 functions, whereas the *Fitbit* firmware only has about 1500 functions. Both firmwares do not have any symbols. However, symbols of some Bluetooth chips are contained in a Software Development Kit (SDK), and the Bluetooth specification explains a lot of details.

7 FITBIT FITNESS TRACKER FIRMWARE

In the following, we explain why and how we analyzed the firmware in Section 7.1, to then apply patches with *Nexmon* as described in Section 7.2. We discuss the capabilities of binary patching on a fitness tracker in Section 7.3.

7.1 STATIC FIRMWARE ANALYSIS

While *Fitbit* officially documents most of the Web Application Programming Interface (API) communication, we were able to extract missing details from the analysis of the smartphone app. In contrast, the proprietary wireless tracker commands sent via Bluetooth Low Energy (BLE)/Generic Attribute Profile (GATT) are undocumented. Reverse-engineering the tracker's firmware is required to fully understand command interpretation and dump generation. Firmware can be extracted from trackers themselves or a sniffed firmware update [Fer+17a; Sch+16]. In the following, we go beyond firmware extraction, and we reverse-engineer and modify the firmware internals.

Static and dynamic analyses provide further insights. We follow protocol parsing and error handling and identify undocumented commands. We modify the firmware and enable support for GNU Debugger (GDB) to facilitate dynamic analysis. We successfully change security-relevant functions inside the original firmware code to disable authentication and encryption in the *Fitbit* Bluetooth protocol. We also open the firmware for better security mechanisms and raw accelerometer data readout.

The analyzed firmware are the *Flex* versions 7.64, 7.81, and 7.88, as well as the *Charge HR* version 18.32 and 18.128. These versions do not contain function names or strings in the main part, which complicates reverse-engineering. We use IDA Pro [Hex17] for static analysis of the firmware binaries.

MEMORY LAYOUT Knowing the main processing chip used (*STM32L151UC* based on an ARM Cortex M3 [Fer+17a]), we can infer which purpose certain memory areas serve.

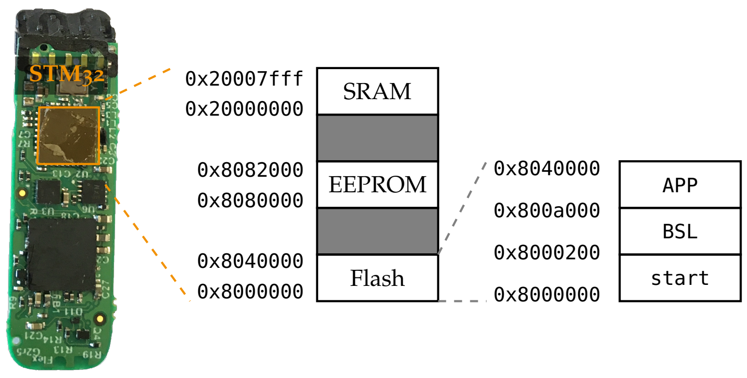


Figure 14: Memory layout of a disassembled *Fitbit Flex*.

This explains all the addresses we show in the left part of Figure 14 [STM17a]. A closer inspection reveals that a firmware update targets two memory areas, `BSL` and `APP`. The `start` area is never updated and contains code which either boots into `BSL` or `APP`. While `APP` contains all functions used during regular operation, `BSL` contains a reduced set of functions that are required for upgrading the `APP` code. We show the structure of this flash memory on the right-hand side of Figure 14.

The remaining two regions are the Electrically Erasable Programmable Read-Only Memory (EEPROM) and Static RAM (SRAM). Data with long-term persistence, such as the K_{dev}, the serial number, and encryption options are stored in the EEPROM. The EEPROM also contains data that should survive an empty battery, such as time zone settings, information about the user, and not yet synchronized fitness data. The short-term memory required for parsing and generating frames is effectively mapped to the SRAM.

Note that the memory layout is very similar within different tracker models. For example, the *Fitbit Charge HR* has the same `BSL` and `APP` separation, the same base addresses and just slightly higher offsets.

BLUETOOTH CHIP The BLE chip used in the *Fitbit Flex* is the *nRF 8001* from *Nordic Semiconductor* [iFi17]. The product specification [Nor19b] states that one byte long commands that must be used to communicate with the chip. We use these commands to locate the functions responsible for the *Fitbit*'s BLE communication. Further investigation reveals that the library in the firmware is almost identical to an open-source library used for an *Arduino* BLE Breakout Board [Ada17]. Therefore, all BLE related function names can be carried over to the tracker's firmware. This helps us to identify the execution path that is used by incoming BLE frames. From there on, we can follow the frame processing through the firmware.

COMMAND LINE We discover that *Fitbit* firmware exposes an internal command line interface, not secured by authentication credentials, and probably used in the factory for testing and initialization of trackers. The commands available on this interface can be listed with a helper function, which is built around a list of function references and a string that describes their behavior. This convenient mapping gave us the ability to label many functions in the firmware.

FITBIT FRAMES AND COMMANDS The firmware code contains a central switch-case statement, which determines whether to parse a command (starting with `c0`) or a chunk of data (everything else). Starting from this decision, we find the actual command and frame parsing structure, as well as the command response generation routine. After each successful wireless command, an acknowledgment is generated by the firmware. If the tracker receives completely invalid commands, it will stop responding and require a BLE reconnect to restore its operation. We extract a complete list of wireless commands, including date and alarm modifications.

ENCRYPTION We identify eXtended Tiny Encryption Algorithm (XTEA) by its characteristic 64-bit delta string. By comparing open source libraries to the decompiled code, we find that `LibTomCrypt` is used, and compare it to further functions, thereby finding that XTEA is used in Encrypt-then-Authenticiate-then-Translate (EAX) mode [Lib]. With the debugger enabled, we find all the required parameters. In contrast to the first research on *Fitbit* decryption [Sch17], we can generate valid encrypted messages containing a Cipher-based Message Authentication Code (CMAC).

7.2 MODIFYING THE FIRMWARE

We use the *Nexmon* framework [SWH17] to apply modifications to the firmware used by the ARM chip. *Nexmon* allows writing firmware patches in C. These patches can be compiled into a firmware binary, which can then be flashed onto a tracker's microcontroller either via an *ST-Link* debugger or via BLE as explained in Section 3.3. We adapted the *Nexmon* framework to the memory layout in the *Fitbit Flex*. Since the `APP` and `BSL` memory regions are not completely used for code, we use the free areas to introduce changes to the firmware.

Code changes between the *Fitbit Flex* firmware versions are minimal; most functions are not modified. Reverse engineered functions from version 7.64 were easy to port to 7.81 and 7.88. As a proof of concept, we published a hook that multiplies the step count by 100 in the most recent firmware version 7.88 released in October 2017 [Jis19]. The same applies to the *Fitbit Charge HR* firmware, as shown in Figure 15. *Fitbit* is probably using the same codebase within all tracker models.

Even in the *Fitbit Ionic*, which is a smartwatch, has similar parsing behavior of the proprietary BLE protocol [Gro19]. The *Fitbit Ionic* comes with a completely new ecosystem and a different operating system. However, *Fitbit* might still be using the same libraries for protocol parsing. Since we have not yet extracted the firmware, we can only make assumptions here.

Most of our firmware modifications are only available for the *Fitbit Flex*. The essential modifications are described in the following. We disable authentication and encryption in Section 7.2.1, enable GDB support in Section 7.2.2, and build raw accelerometer data access in Section 7.2.3.

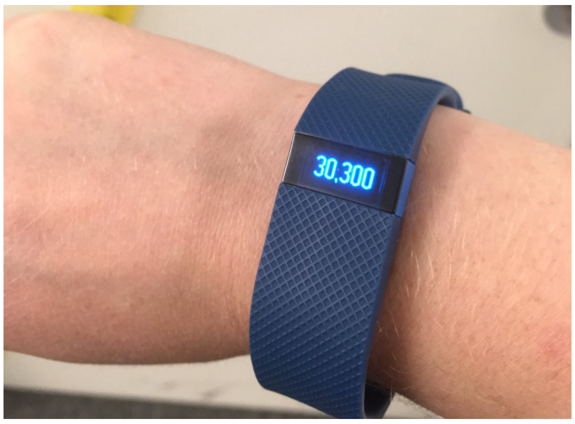


Figure 15: *Fitbit Charge HR* with 30 300 steps that are actually 303 steps.

7.2.1 *Security Mechanisms*

Security is enforced by requiring authentication for specific commands and by enforcing encryption of data. Both can be easily skipped with modified firmware. An attacker who wants to disable these security features as a backdoor can also introduce a hidden BLE command.

DISABLING AUTHENTICATION In Section 3.3.1, a multi-step replay authentication was shown. Patching the firmware allows us to permanently disable authentication, which enables us to skip the authentication step when using, for example, live mode. We identify the function responsible for the authentication by the byte value of the error message issued to signal that authentication is required (that was previously extracted from the *Android* app). The function called before the error is raised returns `true`—if the authentication was successful. Therefore, we can disable authentication by always returning `true` without performing the checks of the original function.

DISABLING ENCRYPTION The method we use to disable authentication can also be applied to encryption. A central function is used in the firmware to check if the communication is encrypted or not. Disabling encryption this way makes it impossible for the server to force a tracker to switch back to encryption mode. Hence, microdumps and megadumps will always be transmitted in plaintext. This poses a substantial risk of privacy leaks.

IMPROVING SECURITY Modifying the tracker's firmware opens the door for more advanced changes to the firmware. For example, instead of completely disabling authentication and encryption, one could introduce more secure algorithms for authentication and encryption.

7.2.2 *Dynamic Analysis and GDB Support*

We gain further insights by dynamically analyzing the firmware during its runtime. The ST-Link debugger can be used as an interface to a GDB server [GNU17]. This allows setting breakpoints as well as memory watchpoints. We use the GDB server, which is included in the *Atollic TrueSTUDIO* [Ato17].

Connecting the debugger with the original firmware is only possible directly after a hard reset of the ARM microcontroller. Continuing execution causes the debugger to disconnect because of the General-Purpose Input/Output (GPIO) port configuration after firmware initialization. Directly after reset, GPIO pins 13 and 14 in group `A` are assigned to `SWCLK` and `SWDIO`, respectively, which enables debugging. When continuing execution, the *Fitbit* firmware reconfigures these pins to analog mode, thereby disabling debugging. To restore the debugging capability, we sought to reconfigure the pins to their original functionality. Finding and disabling the responsible code in the *Fitbit* firmware can prove tedious. We used the *Nexmon* framework to add a function to the firmware, which is executed after initialization and resets the pins to their original purpose. The appropriate addresses that must be used are specified in the reference manual [STM17b].

The reconfiguration of the GPIO pins, however, comes with side effects, as the pins were intended to be used for a different purpose during the runtime of the *Fitbit* code. To minimize the risk of side effects, we implemented an additional BLE command that triggers the reconfiguration of the GPIO pins. Being able to debug the tracker during runtime enables us to verify assumptions made during the static analysis.

7.2.3 *Raw Accelerometer Data Access*

Development platforms with an accelerometer often fail to be light-weight and water-resistant. This makes access to the *Fitbit Flex* accelerometer attractive as an experimentation and development platform. This functionality is implemented in [Han18] and is described in the following.

The *Fitbit Flex* accelerometer is an *LIS2DH*. It is configured to take measurements on all three axes at a rate of 100 Hz. A 2 B value represents each axis acceleration. The firmware copies the three values to a 6 B buffer each time a new measurement is taken.

The normal GATT service that responds to `c0` commands with 20 B values is very inefficient in transmitting this data. Therefore we decide to modify the live mode GATT service. Usually, live mode does not permanently send the current date and step count information, but only if there is new data. To implement an accelerometer mode within the same function, we add a command switch that can either activate live mode or accelerometer mode. In accelerometer mode, the check for new data is skipped, and data is overwritten by the last 6 B accelerometer reading. By sending a separate GATT frame in each round, a measurement transmission rate of 66 Hz can be achieved. This means that some measurements from the original 100 Hz readings are missing, but this rate is still sufficient to measure human activities.

Streaming accelerometer data with GATT is excellent for developing new functions, such as gesture recognition. However, permanent transmissions drain battery power. For getting the full 100 Hz performance, features initially developed with the new GATT service should be ported to the *Fitbit* firmware itself using *Nexmon*.

7.3 BINARY PATCHING CAPABILITIES

Binary patching enables us to access sensors on the *Fitbit Flex* and *Charge HR*, as demonstrated by the accelerometer patch. The *Charge HR* includes a heart rate sensor, and the factory test functions expose a few hints on locations in the binary that handle these sensor readings.

In theory, a completely cloudless implementation of a fitness tracker firmware on a *Fitbit* would be possible. However, we did not reverse engineer all fields in the proprietary dump format, i.e., sleep analysis is still missing. Due to the missing source code of the proprietary firmware, this requires a lot of effort.

Fitbit realized that there are useful applications for lower layer access, and they expose sensor readings such as the accelerometer and heart rate on their newer smartwatches through an API [Fit19a].

MY CONTRIBUTION

Daniel Wegemer ported the Wi-Fi *Nexmon* project for the *Fitbit Flex* firmware [Jis19]. Assembly instructions for both projects are ARMv7 little-endian, so most of the *Nexmon* project could just be copied. I did the reverse engineering of the complete firmware and also located the step counting function. Daniel wrote the first patch to multiply all steps by 100 just before our presentation at 34C3. The step counter hack was quite simple, yet very impressive and even made it to the media. Later, I ported the wireless firmware flashing capabilities to the *Fitbit Charge HR*, and also used the step count example as the first test. Daniel also enabled debugging, which was very useful for Matthias Hanreich later, who did some fuzzing on the *Fitbit Flex* protocol. For his Master thesis, Matthias implemented a function that copies the raw accelerometer data into Bluetooth frames [Han18].

Section 7.1, the Section 7.2 introduction, Section 7.2.1, and Section 7.2.2 were adapted from [Cla+18a, Sec. 6]. Section 7.2.3 is a summary of [Han18].

8 BROADCOM AND CYPRESS BLUETOOTH FIRMWARE

We show why Bluetooth security analysis matters in Section 8.1 and provide an attack model in Section 8.2. Section 8.3 gives an overview of *InternalBlue* use cases and how it differs from standard Bluetooth sniffing setups. Technical details on the architecture of *InternalBlue* are explained in Section 8.4. As a first application, we use *InternalBlue* to test for already known issues within the Bluetooth standard and its implementations in Section 8.5. We then go beyond this and uncover previously unknown implementation issues in popular chipsets in Figure 8.6. We provide an outlook on future work and security patches in Section 8.7.

8.1 MOTIVATION

In this chapter, *InternalBlue* is introduced and used for various Bluetooth analyses. *InternalBlue* is designed not only to support firmware patching but also provides an interactive interface that allows for firmware modification and traffic analysis during runtime. While we ported the patching mechanism from *InternalBlue* for static C code compilation with *Nexmon*, *InternalBlue* does not depend on *Nexmon* per se. Its primary purpose are Bluetooth testing scenarios where one of the devices within a connection is under the control of a researcher. *InternalBlue* is highly flexible and currently runs on various platforms, such as *Linux*, *Android*, *macOS*, and *iOS*. All *Apple* devices running *macOS* or *iOS* are shipped with a *Broadcom* Bluetooth chip supported by *InternalBlue*. This is also true for some *Android* and *Linux* devices, and for *Linux* devices, support can be added by connecting a *Cypress* evaluation board.

All major smartphone vendors support both Bluetooth protocols, classic Bluetooth and Bluetooth Low Energy (BLE). Classic Bluetooth is present in most wireless headsets and some peripherals like keyboards. Its BLE variant has lower hardware and battery requirements and thus is integrated into a majority of Internet of Things (IoT) devices. The application itself does not automatically determine which of these protocols is used, e.g., some keyboards also use BLE and *Android* supports audio over BLE for hearing aids [And19b].

Bluetooth application trends cause many users to enable Bluetooth all the time by default. Recent smartphones do no longer have an audio jack, and many users switched to wireless headphones. Moreover, smartwatches and fitness trackers work best with a permanent connection to a smartphone. Cars provide a smooth handover of calls from smartphones. Overall, Bluetooth provides a great user experience and draws little battery power when running in the background.

8.2 ATTACK SCENARIOS

The application-driven trend to enable Bluetooth all the time is worrisome from a security perspective. While some BLE devices stop sending advertisement packets once

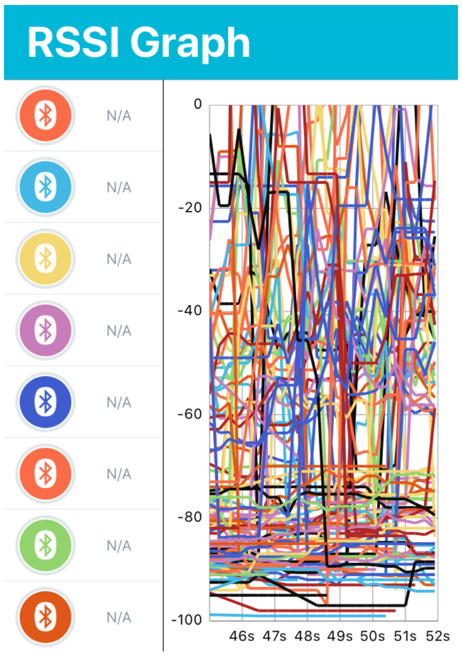


Figure 16: Received Signal Strength Indicator (RSSI) scan output from the *nRF Connect* app at an airport.

they are connected to a smartphone, many continue advertising their identity. Figure 16 shows the output of the *nRF Connect* app while running at an airport, shortly before the app crashes due to the large amount of BLE devices that are around. Even on flight, more than 80 devices were reported in the scan results. In addition to IoT devices, all *Apple* devices broadcast their identity over BLE. Since *iOS 13*, their advertisements no longer include the device name, and all *iPhones* are called `iPhone`. Nonetheless, they remain connectable and will answer to Generic Attribute Profile (GATT) requests.

For classic Bluetooth, the situation is slightly better. Most devices do not permanently advertise their identity, but only when the user initiates pairing. Some devices restrict certain protocol functions prior to successful pairing, such as *L2Ping Echo Replies* with a payload. The actual behavior varies a lot with the operating system configuration. For example, *Xiaomi Mi* smartphones will always advertise their static classic Bluetooth identity as long as Bluetooth is enabled—a behavior that we could not observe on other *Android* smartphones.

If an attacker wants to attack a random device, she will always find various targets in crowded places like airports, train stations, and offices. Due to the limited Bluetooth range, it is feasible to launch an attack against a specific group of people, e.g., next to a company building. While address randomization in BLE makes device tracking and targeted attacks harder, it does not prevent an attacker from connecting to a random device. Moreover, it is also possible to infer the Bluetooth address of a device without advertisements when sniffing traffic with a Software-Defined Radio (SDR) [Mic09].

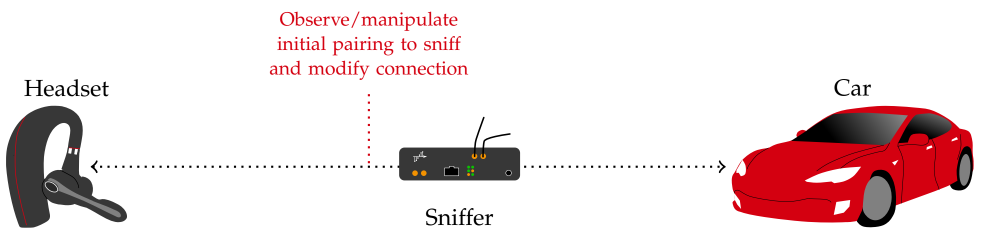
Trust is established on first use, and no Internet connection is required. While this makes it convenient to use in many scenarios, this poses a large attack surface for exploitation. On all *iOS* and *Android* devices we tested, knowledge of the Bluetooth address was sufficient to connect to a device and initiate a pairing request, even if the screen of a smartphone was locked. A lot of messages, including a chip-specific version number, are exchanged before the user of a target device accepts or cancels the pairing request. While this is excellent for user experience, this is worrisome from a security perspective. Pairing can be initiated at any point in time without any third party checking validity of requests.

8.3 BACKGROUND ON BLUETOOTH ANALYSIS

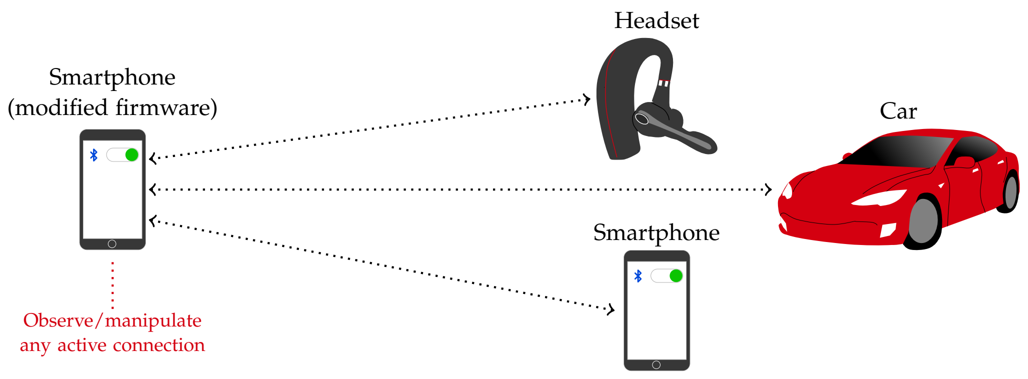
Despite the success of Bluetooth and its comparably large attack surface, there is a lack of functional and open Bluetooth analysis tools. Common Bluetooth sniffing setups use an external device as Machine-in-the-Middle (MITM). Analysis based on binary patching requires changes in the firmware of one of the devices involved in a Bluetooth connection. In the following, we compare these two approaches, which are also depicted in Figure 17.

8.3.1 *MITM Bluetooth Sniffing*

While the PHY of classic Bluetooth and BLE differ, both follow a hopping scheme. Classic Bluetooth hops on 79 1 MHz wide channels, and BLE hops on 40 2 MHz wide channels [Blu19a, p. 350, 2660]. A sniffer is required to either follow the hopping sequence



(a) Bluetooth sniffing setup with passive or active MITM.



(b) Bluetooth analysis with firmware binary patching.

Figure 17: Bluetooth sniffing setups compared.

or acquire and analyze 80 MHz bandwidth at once, to successfully record a Bluetooth connection.

Another challenge for sniffers is encryption. To break encryption, a sniffer needs to overhear the initial pairing. Depending on the Bluetooth version and pairing mode used [Blu19a, p. 257], the attacker also needs to manipulate the pairing procedure actively:

SECURE SIMPLE PAIRING If one of the devices has no input and no output capabilities (i.e., a headset), "Just Works" pairing is used, which is susceptible to active MITM attacks. Otherwise, numeric comparison is used, and both parties are required to ignore a non-matching confirmation number to make an active MITM attack work. Secure Simple Pairing (SSP) was introduced for classic Bluetooth with the 2.1 + EDR specification, and for BLE with the 4.2 specification as *LE Secure Connections*.

LE LEGACY PAIRING has a similar look and feel as SSP [Blu19a, p. 2423], while in fact not being secure against passive MITM attacks. Many IoT devices still use an older Bluetooth version with *LE Legacy Pairing* and smartphones do not indicate to the user which kind of pairing mode is used.

ENTERING 4 DIGIT PIN This legacy method, supported by classic Bluetooth 2.0, was already shown to be broken in 2005 [Sha05]. Nonetheless, it can still be found in some devices with a longer lifetime, such as cars.

Since BLE has a simpler hopping scheme and insecure pairing methods, plenty of very cheap and open-source sniffers exist. One example is the BLE 4.0 *Bluefruit LE Sniffer* for $25 [Ada19], and the *microbit* based tool `btlejack` that costs around $10 [Dam19]. `btlejack` can also follow the new Bluetooth 5 hopping scheme.

One of the first projects that claimed to have solved Bluetooth sniffing challenges is *Ubertooth* [Oss11]. It can even be used to access lower layers since *Ubertooth* is an SDR with extended Bluetooth capabilities. It is still fairly cheap and costs around $120 [Gre19]. However, for classic Bluetooth, *Ubertooth* only supports sniffing of some management frames [SB07]. There is no possibility to sniff encrypted traffic.

For full-featured Bluetooth analysis, expensive commercial products are required. *Ellisys* is a well-known vendor in this sector. Their products are in a price range of $10k–$20k. This is reasonable for anyone involved in Bluetooth hardware development but out of scope for most researchers.

Assuming that an eavesdropper has a working implementation of an MITM sniffer, this technique of sniffing still has the following drawbacks:

- If encryption is used, initial pairing needs to be overheard, and
- if a secure pairing mode is used, the MITM must be active and both parties will be aware of sniffing, and
- data injection requires successful jamming of the original signal, and
- changing small parts in the logic requires re-implementation of the Bluetooth standard within the MITM.

However, a commercial Bluetooth sniffer is likely to work out of the box on any connection between all kinds of devices. This is the primary advantage compared to the approach explained next.

8.3.2 *Bluetooth Firmware Binary Patching*

By design, the host itself is not aware of the internals of the Bluetooth link layer and Physical Layer (PHY). For example, to establish a connection, the Bluetooth host can issue a connection request using the Host Controller Interface (HCI). The Bluetooth controller, which is a hardware chip that runs a Bluetooth firmware, will then try to establish a connection. Establishing a connection might succeed or fail for various reasons. The Bluetooth controller is aware of all steps during connection setup including over-the-air management traffic, and thus, can determine the success or fail reason itself. In contrast, the host is not aware of these internals, and over-the-air management traffic is not forwarded. However, the host gets feedback via HCI if a connection was successfully created or not.

The concept of Bluetooth firmware binary patching is to go beyond HCI. This can be used to sniff any active connection, even if pairing already took place. The Bluetooth controller is aware of encryption and hopping. Thus, all required information is available and already applied correctly by the controller.

Some vendors already expose their firmware functions in a way that makes them easy to patch. This is the case for *Nordic Semiconductor* BLE products, and `btlejack` takes advantage of this to support sniffing, jamming, and injection into active BLE connections [Dam19]. However, what is available there depends a lot on the Application Programming Interfaces (APIs) vendors expose. It is important to note that the `btlejack` setup is used as MITM sniffer, with the drawbacks mentioned in the previous section. `btlejack` requires additional hardware on the host used for sniffing.

Reverse-engineering a chip that supports both, classic Bluetooth and BLE, can go beyond this. For a sniffing setup, the actions performed on the physical and link layer are not changed, but information about them is forwarded to the host using non-standard HCI events or similar. The host can run any operating system–*Linux*, *Android*, *macOS*, *iOS*–as long as it supports the given chip. Software on the host will not be aware of changes inside the controller. Drawbacks of patching Bluetooth firmware to analyze the behavior of other devices are as follows:

- One of the devices in the connection must be under full control (root access), and
- patches depend a lot on the chip vendor and also need to be changed for each firmware revision.

Application scenarios for firmware binary patching are not limited to sniffing and injection of link-layer frames. Anything accessible by the firmware—no matter if intended by the vendor or not—can be changed. The behavior of the chip can be modified to be no longer standard-compliant, for example, to test new security mechanisms or increase performance.

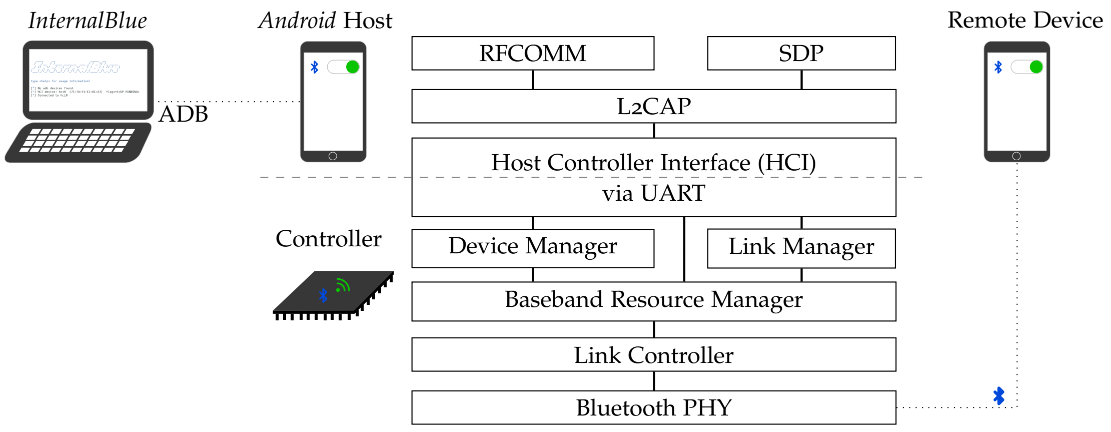


Figure 18: Architecture of the Bluetooth protocol stack with *InternalBlue* control.

8.4 THE INTERNALBLUE FRAMEWORK

InternalBlue is a Bluetooth firmware binary patching framework for *Broadcom* and *Cypress* chips.[1]

Many popular devices use a *Broadcom* Bluetooth chip. *Broadcom* is ranked market leader for wireless communication chips, followed by *Qualcomm* and *MediaTek* [RE18]. Every *Apple* product we tested—*iPad*, *iPhone*, *iMac*, *MacBook*, *Watch*—had a *Broadcom* chip. The only exceptions we have seen so far are the *Apple Watch 5* and the *AirPod Generation 2*, which have a new chip manufactured by *Apple*, internally called *Marconi*. Since 2015, *Apple* sold more than 200 000 000 *iPhones* annually [Sta19]. Moreover, *Samsung* smartphones tend to have a *Broadcom* chip, i.e., *Galaxy S6/S6 edge/S7/S8/S9/Note9/S10e /S10/S10+/Note 10*. Other IoT hardware has a *Broadcom* chip, too, including the *Fitbit Ionic*. The cheap *Raspberry Pi 3/3+/4* platform also has one. The popularity of *Broadcom* Bluetooth chips makes *InternalBlue* applicable to many types of devices, suitable for all kinds of research projects.

InternalBlue currently supports Bluetooth sockets on *Linux* out of the box, patched *Android* Bluetooth drivers, jailbroken *iOS* 12.1.4, and *macOS* including *Catalina*. An overview is shown in Figure 18.

The *InternalBlue* front-end is written in *Python* and currently runs on *Linux* and *macOS* only. This is because some features are easier to maintain on *Linux* and *macOS*, such as displaying traces in *Wireshark* and translating *Assembly* patches. Except for this tool support, *InternalBlue* could be easily ported to run on any platform natively. For *Android* support, *InternalBlue* connects to the *Android* host using Android Debug Bridge (ADB). On *Linux*, HCI sockets provided by *BlueZ* are used directly. The *iOS* solution creates a new socket for the existing serial Bluetooth device. *macOS* does not expose HCI over an official API, but it can be attached to the non-documented `IOBluetooth` interface that is also used internally by *macOS*.

1 *Cypress* acquired the IoT department of *Broadcom* in 2016 [Cyp16], which means that their chips are technically similar. Whenever we refer to *Broadcom* specific techniques, they also apply to recent *Cypress* chips. Another name change is likely because *Cypress* is in the process of acquisition by *Infineon* in late 2019 [Inf19].

HCI is the lowest layer available on the host. An HCI socket only exists locally between host and controller. In hardware, it can be Universal Asynchronous Receiver-Transmitter (UART), Universal Serial Bus (USB), or similar. This differs from other socket types, such as Radio Frequency Communications (RFCOMM). RFCOMM sockets provide a logical link between two Bluetooth hosts that can on top exchange data over a serial protocol. However, the serial service they support has nothing to do with the controller hardware.

InternalBlue only uses the HCI. *Broadcom* extends the HCI with an additional diagnostic message type, used primarily for link-layer sniffing, described in Section 8.4.1. Besides, *Broadcom* provides vendor-specific HCI commands, with many applications including firmware patching, described in Section 8.4.2.

8.4.1 *Broadcom Diagnostics Protocol*

Broadcom chips have a diagnostic protocol to log link-layer management packets for classic Bluetooth and BLE, which use Link Manager Protocol (LMP) respectively Link Control Protocol (LCP). Diagnostic logging is available within various chips. We can confirm it is present in chips with build dates ranging from 2008 to 2018. This means, regardless of the specific firmware version, logging LMP and LCP traffic will always work on a *Broadcom* chip.

The diagnostic protocol is partially supported by *BlueZ* on *Linux* [Hed15], and *Apple*'s *Packet Logger* for *macOS*, which is part of the additional tools for *Xcode*. Both solutions sometimes fail in accurately detecting that a chip has this feature. For example, *Linux* seems to apply a different detection mechanism depending on the chip's interface, which can be USB or UART. By reverse-engineering the *Packet Logger* as well as

Table 2: H4 field bytes indicating UART traffic types.

H4 Type	**Command**	**Direction**
HCI Command	`0x01`	Host → Controller
ACL Data	`0x02`	Host ↔ Controller
SCO Data	`0x03`	Host ↔ Controller
HCI Event	`0x04`	Host ← Controller
Broadcom Diagnostics	`0x07`	Host ↔ Controller

Table 3: Diagnostic logging features and their command codes on *CYW20735B1*. Further diagnostic features are redacted.

Diagnostic Feature	**Command**	**Direction**
LMP Sent	`0x00`	Host ← Controller
LMP Received	`0x01`	Host ← Controller
LCP Sent	`0x80`	Host ← Controller
LCP Received	`0x81`	Host ← Controller
Toggle LMP/LCP Logging	`0xf0`	Host → Controller

the *CYW20735B1* evaluation board, we can now also provide the *Broadcom* diagnostic protocol on *Android* devices.

The diagnostic protocol uses the same UART interface as HCI. By default, it is not interpreted or forwarded by the operating system driver, because it is not using the standard H4 types `0x01`–`0x04`. Instead, a new message type `0x07` is used, as listed in Table 2. Note that *Broadcom* could also use vendor-specific HCI commands and events, which would be compliant to the Bluetooth specification. Instead, they hide this feature from common HCI loggers by inventing the new message type. Moreover, to get those diagnostic messages, the sequence `0x07f001` must be sent, which is a diagnostic message that switches diagnostic logging on. Afterward, LMP and LCP traffic can be received, with traffic type prefixes encoded as listed in Table 3.

The diagnostic protocol has additional features, which allows entering a test mode, getting connection statistics, or reading and writing memory. Our complete reverse engineering of this protocol can be found in [CH19]. Most functionality provided by the diagnostic protocol is also exposed as vendor-specific HCI commands. Since HCI commands have better host support than the diagnostic protocol, *InternalBlue* is using HCI by default if available. Only connection statistics as well as LMP and LCP sniffing are using the diagnostic protocol.

In the context of LMP, one of the interesting vendor-specific HCI commands is *Send LMP PDU* (`0xfc58`). It injects valid LMP traffic into existing connections. To also inject LCP traffic or send invalid data, neither the diagnostic protocol nor vendor-specific HCI commands can be used. Security analysis that requires non-compliant traffic modification requires binary patching, which is explained in the next section.

8.4.2 *Patching Broadcom Controllers*

Broadcom Bluetooth chips store all their firmware in a static Read-Only Memory (ROM). Thus, they require a *Patchram* mechanism to modify ROM, either during setup or during runtime.

PATCHRAM OVERLAY The *Patchram* is a unique mechanism that enables changing the program flow inside ROM. It allows overwriting 4 B slots in ROM. The number of slots varies depending on the chip type, i.e., the *Nexus 5* has 128 slots, while evaluation boards and newer devices have up to 256 slots. 4 B are sufficient to encode a branch instruction on Advanced RISC Machine (ARM). Usually, these branches point to Random Access Memory (RAM) sections, which have fewer restrictions in being written, and contain the actual code for patches. There is no execution prevention for memory that can be written.

SYSTEM PATCHES In principle, the Bluetooth firmware works without any patches. Yet, some *Broadcom* chips even require firmware patches to run on *Linux* host systems. Patches are not only required for compatibility and new features but are also the only way to fix security issues in the Bluetooth firmware.

Broadcom is using the `.hcd` format to ship patches. A `.hcd` file contains vendor-specific HCI commands. The following happens when `.hcd` patches are applied, with the according HCI commands noted in brackets:

- The host starts the patching procedure by entering the so-called *Download Minidriver* mode (`0xfc2e`). This is usually done during driver initialization after reset (`0x0c03`), i.e., *Linux* and *Android* provide a special routine that starts this mode [Lin19].
- The controller enters *Download Minidriver* mode. This has two effects. First, execution of all other threads and interrupts is paused. Second, only a subset of HCI commands is accepted. Both are required to ensure a smooth firmware patching process.
- All HCI commands of the `.hcd` file are applied. This file includes multiple *Write RAM* commands (`0xfc4c`) that write to RAM and *Patchram*. The final command executes a *Launch RAM* (`0xfc4e`) to the non-existing address `0xffffffff`, which quits *Download Minidriver* mode and continues executing the normal Bluetooth threads.

It is important to note that the newest chips, which are connected via Peripheral Component Interconnect Express (PCIe), use a different file format ending with `.bin`. While the `.hcd` file contains HCI commands that are applied one after each other, the `.bin` file is copied in one piece. Its header contains checksums, lengths, and locations for the memory areas the data is copied into by the controller. On the *iPhone 11* series, these memory regions are two areas for patch code and one area for configuration. While the *iOS* `bluetoothd` takes care of appropriately copying this data, the Bluetooth chip itself is still using the same *Patchram* mechanism internally.

Surprisingly, `.hcd` and `.bin` patch files do not contain any security mechanisms like vendor signatures or encryption. There are multiple ways to obtain a patch file. They are often shipped with the operating system. In *Android 9*, they are readable by the non-privileged local user and located in `/vendor/firmware/`. The `.bin` and `.hcd` files can be extracted from *iOS* images without jailbreak, and be accessed on any *macOS* system. The *iPhone 11* firmware has 4 B checksums but no signatures, and the PCIe firmware for the newest *MacBook* only added a signature for host-side verification.

It is possible to generate custom `.hcd` files. *Cypress* provides the *WICED Studio* tool suite to write patches for evaluation boards, which compiles C code into `.hcd` files. Within *Nexmon*, we also integrate Bluetooth chips and support C code but are not limited to evaluation boards [CS19].

LIVE PATCHES It is possible to issue the *Write RAM* (`0xfc4e`) and *Launch RAM* (`0xfc4c`) any time outside of the *Download Minidriver* context. This is very useful when developing code since no restart of the Bluetooth driver is required. However, it is also less reliable and potentially unstable. For example, applying a *Patchram* patch to a function that is currently being used might crash the firmware. Moreover, executing something via *Launch RAM* that relies on switching threads might also crash the firmware. The actual implementations of *Launch RAM* differ a lot between chips, and all seem to have their individual bugs. On the *Nexus 6P*, *Launch RAM* is almost not usable: If any other HCI command is issued after *Launch RAM* within the next 6 s, the firmware will crash.

SYMBOLS AND DOCUMENTATION When writing a patch for an existing firmware binary, the most valuable information is the position of symbols. Symbols are function and variable names. While the names potentially stay the same over various firmware revisions, their position will change each time the firmware is compiled. The compiler tends to keep libraries in one non-fragmented region, which means that symbols of one library will usually be nearby.

Without any prior knowledge about the firmware, reverse engineering of symbol positions is not trivial. Bluetooth firmware implements the Bluetooth specification [Blu19a], which can be used to identify standard-compliant functions. Vendor-specific HCI commands are partially documented in datasheets. We spent months with only this information available to reverse engineer functions in the *Nexus 5*, prior to finding other sources of symbols.

WICED Studio, available for some Bluetooth evaluation boards, contains all symbols in a `patch.elf` file for the supported boards. These symbols make the firmware almost as readable as C code with meaningful function and global variable names—but without comments and structs. From *WICED Studio 6.2* we extracted symbols for the *CYW20719* and *CYW20735* boards, as well as the *BCM20703A2* firmware. The latter is included in a different format and actively used in MacBooks between 2015 and 2016. *Cypress* switched to *ModusToolbox* for newer evaluation boards. *ModusToolbox 1.1* still contains a `patch.elf` for the *CYW20819* chip. This means that we have correct symbols for various firmware compiled between October 22 2015 and May 22 2018. Matching unknown functions of a firmware without symbols to firmware with symbols is way more straight forward than using the Bluetooth standard and datasheets.

Correct symbols still lack documentation, but some of the libraries are documented. The most important documentation is for the firmware's underlying operating system, which is *ThreadX* [Exp19]. While older firmware versions contain a copyright note, this was removed from newer firmware. For example, the *BCM20702A1* firmware contains the copyright notice `Copyright (c) 1996-2003 Express Logic Inc. * ThreadX ATMEL/Green Hills Version G4.0b.4.0c *`. Since older firmware versions lack a build date, we do not know when exactly the copyright notice was removed. Newer firmware versions use similar function names, meaning *ThreadX* is still the operating system.

8.5 TESTING FOR KNOWN SECURITY ISSUES

Inside the *InternalBlue* Command Line Interface (CLI), we implement the feature to connect to any device by its Media Access Control (MAC) address even without prior pairing, as described in Section 8.5.1. As a demonstration of security research, we implement device testing for the so-called *Niño* attack—pairing devices without input and output—in Section 8.5.2 as well as an attack on Elliptic Curve Diffie-Hellman (ECDH) device pairing key exchange in Section 8.5.3. After our first release of *InternalBlue*, Antonioli, Tippenhauer, and Rasmussen used it to test an attack against the Key Negotiation of Bluetooth (KNOB), as shown in Section 8.5.4.

Implementations of the *Niño*, ECDH, and KNOB tests are not contained as *InternalBlue* commands but as example files[2] that use functions of *InternalBlue* as a library. The smartphone itself performs the attacks and is already involved in the connection, and the attack is not carried out as MITM. Therefore, these implementations are only suitable for testing how other Bluetooth stacks react to *Niño*, ECDH, and KNOB attacks—we do not provide active attacks on a victim's established connections.

2 `https://github.com/seemoo-lab/internalblue/tree/master/examples`

8.5.1 *Establishing Connections to "Invisible" Devices*

Bluetooth devices can be invisible during scanning but still accept connections. Accepting connections in the background is required to support automatic reconnect an already paired smartphone and headset without user interaction. They know each other's MAC address from the previous pairing, and the headset tries to connect to the smartphone once it is turned on. Surprisingly to most users, initiating a connection request to a device does not require a previous pairing. All that needs to be known is a valid MAC address [Mic09]. *InternalBlue* can list the array of all current connections and initiate new connections with the command syntax `connect de:ad:be:ef:00:00`.

The existence of a connection inside the firmware's list is required to send LMP packets. This list differs from the list of known devices on the host, and it is just meant to manage connection states of ongoing connections. A connection is not required to be established to appear in this list—it is sufficient to initiate a connection. If a connection cannot be established successfully, it will remain around half a minute inside the firmware's list until it is deleted. This allows sending arbitrary LMP commands to arbitrary devices without successful pairing.

8.5.2 *No Input No Output Pairing*

Bluetooth provides pairing based on ECDH key exchange with Secure Simple Pairing (SSP) to be secure against passive and active MITM attacks. Security against active MITM is provided by visual number comparison or entering a number shown on one device with a keyboard on the other device. This protection requires both devices to have Input-Output (IO) capabilities, which is often not the case for IoT gadgets and headsets. The fallback option in case of missing IO capabilities is to perform key exchange without active MITM protection, the so-called "Just Works" mode. The Bluetooth 5.1 specification explicitly marks this as being insecure but does not suggest showing any warnings to the user. The exact implementation is up to the product manufacturer [Blu19a, p. 260].

Exchange of IO capabilities supported by the devices takes place before pairing. Since there is no established root of trust between them at this stage IO capabilities are prone to active attacks themselves—an active MITM can replace capabilities by no input no output, also known as the *Niño* attack [HH07].

We patch the SSP state machine and claim our *Nexus 5* has no display and no keyboard. Our question is if vendors accept a "Just Works" pairing without asking a "yes/no" message or warning the user to check if the device they are pairing to has no IO capabilities. We find that neither *Android* nor *iOS* devices warn users about missing IO capabilities, even though our *Nexus 5* advertises itself as smartphone, which should always have IO capabilities. However, users are required to confirm the pairing with a "yes/no" message.

Figure 19 shows a modified *Nexus 5* pairing with an *iPhone 6*. The only modification in the SSP state machine function located at `0x303D4` is to always set the IO capabilities to `0x03` in variable `0x20387D`, with `0x03` resembling no input and no output according to the Bluetooth 5.1 specification [Blu19a, p. 865]. Algorithms within *Nexus 5* are not coded to consider the case of no longer having a display with such a change and continue passing a code to *Android*, which actually works because keys are derived either way similarly. *iPhone 6* simply displays a "yes/no" message and pairing is achieved. Yet, operating systems seem to cache some of this information locally, making tampering with IO capabilities hard if not applied within each pairing. To ensure a *Niño* attack works, an attacker should be present during the initial pairing. In our experiments, users were not be displayed any warnings if a *Niño* attack fails due to the detection of inconsistent IO capabilities, but the pairing was aborted.

8.5.3 *ECDH Device Pairing Vulnerability Scan*

Another application of *InternalBlue* is a security test for the fixed coordinate invalid curve attack *CVE-2018-5383* [Eli18]. This attack affects device pairing based on Elliptic Curve Diffie-Hellman (ECDH), both SSP used in classic Bluetooth and LE Secure Connections (LE SC) used in BLE. The underlying protocol vulnerability exists if a Bluetooth implementation does not check for invalid curve parameters in the key. Since the Bluetooth 5 specification did not consider countermeasures against this attack mandatory, vulnerable implementations are widespread. Testing for this attack is not straightforward, though, as the authors did not release any tools and the attack requires special hardware.

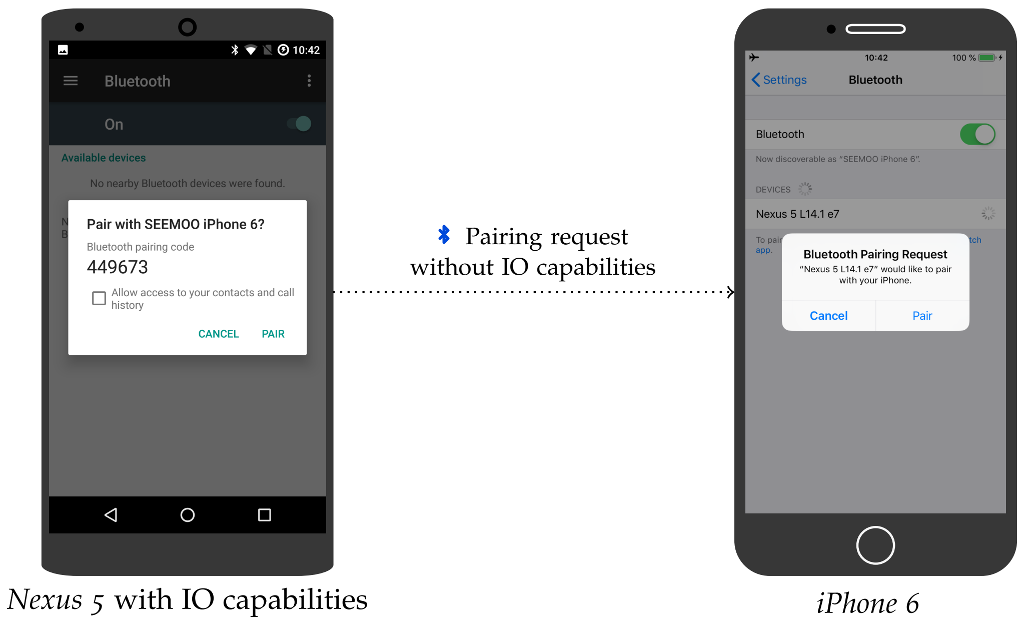
We implement the semi-passive fixed coordinate invalid curve attack, which zeroes the y-coordinates of both public keys belonging to the *InternalBlue* smartphone and the device under test. If the device under test does not perform an invalid curve parameter check, this semi-passive attack would be successful in 25% of all device pairing attempts under the original attack's assumptions [Eli18]. In contrast, the *InternalBlue* success rate is raised to 50% by setting an even smartphone private key, which is possible since the attack model is not a wireless link under control but the actual smartphone initiating a connection. Although not performing this attack as MITM, testing other smartphones is fast and portable with our solution.

InternalBlue could also help to fix this vulnerability in older *Broadcom* Bluetooth chips. Applying security patches this way is feasible if a vendor is no longer supporting old chips or reacts slowly in publishing updates.

8.5.4 *KNOB Attack Test*

After SSP, every time two devices connect, they perform a key negotiation. During this key negotiation, the key entropy can be reduced from 16 B to a lower value upon request of the master or the slave. This attack is compliant with the Bluetooth 5.1 specification [Blu19a, p. 598]. The attacker has to inject a request to reduce the entropy, which is called `LMP_encryption_key_size_req`. Note that despite this name the key length remains 16 B, but the entropy can be reduced to as low as 1 B on many devices. Hence, the effective key size will be 1 B. This attack, called Key Negotiation of Bluetooth (KNOB) attack, has been discovered and tested recently [ATR19]. The authors tested this attack with our first *InternalBlue* release, and most devices accepted the minimal key size.

After the publication of the initial KNOB paper, we ported the attack from the *Nexus 5* to further devices, including the popular *Raspberry Pi 3/3+/4* and the more recent *Samsung Galaxy S8*. As shown in Figure 20, this attack is detected on recent *iOS* devices.



No nearby Bluetooth devices were found.

Pair with SEEMOO iPhone 6?

Bluetooth pairing code

449673

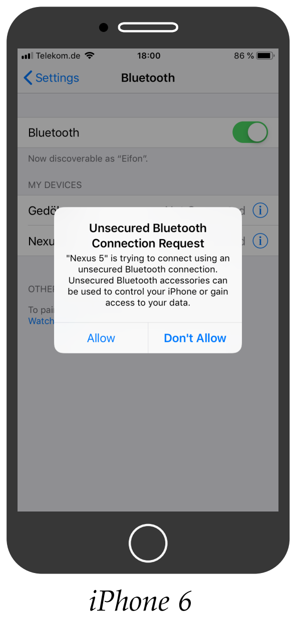
Allow access to your contacts and call history

Now discoverable as "SEEMOO iPhone 6".

Bluetooth Pairing Request

"Nexus 5 L14.1 e7" would like to pair with your iPhone.

Figure 19: "Just Works" pairing on *iOS* 12.1.4.



Now discoverable as "Eifon".

Unsecured Bluetooth Connection Request

"Nexus 5" is trying to connect using an unsecured Bluetooth connection. Unsecured Bluetooth accessories can be used to control your iPhone or gain access to your data.

Figure 20: KNOB attack during connection setup with a previously paired device since *iOS* 12.4.

Users can still decide to accept the connection, but the default option is to not allow the connection.

8.6 DISCOVERING AND FIGHTING NEW VULNERABILITIES

While exploring command handlers in the *Broadcom* firmware, we have found a severe vulnerability affecting a variety of devices, as described in Section 8.6.1. To analyze the vulnerability's impact, we implement runtime testing and emulation solutions, as detailed in Section 8.6.2. Besides, these can also be used to detect further bugs or simply to locate and dissect more functions and realize more features. In Section 8.6.3, we come up with a solution that can help against even more attacks.

8.6.1 *Remote Code Execution Vulnerability*

In the following, we explain how we discovered *CVE-2018-19860*, a vulnerability allowing over-the-air attackers to execute a subset of functions within a large set of *Broadcom* firmware. The vulnerability only requires the device under attack to have Bluetooth enabled, no previous pairing or visibility to the attacker is required.

8.6.1.1 *Vulnerability Description*

While dissecting the LMP dispatcher and analyzing its functions, we found vendor-specific commands. The Bluetooth standard only defines vendor-specific commands on the HCI layer, which can only be executed locally on the host as superuser. However, *Broadcom* uses LMP opcode `0x00` for Broadcom Proprietary Control Signaling (BPCS) mapped by `lm_BPCS_getLmpInfoType`. Exploring this unusual LMP handler, we discovered that it accepts 256 input values in the field following opcode `0x00`. *Broadcom* only intended to implement BPCS commands `0x0000`–`0x0005`. Because of a missing range check commands `0x0006`–`0x00ff` interpret memory located after the intended BPCS handler table similarly.

As BPCS is a proprietary feature, a *Broadcom* chip only answers these requests if they originate from a connection to another *Broadcom* device. The attack can also be launched from a *Cypress* evaluation board, if the vendor identifier is changed from *Cypress* (`0x0131`) to *Broadcom* (`0x000f`) in the `LMP_version_res`.

Compilers tend to put similar library functions close to each other. Even though compiling is hard to predict and contents are different for each *Broadcom* chip, handler tables are very likely to be put after each other. On *Nexus 5* other handler tables are indeed put after the BPCS handler table and include an interesting selection of HCI commands shown in Figure 21 that should never be launched from a remote host, such as `Launch_RAM`. Function arguments are passed in different only partially controllable registers when called over this path, for example, `Launch_RAM` is executed but always launches an invalid address. Execution of invalid commands or accessing invalid memory just crashes the Bluetooth firmware and has no further side effects. On some operating systems, the driver restarts Bluetooth automatically within a minute, but even in this case, ongoing connections are interrupted. In general, chances of crashing the *Broadcom* firmware are high.

Interpreting out of bounds commands in Figure 21 also results in extremely long LMP packets, i.e., the handler for command `0x0095` has a length of 219 bytes. When trying to use *InternalBlue*'s `sendlmp 00 95`, the packet seems to be sent according to the monitoring hooks but is never actually sent because the LMP packet buffer is limited to 32 bytes. When attacking other smartphones with a *Nexus 5*, the attacker first needs to patch the vulnerable BPCS handler locally to change the length and option fields. A *Nexus 5* under attack will not wait to receive all 219 bytes and interprets the LMP packet with the wrong length.

BCM4339 on *Nexus 5* is a combo chip using a shared antenna for Bluetooth and Wi-Fi. Further investigating crashes on *Nexus 5* shows that they result in 2–5 s interrupts on 2.4 GHz Wi-Fi only. 5 GHz Wi-Fi is not affected. Similar Wi-Fi interrupts also happen during normal Bluetooth pairing. However, they do not seem to happen when regularly exchanging data. The default configuration of many systems is to restart Bluetooth after a crash, and an attacker can reduce Wi-Fi performance multiple times by attacking again when the Bluetooth of the victim reappears. Analyzing possible causes for this behavior is complicated—it could range from incorrect sharing of the antenna up to *Android*-specific issues.

8.6.1.2 *Affected Devices*

Vulnerable devices are easy to detect as they do not answer BPCS commands with `LMP_not_accepted`. Very old *Broadcom* chips do not implement these commands at all and are not vulnerable. *Broadcom* did not share a list of vulnerable devices after we reported the issues, but we found an interesting selection listed in Table 4 by testing de-

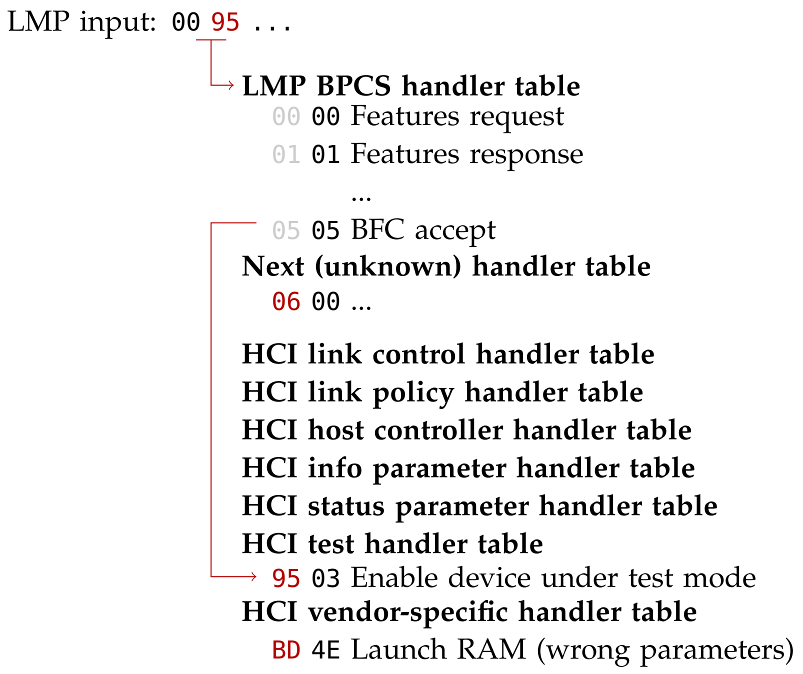

LMP input: 00 95 ...
LMP BPCS handler table
00 00 Features request
01 01 Features response
...
05 05 BFC accept
Next (unknown) handler table
06 00 ...
HCI link control handler table
HCI link policy handler table
HCI host controller handler table
HCI info parameter handler table
HCI status parameter handler table
HCI test handler table
95 03 Enable device under test mode
HCI vendor-specific handler table
BD 4E Launch RAM (wrong parameters)

Figure 21: Jumping from one handler to another, example handlers picked from *BCM4339* firmware.

vices we had access to. A major version number of 5 or lower seems to be secure against our attack but corresponds to the outdated Bluetooth 3 standard [Blu19c]. Only some of the chips with major version 8 were vulnerable, which corresponds to Bluetooth 4.2.

Before pairing, devices exchange their version number within a `LMP_version_req` and `LMP_version_res` to know which version of the Bluetooth specification is supported by each other. This request also includes a subversion that further specifies the firmware version running on the chip. This gives the attacker hints which payload could execute meaningful functions.

We extracted firmware from various vulnerable and non-vulnerable devices to further track down when the function `lm_BPCS_getLmpInfoType` was fixed and if handlers located behind are indeed HCI or similar interesting functions. Firmware internals of those versions that give most insights about patching are listed below.

RASPBERRY PI 3 / *BCM43430A1* is vulnerable, and its build date is June 2 2014. It has a few callback function tables located in the vulnerable memory region, but most of the memory is filled with encryption constants.

RASPBERRY PI 3+ / *BCM4345C0* is *not* vulnerable and its build date is August 19 2014.

MACBOOK PRO 2016 / *BCM20703A2* is vulnerable and its build date is October 22 2015. Various callback and HCI function tables are located in the vulnerable memory region.

We assume that *Broadcom* internally discovered this bug in summer 2014, but was not aware of its criticality at all when adding the missing BPCS opcode check. Moreover, a newer build date does not necessarily mean that a firmware has the newest library versions, as the *BCM20703A2* example shows.

As firmware development cycles are quite long—even the evaluation board is shipped with a one-year-old firmware—devices released in 2016 are still vulnerable. We estimate

Table 4: Devices vulnerable to *CVE-2018-19860*.

Device Name	Chip	Version	SubVersion
MacBook Pro 13" mid 2012	*BCM4331*	6	8859
iPhone 5	*BCM4334*	7	16653
iPhone 5s	*BCM4334*	7	8707
Xperia Z5	*BCM43xx*	7	8975
Nexus 5, Xperia Z3, Samsung Galaxy Note 3	*BCM4339*	7	24841
Raspberry Pi 3	*BCM43430A1*	7	8713
Huawei Honor 8	*BCM4345*	8	24857
iPhone 6	*BCM4345*	8	16649
MacBook Pro 13" early 2015	*BCM20703A1*	8	8609
MacBook Pro 13" early 2015 #2	*BCM20703A1*	8	8614
Fitbit Ionic	*BCM20707*	8	8715
MacBook Pro 2016 A1707	*BCM20703A2*	8	8752
MacBook Pro 2016	*BCM20703A2*	8	8774

around half of the devices with *Broadcom* Bluetooth chips actively used in December 2018 to be vulnerable.

8.6.2 *Firmware Emulation and Fuzzing*

Exploring abilities gained with the BPCS vulnerability is hard. To analyze crash causes and impact of non-crashing functions, we develop a toolchain that allows analyzing crashes, debugs functions dynamically on the chip, and explores the information flow throughout the firmware based on emulation.

8.6.2.1 *Software Tracepoints*

Functions are reached over long call graphs with parameters depending on many dynamic inputs. On the *Nexus 5*, the function `lm_HandleLmpReceivedPdu`, which processes LMP packets, is located at `0x3f3f4`.

The *InternalBlue* command `tp add 0x3f3f4` adds a tracepoint to this function. Each tracepoint is realized as *Patchram* hook. Once a tracepoint is called, it is deleted from *Patchram* to prevent the firmware from being stuck in tracepoints while debugging. A tracepoint imitates the behavior of the original *Broadcom* fault handler contained in the firmware and dumps all register contents as well as stack and heap but continues operation afterward. Multiple tracepoints can be observed in a row, e.g., to check the behavior of multiple functions called for a given input.

Example output for `lm_HandleLmpReceivedPdu` is shown in Listing 1. The link register points to a function *Broadcom* named `lm_handleLmpMsg`, `r0` points to `lm_curCmd`.

8.6.2.2 *Emulation with Unicorn and Radare2*

To investigate the LMP handler remote code execution, we use registers from a tracepoint of `lm_HandleLmpReceivedPdu` but install the second tracepoint in the handler for BPCS itself to dump the corresponding stack and heap contents of a BPCS payload. With a combination of *Unicorn* and *Radare2*, we modify a *Python* script that emulates `lm_HandleLmpReceivedPdu` [pan19; QV19; Rei16]. BPCS commands are passed by changing the corresponding blocks in the stack and heap memory region. Emulation of an LMP command stops if the end of `lm_HandleLmpReceivedPdu` is reached or a timeout is exceeded.

A call trace for `lm_HandleLmpReceivedPdu` is given in Listing 2. In this case, the function is passed an LMP packet containing `0x000a`. Following handler tables as in Figure 21, position `0x0a` in the misinterpreted BPCS handler table calls a `NULL` pointer that is then interpreted as a function address, and ROM contents at `0x00000` are executed.

```
[*] Tracepoint 0x3f3f4 was hit and deactivated:
    pc:  0x0003f3f4     lr:   0x00008c33    sp:  0x0021734c     cpsr: 0x00000000
    r0:  0x00200478     r1:   0x002179a8    ...                 r12:  0x40000000
```

Listing 1: Tracepoint in `lm_HandleLmpReceivedPdu`.

This trace is still very short because it crashes immediately. However, traces can become very complex and hard to analyze.

8.6.2.3 *Call Trace and Memory Interpretation*

The emulator recognizes branches and extracts addresses when entering a new block after a branch to generate a detailed call trace. Moreover, it dumps memory after finishing execution, which can later be compared to the original state. BPCS commands that depend on arguments passed to the handler can be located by emulating the same commands multiple times with different inputs and then searching for traces with different execution paths or by memory differences.

8.6.2.4 *Crafting Exploits: Turn Remote Devices Into Jammers*

One function standing out in memory analysis changes the content of a very high memory section, which turns out to be responsible for test mode configuration. Further investigation reveals that the LMP payload `0x0095` calls the HCI handler function enabling device under test mode. Without being embedded into HCI logic, the function gets executed but never appears in *HCI Snoop Log* on the host. Therefore, a host under attack will not be able to filter or observe this event.

After enabling test mode with the malicious payload, `LMP_test_activate` and `LMP_test_control` can be used to run tests. The master controls test mode, and the slave is the device under test. Each test is performed for multiple seconds. During a test, master and slave usually hop over all channels defined in their Adaptive Frequency Hopping (AFH) configuration. In conjunction with `LMP_set_AFH`, AFH can be disabled on the attacked device; thus, causing it to jam a selected frequency, which could be pilots of a Wi-Fi signal the attacker wants to block. The `LMP_set_AFH` payload must contain a valid Bluetooth clock to enable jamming of a single frequency, which requires writing an assembly patch. A sequence of frames successfully turning a *Nexus 5* and *Xperia Z3*

```
0x3f3f4, 0x3f400, 0x42c04, 0x42c0a, 0x3f406, 0x3f40e, 0x3f418, 0x4a868, 0x4a87a,
0x3f41e, 0x3f426, 0x3f44c, 0x3f44e, 0x3f34c, 0xd30d0, 0xd3152, 0xd315e, 0xd3188,
0xd3192, 0xd3130, 0x3f456, 0x3f458, 0x3f464, 0x00000,
invalid memory 0x661e1000, invalid memory 0x65f7f3c0
```

Listing 2: Call trace starting at `lm_HandleLmpReceivedPdu`.

```
# LMP_set_AFH to disable hopping
sendlmp 60 0000000000ffffffffffffffff0000
# Enable test mode via exploit
sendlmp 00 95
# LMP_test_activate
sendlmp 56 00
# LMP_test_control, TX frequency 2433 MHz
sendlmp 57 545575755555555255
```

Listing 3: Remote jammer attack.

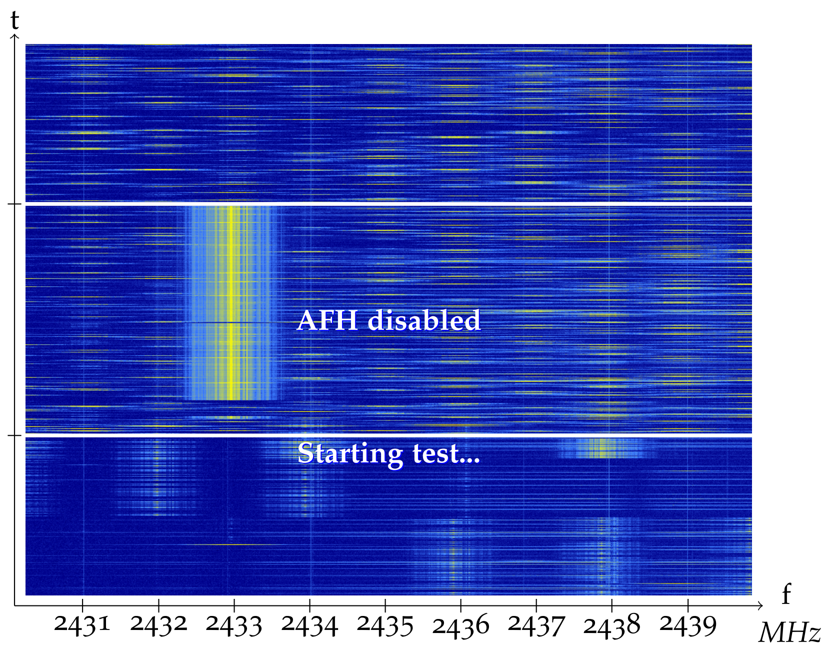


Figure 22: Device under test mode exploit.

into a remote jammer is given in Listing 3. Note that `LMP_set_AFH` is not under perfect control of the attacker without passing a proper Bluetooth clock. AFH is not stopped on the attacking device but the device under attack nevertheless temporarily accepts the AFH configuration. With the presented payloads, the device under attack keeps hopping on all Bluetooth channels most of the time, as shown in Figure 22.

8.6.3 *MAC Address Filter*

As a basic defense against devices injecting malicious frames, a MAC address filter can be used. Typically, users only use a few devices they trust, such as a headset. We implement a whitelist inside `lm_HandleLmpReceivedPdu`. Untrusted devices are rejected within LMP—an attacker not imitating a valid MAC address cannot establish connections or even tamper with any of the LMP handlers or protocols above. To successfully imitate a MAC address, attackers need to analyze traffic with an SDR for a while to calculate the target address from demodulated signal chunks [Mic09]. In contrast to Wi-Fi, Bluetooth chunks do not contain full MAC addresses, therefore guessing these is harder than in Wi-Fi.

Currently, the MAC address filter is implemented as a simple *Python* proof of concept executed at runtime by using *InternalBlue* as a library. It can be integrated into more user friendly solutions, such as an *Android* app that automatically generates a permanent HCD file allowing only connections from devices that were successfully paired at the time of its generation.

A MAC address filter is no longer required to defend against *CVE-2018-19860*, as most vendors have rolled out patches. *Broadcom* did not include us in the feedback loop after informing vendors in December 2018, but we tested previously affected devices and can confirm that our vulnerability was fixed in *iOS* 12.1.3. The *macOS* update fixing the vulnerability increases the LMP subversion by one.

8.7 RESPONSIBLE DISCLOSURE AND OUTLOOK

Broadcom and *Cypress* chips are shipped in hundreds of millions of devices annually. This popularity comes with responsibility. Indeed, *Broadcom* seems to do a lot of security testing. Issues found in older ROMs were often patched in newer ROMs. However, we observed these patches are not included in the corresponding `.hcd` patch files for older chips. The supply chain for patches after releasing a ROM firmware image is very complex. Usually, the ROM build date is a year behind the product release date, i.e., the *iPhone 7* ROM is from September 2015, while the product was released in September 2016. Even evaluation boards lag a year behind. Low-cost products like the *Raspberry Pi 3+* are multiple years behind. Any issue found during the development cycle must be fixed through `.hcd` files—this not only applies to security but any feature. Once the product is live, *Broadcom* still needs to inform all their customers about patches and provide these. Applying a patch means rolling out a new `.hcd` file with the next operating system update. Patching capabilities inside the chip are limited by the amount of *Patchram* slots and free *RAM*, which means that at some point, the vendors must choose between patches. Vendors might not prioritize patches for security issues that are not publicly known. In addition, since `.hcd` files are plaintext, the actual vulnerability might just leak by patching a not yet publicly known security issue.

From firmware build dates, we can deduce that *Broadcom* found *CVE-2018-19860* internally. Probably they were not aware of the vulnerability's criticality. They likely informed vendors at the end of November or the beginning of December 2018. *Apple* silently patched the issue in January 2019 in *iOS* devices, their first advisory mentioning this Common Vulnerabilities and Exposure (CVE) is a *macOS* update from July 2019 [App19b]. They only have a few chips in their ecosystem, and it is comparably easy for them to roll out custom `.hcd` patches. In contrast, *Google* mentions the issue in their May advisory [And19a], and usually, these patches will only be applied afterward by affected vendors like *Samsung*. We are not aware of any patches for *Linux*—in that ecosystem, it is usually hard to find up-to-date `.hcd` files.

The security impact of Bluetooth is often underestimated. Users allow headsets to share their contacts and *Android* enables users to set up a paired device for a restricted unlock. Assuming the Bluetooth driver on the host has security issues, an attacker could escape to the host with high privileges, as was the case in recent attacks on Wi-Fi [GP19; Ang19]. On systems with a *Broadcom* Bluetooth/Wi-Fi combo chip, we ran first experiments on a *Nexus 5* that show an attacker with control over Bluetooth can slow down or disable Wi-Fi and vice versa. IoT devices that run their application within the *Bluetooth* chip have a special risk. This setup is typical of how *WICED Studio* enables IoT applications on *Cypress* evaluation boards. On such systems, an attacker can directly escalate into the IoT application without permission restrictions.

MY CONTRIBUTION

Credit to the first version of *InternalBlue* goes to Dennis Mantz, who chose Bluetooth binary patching for his Master thesis topic [Man18]. He was initially supervised by Matthias Schulz, who handed over supervision to me. Dennis wrote clean code and with good documentation, thereby enabling others to start with Bluetooth analysis and binary patching. Before I found symbols for the *Cypress* evaluation kits, we both invested months in reverse engineering functions according to the Bluetooth standard in the *Nexus 5* firmware. With these initial symbols, we were able to write Proof of Concepts (PoCs) for known Bluetooth vulnerabilities. Just after submission of Dennis' thesis, an attack on the ECDH key exchange was published, which he then implemented as a first demo and used for security testing of other Bluetooth chipsets. Reverse engineering *Nexus 5* symbols lead me to *CVE-2018-19860*.

A publication on all the security-relevant findings was published at *MobiSys* [Man+19]. Later I found that LMP monitoring and injection capabilities implemented as Assembly patches by Dennis were already supported by a diagnostics protocol [CH19].

After finishing his thesis, Dennis still had some time in continuing development and research with *InternalBlue*. We gave talks together at *35C3*, *TROOPERs*, *REcon*, and *MRMCD* [CM18; CM19a; CM19c; CM19b].

Analysis of *Broadcom* Bluetooth chips is an ongoing project. The latest features include a port of *InternalBlue* to *iOS* by Dennis Heinze, and to *macOS* by Davide Toldo. Together with Michael Spörk and the team at TU Graz, we added link-layer feature extraction to improve the reliability BLE connections [Spö+20]. Jan Ruge developed an emulation and fuzzing framework *Frankenstein* that can run Bluetooth firmware binaries [Rug19]. He discovered further arbitrary code execution vulnerabilities. Together with Francesco Gringoli, I exploited Bluetooth coexistence with Wi-Fi (*CVE-2019-15063*), which even caused a full device reboot on various *iOS* and *Android* devices.

External entities are also interested in *InternalBlue*. We are involved in responsible disclosure processes with *Broadcom*, *Cypress*, *Apple*, *Samsung*, and *Google*, as well as collaborations with other universities that are interested in performance improvements and security analysis.

Section 8.5 is based on [Man+19, Sec. 4]. Figure 8.6 is adapted from [Man+19, Sec. 5]. The tables in Section 8.4.1 are based on [CH19].

Part IV

FUTURE WIRELESS STANDARDS

New wireless standards are highly relevant for security research, as they might enable new security mechanisms. Especially the lower layers change with wireless technology. In this part, we focus on wireless Physical-Layer Security (PLS) for Visible Light Communication (VLC) and relate it to findings for mmWave communication. Chapter 9 shows that VLC and mmWave connections can be eavesdropped in real-world setups despite their directionality and short-range. Applicability of PLS techniques to VLC is discussed in Chapter 10.

9 EAVESDROPPING VISIBLE LIGHT AND MMWAVE CONNECTIONS

This chapter is structured as follows. In Section 9.1, we explain why it is crucial to research eavesdropping in narrow-beam Visible Light Communication (VLC) and mmWave transmissions. Our VLC testbed is explained in Section 9.2. We perform practical VLC measurements in Section 9.3. The findings are compared to a similar mmWave eavesdropping setup in Section 9.4. Section 9.5 concludes our measurement results.

9.1 MOTIVATION

VLC ( )transmits data using everyday light sources. Light is blocked on its way by walls and almost any obstacle. Humans can intuitively observe the VLC transmission range. In contrast, Radio Frequency (RF) transmissions can pass through multiple walls depending on the material and frequency. For a human without measurement equipment, it is hard to predict and impossible to infer the range of an RF transmission. For example, a standard 2.4 GHz Wi-Fi ( ) might be receivable outside a house or apartment where it is located, yet sometimes it will be hard to receive in a neighboring room inside the same building.

The mmWaves ( ) we experiment with operate in the 60 GHz range. Due to their small wavelength, they behave in a quasi-optical manner and are easily blocked by obstacles. Moreover, to increase transmission range to a couple of meters, they need directional transmission.

Both technologies, VLC and mmWaves, are claimed to be hard to eavesdrop due to blockage and directionality [Pov11; Fre13; Yan+15]. We consider passive attackers hiding in Non-Line-of-Sight (NLOS) paths that attempt to intercept the communication stream by exploiting the structure of the physical environment. For VLC eavesdropping, we focus on materials used in buildings, windows, keyholes, and doors. Our mmWave eavesdropping setup considers reflections on small-scale objects and reflections on the intended receiver device itself. To avoid being detected and blocking the main signal beam, the attacker must exploit secondary and non-ideal propagation paths and contend with degraded Signal-to-Noise Ratio (SNR).

Our results show that nearby attackers, the "spy next door", can often intercept VLC and mmWave signals, potentially revealing information on personal habits in smart home applications or even sensitive health data. Indoors, VLC reflections on walls are sufficient for eavesdropping, thus also affecting Internet of Things (IoT) and smart home applications. mmWaves are reflected perfectly on smartphone and laptop displays, making the intended receiver herself an additional risk for eavesdropping.

9.2 VLC TESTBED AND EVALUATION SETUP

In the following, we present a VLC attack model in Section 9.2.1. To succeed, an eavesdropper must decode an advanced modulation scheme from a non-optimal position

(Section 9.2.2). Due to this modulation scheme, in contrast to modulation schemes like on-off keying, an attacker is not successful merely by detecting if the light is on or off. Instead, a higher-order modulation based on Orthogonal Frequency-Division Multiplexing (OFDM) must be decoded. Our practical hardware testbed and software setup are described in Section 9.2.3.

9.2.1 *Attack Scenarios*

We consider that Alice transmits to Bob using VLC. Alice attempts to prevent eavesdropping by relying on visible light's directionality and blockage characteristics. However, Eve aims to undermine this privacy and eavesdrops on Alice's transmission. We consider NLOS positions where Alice and Bob might not expect eavesdroppers because they are considered to have poor or no reception outside of the room. Eve takes advantage of the gap underneath a door, keyholes, and windows. Inside the room, she can rely on environmental reflections to eavesdrop, even if she is not located within the light beam. In all scenarios, we assume that Bob is near the reflector or opening but not blocking the signal to Eve. We use Eve's Bit Error Rate (BER) to quantify the success of attacks. In our experiments, no error coding is used. Hence, we assume that Eve can still decode signals with a BER lower than 10 %. Yet, as long as the BER stays below 50 %, Eve receives parts of the conversation, which can be sufficient to reconstruct the message.

Note that the modulation scheme is important for successful eavesdropping—since Eve has a potentially worse path to Alice than Bob, and she might not be able to decode information that is still received by Bob. In general, Eve could improve her eavesdropping capabilities by using more advanced hardware that has better sensitivity. While Eve can infer whether a light is on or off through a door gap from a large distance, we evaluate if this leakage is sufficient for decoding higher modulation order signals.

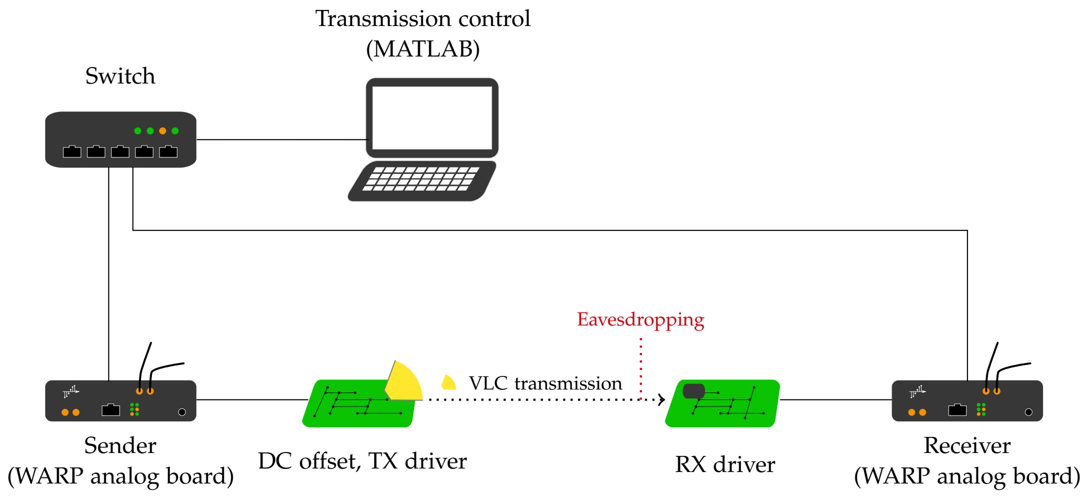


Figure 23: VLC setup based on Wireless Open-Access Research Platforms (WARPs).

9.2.2 *Modulation*

Light sources installed in buildings and public spaces are designed to illuminate the greater environment and thus have a large beamwidth. In comparison to legacy radio Wi-Fi systems, VLC signals feature a more distinct area of reception and do not encode phase information [LSK11]. Furthermore, VLC is different from optical communication with lasers, in which coherent signals propagate on a narrow path from sender to receiver, thereby restricting eavesdropping to a few locations [LMGGB15].

We use a white Light-Emitting Diode (LED) as a sender and therefore cannot use Color-Shift Keying (CSK) as defined in IEEE 802.15.7 [The11]. Instead, we modulate our data with DCO-OFDM as in the setup in [QHK14]. DCO-OFDM is similar to OFDM in the IEEE 802.11a/g Wi-Fi baseband [The12], and efficiently uses the limited bandwidth of our LED, which has a linear operation range of only 2 MHz. Since we modulate the light intensity of incoherent light, there are two restrictions: first, the light intensity can only be positive in contrast to an electromagnetic field; and second, incoherent light has no consistent carrier phase that could be used for modulation. Therefore, the symbol representation must be real-valued and positive. DCO-OFDM meets these requirements by using only half of the total subcarriers to obtain a real-valued OFDM signal and then adds a constant Direct Current Offset (DCO) to produce a positive intensity.

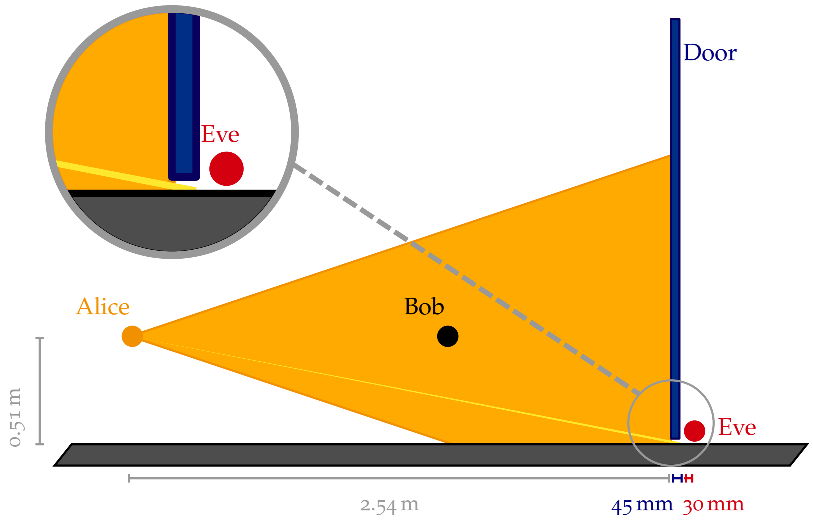
9.2.3 *Hardware Setup*

An overview of our hardware setup for VLC is shown in Figure 23. It is based on Software-Defined Radios (SDRs), namely the Wireless Open-Access Research Platform (WARP) v1.2 [WAR19]. We use an array of white *Bridgelux BXRA-40E0950-B-03* LEDs as a sender and an *LEC-RP0508* Photo Diode (PD) as a receiver. WARP is made for rapid prototyping wireless communication systems in combination with MATLAB. By using the WARPLab Field-Programmable Gate Array (FPGA) design, a computer encodes the signals and triggers the transmission on multiple connected WARP boards. This WARP-based setup is highly flexible and can be extended to drive multiple senders and receivers simultaneously.

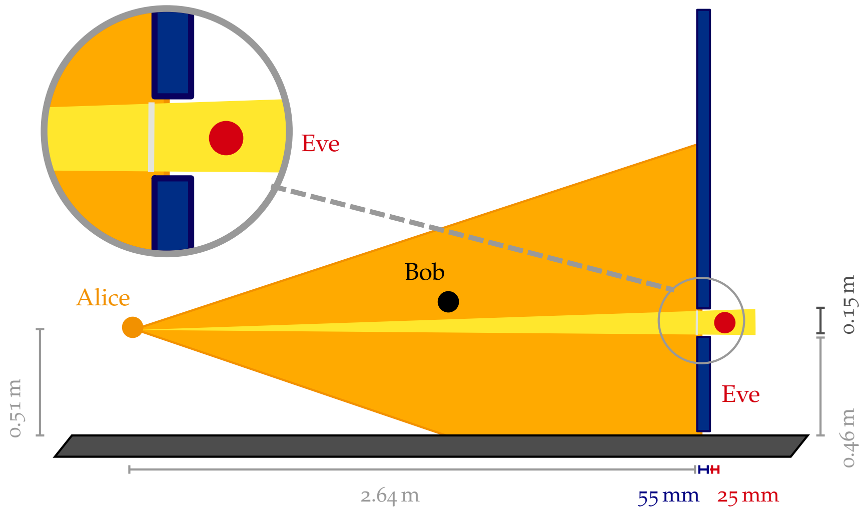
We extend the WARPs, which typically output Wi-Fi carrier signals, with analog boards interfacing arbitrary baseband signals as required for DCO-OFDM. This setup provides a sampling rate of 40 MHz that is sufficient for transmission with LEDs that typically have a modulation bandwidth of 2–3 MHz. We modulate 128 subcarriers, but only 64 carry the data to obtain a real-valued OFDM output from the analog boards. A DCO is added by another circuit, and its voltage can be adjusted to enhance the optical power. The driver circuit converts the voltage to a current, which alternates the LED brightness and thus modulating the signal onto the LED. At the receiver, the photodiode converts light to current, which is amplified by the VLC receiver, and converted to a voltage by the driver circuit.

9.3 PRACTICAL VLC EAVESDROPPING

In the following, we conduct various experiments to evaluate VLC eavesdropping performance in real-world scenarios. These scenarios are eavesdropping through a door



(a) Eavesdropping setup through a door gap.



(b) Eavesdropping setup for window and keyhole.

Figure 24: Eavesdropping setups.

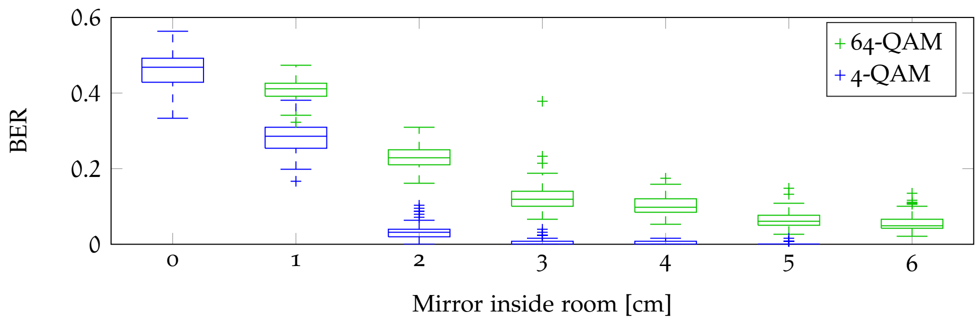


Figure 25: Reflection zone measurement by moving a mirror inside.

gap with multiple flooring materials (Section 9.3.1), a keyhole (Figure 9.3.2), and a window with various covers (Section 9.3.3), as well as eavesdropping a reflection from a wall (Section 9.3.4).

Each experiment is conducted with 100 repetitions to determine the median and confidence bounds. Boxplots represent the 25th and 75th percentiles using a box while the whiskers extend to the non-outliers. We vary sender and receiver positions and place obstacles in between, as described in detail in the following.

9.3.1 *Door Gap Eavesdropping*

We examine if eavesdropping based on floor reflections through a door gap from a $4.83\,\mathrm{m} \times 2.73\,\mathrm{m}$ room is possible. Alice is transmitting from a chair facing the door with a $37°$ cone, which is slightly focusing her light. Eve is located outside the room at the door's center on the ground. She is not facing Alice directly but exploiting reflections, as depicted in Figure 24a. We find that Eve's location is optimal at a $30\,\mathrm{mm}$ distance from the gap.

First, we record a regular receiver baseline with an open door. Then we measure the optimal scenario for Eve using mirrors as well as typical flooring materials listed in Table 5. Our floor samples have a limited size. Thus, we first inspect the influence of reflector sizes in Section 9.3.1.1. The original flooring is a gray carpet leaving a $15\,\mathrm{mm}$ gap between the door and the floor. When testing with different materials, the gap size narrows based on the thickness of the material under test. Hence, we investigate the impact in Section 9.3.1.2. Afterward, we focus on materials with clear grooves and surface structures in Section 9.3.1.3. Finally, we compare different flooring materials and their effectiveness for eavesdropping in Section 9.3.1.4.

9.3.1.1 *Reflection Zone*

We measure the impact of reflection zone dimensions in the following setup: Eve is located on top of a mirror in $3\,\mathrm{cm}$ distance to the door. The mirror is moved inside centimeter-wise towards the transmitter, while Eve's distance to the door remains $3\,\mathrm{cm}$ in all experiments. Since the original carpet is a poor reflector (which will be shown in Section 9.3.1.4), the small-sized floor samples give us an upper bound on the BERs. The measurement at $0\,\mathrm{cm}$ is taken at alignment with the inner surface of the door, and $1\,\mathrm{cm}$ means that the mirror is already inside the room. As shown in Figure 25, Eve's 4-QAM BER significantly drops when the mirror is $2\,\mathrm{cm}$ inside the room while she requires around $5\,\mathrm{cm}$ to achieve an excellent eavesdropping performance for 64-QAM. This suggests that Eve only needs little of the reflector to be inside the room to eavesdrop, even for higher-order modulation schemes. She could place a tiny reflector underneath the door if the existing flooring material is not sufficient for eavesdropping.

To validate these findings with other materials, we use a rectangular piece of parquet with a shiny surface as a reflector and measured Eve's BER for different orientations of the parquet. As Figure 27 shows, the parquet # 11 has the lowest BER when aligning the longer side along the path towards the transmitter. This corresponds to a larger reflection zone size on the NLOS path and means that our measurements in the remainder of this paper indicate an upper bound BER for a room completely filled with the floor material under test.

9.3.1.2 *Gap Size*

Many of the tested samples are of different thicknesses and thus alter the door gap size. To analyze if this has a significant bias on our measurement results, we raise a test layer of acrylic glass to different heights to reduce the available eavesdropping gap size and then measure the BER at each of these heights. As seen in Figure 26, a narrow gap size especially raises the BER for higher modulation orders. Hence, our measurement results are upper bound BERs for each material, which can be decreased with larger gap sizes. Although a particular material may cause better reflections in general, the performance gains may be dwarfed by the additional losses from reducing the door gap size. For eavesdropping, this means that there is a trade-off between using the existing floor material and maliciously placing another material under the door.

9.3.1.3 *Material Surface*

Intuitively, surface structures on materials block light on the way to Eve. To exclude the source of different BERs from being material differences or the size, we rotate the same squared vinyl plank # 5 with wooden structure crosswise and lengthwise towards Eve—since it is squared, we are only changing its orientation but not the reflection zone. Figure 27 shows the difference between the orientations: even though it is the same square sized material, the mean BER drops from 36.23 % to 31.01 % for the lengthwise orientation of the wooden structure, due to having fewer ripples from the vinyl on the way to Eve. This difference is even more significant for lower order modulation schemes

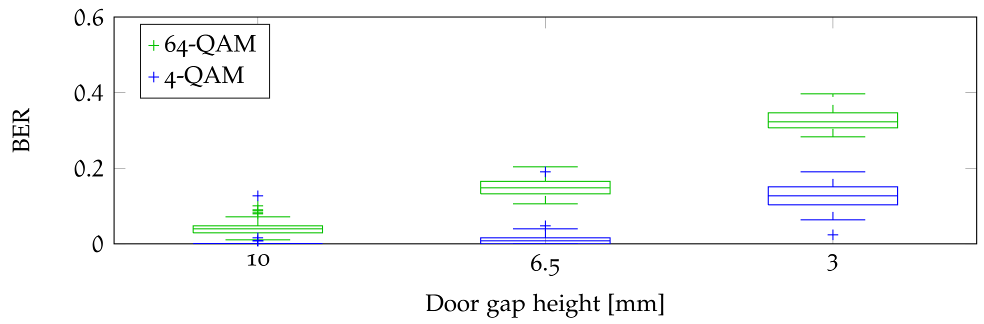


Figure 26: Acrylic glass at different heights.

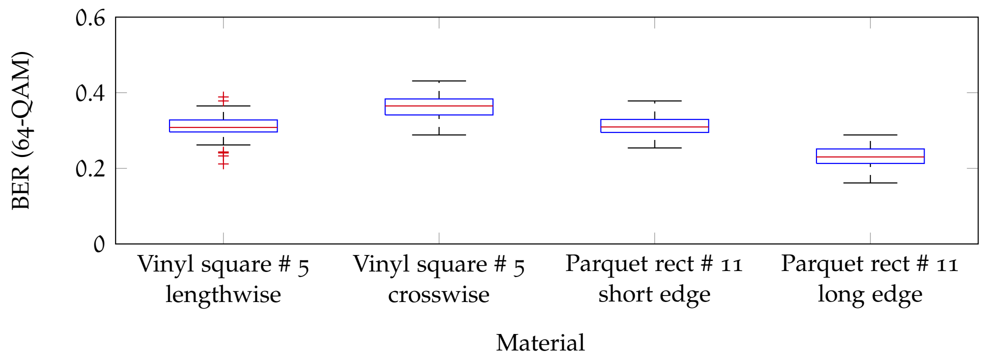


Figure 27: Different tile orientation eavesdropping performance.

(e.g., 9.81 % vs. 17.35 % for 4-QAM). This means that Eve can noticeably improve her performance by optimizing her position with regards to the flooring material's structure, even if she cannot change the flooring material itself.

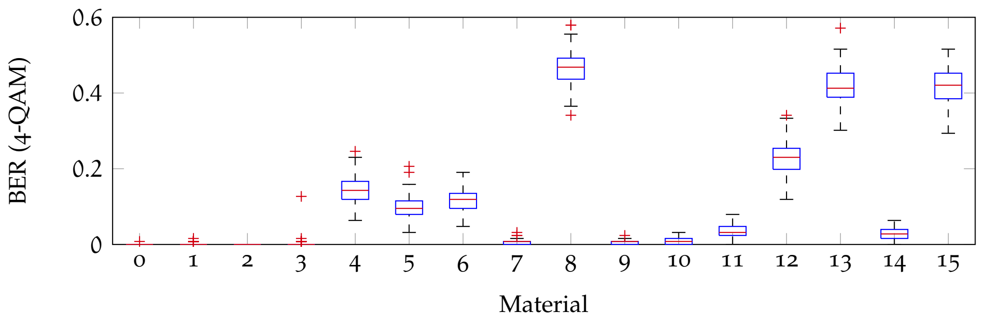
9.3.1.4 *Flooring Comparison*

In this experiment, we compare all materials in their optimal orientation and positioning with a 30 mm distance in front of the door's center front. Since the original flooring has the second-worst BER, all better BERs exclusively originate from the support of the flooring materials. Note that these results are an upper bound on the BERs that would be reduced by a completely filled floor of the material with a constant 15 mm gap size. In the case of surface structure, we turned tiles in the optimal orientation.

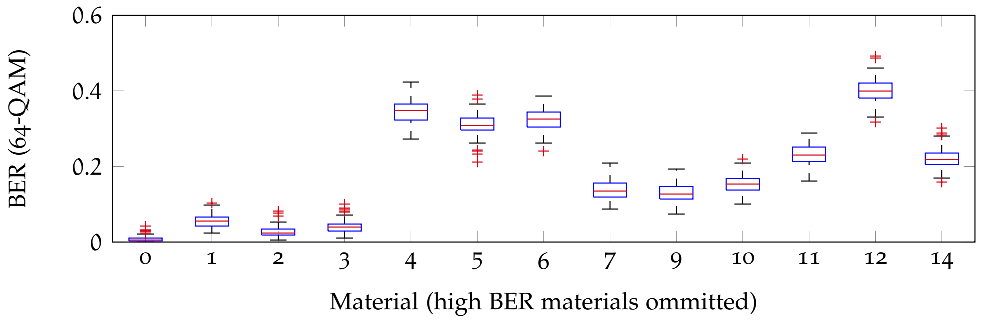
The results are shown in Figure 28. We find that the non-typical flooring materials—mirrors, metal, and acrylic glass—are the best reflectors. Even at 64-QAM, their BERs stay below 5.6 %. Note that the mirror in this experiment is slightly better than in prior experiments, because it is as far in Alice's room as possible, allowing it to reflect more of her signal beam towards Eve. The shiny glazed tiles perform better than the structured tile, but all tiles work well at 4-QAM. Even the structured tile has a 2.8 % mean BER. For laminate and parquet, the gap size was reduced by at least 8 mm respectively 10 mm. Yet, their mean BERs are 0.6 % and 3.6 % for 4-QAM. We assume that the other laminates and parquets are also good reflectors for larger gap sizes. A defense against VLC eavesdropping is carpet; the original carpet has 41.5 % BER, and the 10 mm carpet under test has 46.6 % BER. This result is counterintuitive—even though one can see if

Table 5: Floor material sizing and description.

#	Image	Material	Height	Length	Width
0		Baseline with open door	—	—	—
1		Metal sheet	0.5 mm	750 mm	406 mm
2		Mirror	4.5 mm	302 mm	302 mm
3		Acrylic glass	5 mm	209 mm	406 mm
4		Vinyl plank "Antique Elm"	1.5 mm	100 mm	100 mm
5		Vinyl plank "Rosewood Ebony"	1.5 mm	100 mm	100 mm
6		Vinyl plank "Corfu"	1.5 mm	100 mm	100 mm
7		Laminate "Whitewashed Oak"	8 mm	126 mm	88 mm
8		Carpet "Antique Bone"	10 mm	103 mm	103 mm
9		Black shiny glazed tile	7 mm	106 mm	16 mm
10		White shiny glazed tile	7 mm	106 mm	106 mm
11		Parquet "Character Maple"	10 mm	128 mm	89 mm
12		Parquet "Mahogany Natural"	11 mm	127 mm	87 mm
13		Laminate "Middlebury Maple"	12 mm	134 mm	91 mm
14		Gray structured dull glazed tile	8 mm	147 mm	147 mm
15		Original carpet	0 mm	—	—



(a) 4-QAM eavesdropping.



(b) 64-QAM eavesdropping.

Figure 28: Door gap eavesdropping on different flooring materials.

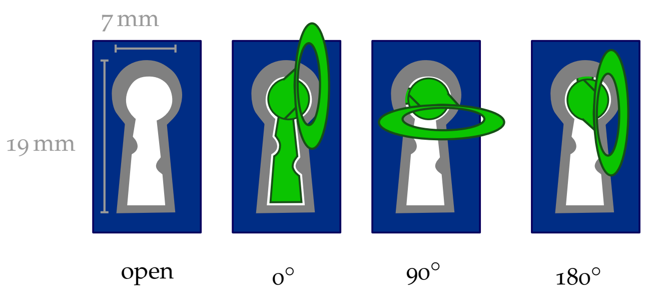


Figure 29: Eavesdropping keyhole setup.

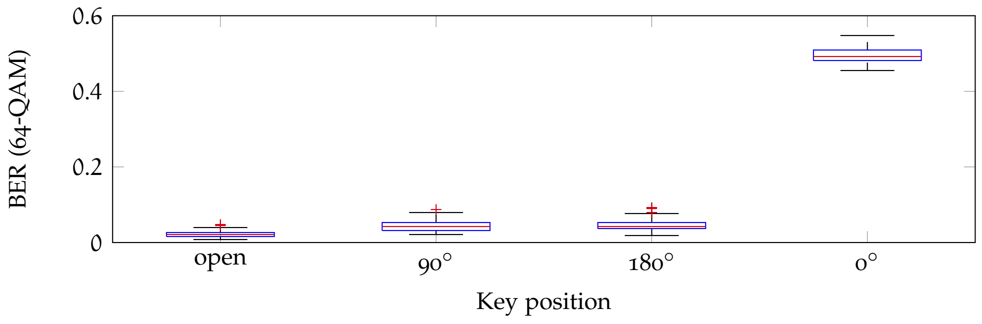


Figure 30: Eavesdropping through a keyhole.

the light is on or off in the room from the position of Eve's photodiode, carpet reflections are insufficient for decoding higher-order modulation schemes.

9.3.2 *Keyhole Eavesdropping*

Keyholes are a similar target as door gaps and attackers can also use these to eavesdrop on transmissions from outside the room, but from a different angle. In the following, we test if Eve can eavesdrop an ongoing transmission with Bob being located by a keyhole. We setup Alice and Eve as shown in Figure 24b, with the actual keyhole aligned 51 mm from the bottom of the opening surrounded by 55 mm thick black foam and align Eve so that her photodiode is directly behind the keyhole opening. We use a brass Victorian lockset with a keyhole depth of 26 mm. We assume that Bob is not blocking any of the signals from Eve.

We measure Eve's BER when the keyhole is open when the key is blocking the keyhole, when the key is turned 90°, and when the key is turned 180° as illustrated in Figure 29. The results in Figure 30 show that partially blocking a keyhole has almost no effect on Eve. However, completely blocking the keyhole prevents eavesdropping. In the 90° and 180° positions, the key handle blocks slightly different amounts of light in the path. These results show that small holes are sufficient for eavesdropping on Line-of-Sight (LOS) paths.

9.3.3 *Window Eavesdropping*

Intuitively, windows allow eavesdropping on VLC. We investigate if common window add-ons—including a privacy foil—reduce information leakage. The experimental setup is the same as in Figure 9.3.2, but we mount a 2 mm thick glass window pane in front of the black foam 15 cm × 15 cm opening.

As a baseline, we first measure the 4-QAM and 64-QAM BER at Eve, without and with glass. Afterward, we attach a window film and insect screen to the glass window. The window film is an *Artscape* "etched glass" film that is advertised to provide UV protection and create privacy. One could think that this material blocks a significant amount of light, since objects behind the window with this film are hard to recognize. The insect screen is a black screen intended for 4.1 mm or 4.6 mm spline sizes. Each rectangle in the screen's mesh is approximately 1.2 mm × 2 mm.

Neither the mesh nor the privacy screen offer any protection against eavesdropping when using 4-QAM; the mean BERs for all combinations stay below 1.6 %. As shown in Figure 31, even using 64-QAM, these window add-ons only reduce Eve's signal quality minimally, and Eve can still decode with a small BER of a maximum 3.2 % mean for combining these two modalities.

9.3.4 *Wall Eavesdropping*

In smart homes and IoT application scenarios, many devices are equipped with communication interfaces and cameras. Depending on the modulation scheme, even cameras can decode VLC. In the following experiment, we analyze if devices that are nearby but in NLOS can eavesdrop. For this, we mount Alice on a rotor, resulting in 0.97 m height

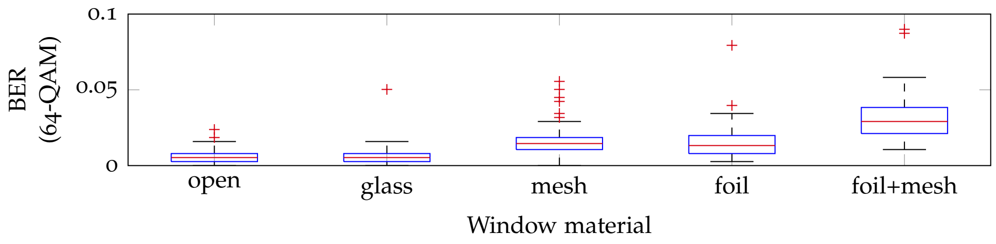


Figure 31: Eavesdropping impact of partially covering a window.

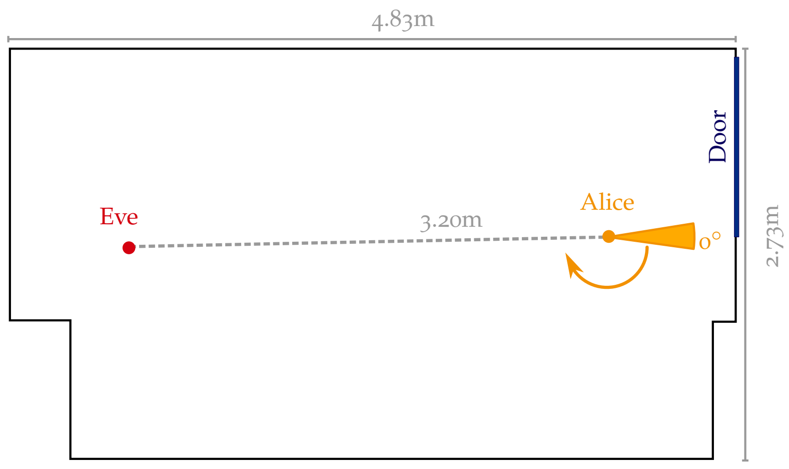


Figure 32: Setup for measuring reflections inside a room.

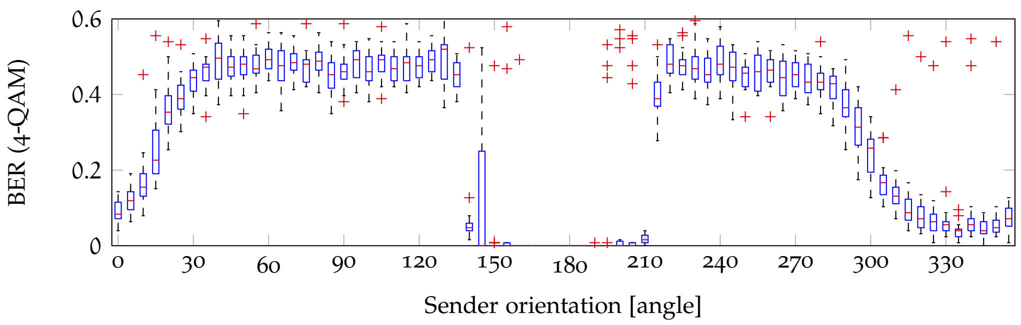


Figure 33: Eavesdropping on wall reflections.

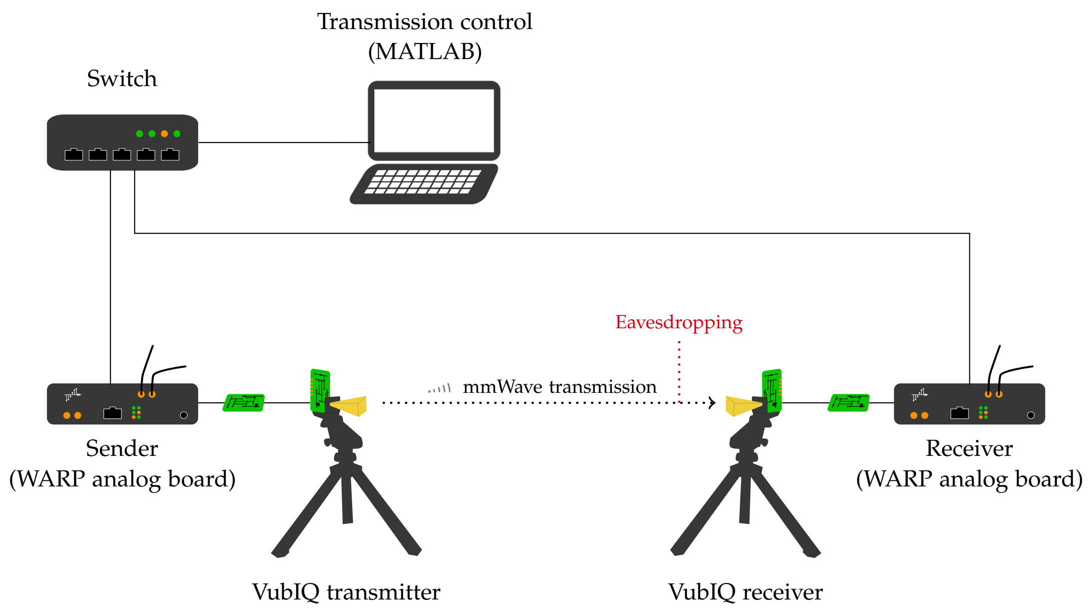


Figure 34: mmWave setup based on WARPs and VubIQs.

in total, and take 20 measurements in 5° steps. We replace her reflector with a narrower 17° cone instead of the 37° cone in the previous experiments to get higher directionality, which is required to refine the reflections' origins further. Eve is positioned at the other side of the room, as shown in Figure 33, and Bob is assumed to be in the direction that Alice is pointed in. The results show that the white wall and the blue painted wooden door generate sufficient reflections for eavesdropping in 4-QAM when Alice is sending in the opposite direction.

9.4 PRACTICAL MMWAVE EAVESDROPPING

When eavesdropping mmWave communication, we find similar results [Ste+15]. Everyday objects reflect signals of higher modulation orders sufficiently for eavesdropping.

The setup for mmWave eavesdropping shown in Figure 34 is very similar to the VLC setup. For the sender and the receiver, WARP analog boards are used [WAR19]. Their analog baseband signal is mixed to 60 GHz with the VubIQ development system [Pas13]. Due to incompatible signals between the WARP and VubIQ system, we employ an additional circuit in between to make the required baseband adjustments. We mount the antenna with 7° beamwidth (the narrowest available) throughout all experiments.

Indeed, mmWaves propagate in a similar manner to visible light. An *iPhone* display produces good reflections, and a larger laptop display results in perfect reflections. Another very good reflector is a metal block. Acrylic glass is an interesting material, as it reflects mmWaves quite well while almost not blocking them. Moreover, round surfaces like a cup scatter the signal in all directions. Thus, round objects reduce the signal strength but allow the eavesdropper to be variable in her location.

Due to the materials, most electronic devices are made of and the environment they are used in, mmWaves can be eavesdropped in practice despite their directionality.

9.5 CONCLUSION

In scenarios where devices are nearby, neither VLC nor mmWaves should be considered secure against eavesdropping by their physical nature. Blocking VLC information leakage through windows while keeping them functional seems to be impossible; privacy films offer almost no protection. mmWaves can be reflected by the intended receiver, such as a smartphone. Even though blockage is a challenge for high throughput in VLC and mmWave applications, an eavesdropper who carefully selects a position or places reflecting materials can be successful. Eavesdroppers might be able to compensate for their disadvantaged position by employing better photodiodes or antennas. In IoT deployments and smart homes, where trusted and untrusted devices are nearby, directionality and blockage of signals should never be the only security measure.

MY CONTRIBUTION

My main project during my stay in Houston at Rice University in Edward Knightly's group was VLC eavesdropping [Cla+15b]. Daniel Steinmetzer joined me in writing the paper, and Joe Chen assisted me in performing experiments. Daniel did similar experiments to mine for mmWave communication, with him leading the project [Ste+15].

Section 9.1 is partially taken from [Cla+15b, Sec. 1]. Section 9.2 is based on [Cla+15b, Sec. 2]. Section 9.3 is based on [Cla+15b, Sec. 3]. Section 9.4 is a summary of [Ste+15].

10 PHYSICAL LAYER SECURITY FOR VISIBLE LIGHT COMMUNICATION

In this chapter, we survey existing Visible Light Communication (VLC) Physical-Layer Security (PLS) solutions and specify a new attack model, which uncovers many attacks that have not been considered. Section 10.1 motivates PLS as security solution and why to re-evaluate it for VLC. Section 10.2 briefly explains what wireless PLS is and links it to Internet of Things (IoT) environments. In Section 10.3, we outline the prospects of PLS in VLC and explain how VLC differs from Radio Frequency (RF) transmissions. Section 10.4 categorizes and surveys existing work on VLC PLS. We define a new attack model in Section 10.5 for threats in VLC PLS and use it to evaluate the security of existing solutions in Section 10.6. This chapter is concluded in Section 10.7.

10.1 MOTIVATION

Wireless transmissions come with numerous physical properties that can be useful to reach security goals. This field of research is also called PLS.

The previous chapter focused on security provided by the limited range of VLC and mmWave transmissions, which is a special case of PLS. Transmission range is the most intuitive physical property that can be integrated into solutions to reach security goals like confidentiality, authentication, and integrity. For example, if a transmission's range is appropriately limited, this ensures confidentiality and integrity. A location or the proximity can be used as authentication factor. While the previous chapter showed that VLC can be eavesdropped by Non-Line-of-Sight (NLOS) attackers in certain scenarios, the VLC Physical Layer (PHY) might still provide additional information to take into account.

PLS is well-known in the RF domain, and many solutions based on 2.4 GHz Wi-Fi exist. While RF and VLC both enable wireless communication, some of their physical properties differ. This includes but is not limited to the different propagation characteristics. Adapting RF solutions to VLC is technically possible, and the related work we review mainly follows this approach. However, this might lead to security degradation compared to the original solution. Security guarantees depend on the PHY and need to be reconsidered once the PHY changes.

10.2 WIRELESS PHYSICAL LAYER SECURITY

The following paragraphs introduce the most interesting features and limitations of wireless PLS.

RE-PURPOSING EXISTING INFORMATION Wireless communication between devices includes various characteristics, which are mostly not forwarded to upper layers. Physical properties of a signal come with a minimal overhead as they are already included in a transmission. Thus, many solutions based on these properties are comparably light-weight. However, most PHY properties are discarded on reception by commodity hardware. Available information on the wireless PHY includes a highly environment and location-dependent communication channel as well as hardware-specific variations.

INFORMATION-THEORETIC SECURITY PLS can provide information-theoretic security. While typical cryptographic solutions rely on computationally hard problems, information-theoretic secure systems are built on assumptions that cannot be broken by attackers with unlimited computation power. Since computation power increases over time, users should think of encrypted data sent over the Internet as "confidential for the next 20 years". The Vernam One-Time Pad (OTP) prevents computational attacks by using an encryption key that has the same length as the plaintext. Since the encryption key is a bitstream that is never reused and applied with eXclusive OR (XOR), it is possible to calculate the encryption key from known plain- and cyphertext. However, knowledge of the plaintext is useless to the attacker for decrypting further messages.

Even though the Vernam OTP is well-known and information-theoretic secure, practical systems still rely on encryption keys way shorter than the plaintext, which makes them vulnerable to computational attacks. Considering the number of messages exchanged between current systems and the hardware limitations for consumer and IoT devices, as well as initial key distribution, the Vernam OTP cannot be applied.

A model similar to the Vernam OTP but for wireless communication is Wyner's wiretap channel [Wyn75]. It relies on the assumption that wireless channels between all communicating devices are always slightly different, and this difference can be encoded to transmit confidential information. It is more light-weight and suitable for IoT devices.

LIMITATION TO DIRECT LINKS PLS can only be used between devices with a physical link since wireless properties cannot be measured via multiple hops. This is sufficient in many IoT environments with directly interconnected components. Some link properties can be passed to and processed by upper layers—for example, the individual wireless channel can be used to extract symmetric keys due to its reciprocity.

10.3 APPLYING PHYSICAL LAYER SECURITY TO VISIBLE LIGHT COMMUNICATION

In Section 10.3.1, we re-introduce what VLC is from a more signal and implementation focused point of view. In Section 10.3.2 we detail the most important properties on the PHY that change in comparison to RF communication.

10.3.1 *What is Visible Light Communication?*

For a better understanding of how PLS can be applied to VLC, we need to define VLC and its applications.

VISIBLE LIGHT TRANSMISSIONS In VLC systems, data is modulated on visible light that is emitted by common illumination sources. Being dimmed below eye sensitivity [RRL12], VLC becomes typically invisible for the human eye. Hardware requirements on transceivers are low: common Light-Emitting Diodes (LEDs) facilitate data transmission, and receivers use Photo Diodes (PDs) or simple camera sensors. With optical propagation characteristics and a large spectrum of available bandwidth, VLC promises to overcome significant bottlenecks in future wireless applications and the IoT, as for example aircraft cabin communication, location-based services, and health monitoring [TC14].

VISIBLE LIGHT IMPLEMENTATIONS VLC concepts fundamentally differ from those of RF. Ideally, VLC uses *off-the-shelf* light sources that simultaneously illuminate the environment [Sch+14]. Even a standard smartphone display and camera sensor can be leveraged for data transmission, not perceivable by human eyes [Li+15]. In the following, we outline the aspects that need to be considered when adopting wireless communication concepts from legacy Wi-Fi to VLC.

10.3.2 *Visible Light Properties Compared to Radio Frequency Properties*

Communication paradigms differ when comparing RF communication to VLC, which also influences security considerations. Many existing RF PLS concepts can be adapted to VLC, such as securing the communication by leveraging radio wave propagation characteristics to achieve information-theoretic security [Wyn75]. However, optical propagation is fundamentally different from radio waves: light is highly susceptible to reflections and blockage and less sensitive to scattering or diffraction. Properties of light are highly interesting to PLS, i.e., the blockage can be a security advantage, polarization is easier to change, and users are well aware of light propagation. Though, also the attacker capabilities change, leading to pitfalls in securing VLC.

In the following, we will discuss the most different physical aspects when compared to RF communication, which are coherence, intensity, reciprocity, polarization, and propagation.

NO COHERENCE Coherence is an essential characteristic of light that limits the achievable throughput. The coherence time τ_c of a light wave determines the duration of correlated phase fluctuations. As light propagates with speed-of-light c, the coherence length $l_c = c\tau_c$ defines the distance in which meaningful phase information can be retrieved and should be longer than the optical path. However, the typical coherence length of LEDs is $20\,\mu m$, which is too short in practice [STS91]. Hence, phase information cannot be used for data modulation in VLC.

INTENSITY-BASED MODULATION SCHEMES Without the possibility of modulating the phase, amplitude only Intensity Modulation (IM) is required for data transmission in VLC. Averaging the light intensity as a square magnitude over a time longer than one optical period [STS91] achieves only positive values, which can be leveraged for encoding. On-Off Keying (OOK) encodes bits by switching a light source on and off. By alternating the light amplitude in the baseband over time, sine waves with a Direct Current Offset (DCO) can be represented. In comparison to legacy Wi-Fi, this achieves a lower spectral efficiency but allows for modified versions of Orthogonal Frequency-Division Multiplexing (OFDM). The missing phase information also implies that Channel State

Information (CSI) between transmitter and receiver, typically denoted as channel matrix H, only contains real intensities without complex phase shifts.

MISSING RECIPROCITY RF systems use the same antenna for reception and transmission. Achieving a similar behavior in both communication directions, this results in channel reciprocity, meaning that one channel becomes the inverse of the other channel. In VLC, the channels are spatially separated: the LEDs for transmission and the PDs for reception are located in different positions. These two unidirectional channels are independent of each other and hence suppress channel reciprocity.

POLARIZATION Polarization filters are easy to implement for light and already integrated into monitor screens and camera optics. Using filters at transceivers allows separating specific light components. With unmatched polarization, however, filters cause high attenuation of light intensity.

PROPAGATION Visible light has different propagation properties compared to RF in the 2.4 GHz range. Light does not penetrate most obstacles; it typically propagates throughout a single room over multiple reflections. Eavesdroppers can exploit these reflections or gaps between obstacles, as shown in the previous chapter; blockage alone should not be considered as a security feature.

10.4 SECURITY MECHANISMS AND ASPECTS

In this section, we reference related work on VLC security for the goals confidentiality (Section 10.4.1), localization (Section 10.4.2), and integrity (Section 10.4.3). We categorize visible light security approaches and relate them to existing legacy Wi-Fi PLS solutions.

10.4.1 *Confidentiality*

The majority of existing work on PLS related to VLC focuses on achieving confidentiality. Confidential information is missed by unintended receivers on the PHY, for example, due to the eavesdropper's location or artificial noise.

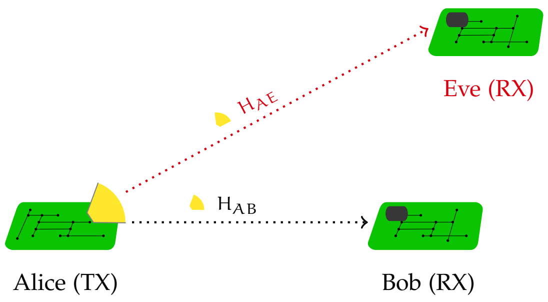


Figure 35: Wyner's wiretap channel (simplified).

WYNER'S WIRETAP CHANNEL In Wyner's wiretap channel, Alice and Bob communicate with each other while Eve attempts to eavesdrop, as shown in Figure 35. The channels H_{AB} and H_{AE} have different CSIs, such that Eve misses information and Bob achieves a secrecy capacity. Based on Wyner's wiretap channel, various work for establishing confidentiality on VLC links has been published.

The two confidentiality schemes by Zhang et al. assume that Eve misses information displayed on screens [Zha+14]. In their first approach, two smartphone screens facing each other at $0.1\,m$ distance and a view angle of $2°$ cannot be eavesdropped if a single location attacker is more than $1.4\,m$ away. Their second approach adds further security as the user has to actively rotate the screen within a certain angle, which further limits possible eavesdropping positions.

Another wiretap channel implementation is Orthogonal Blinding (OB), where Zero-Forcing Beamforming (ZFBF) is used to transmit data towards Bob and noise in all other directions, such that eavesdroppers at unknown positions have a bad Signal-to-Noise Ratio (SNR) [ALK12]. For ZFBF a zero-forcing filter, which is the pseudo-inverse channel, is applied by Alice before sending data to Bob. H_{AB} and ${H_{AB}}^{-1}$ cancel out each other, and Bob receives data without channel distortions. This effect does not cancel out at Eve's position, and her data is affected by normal channel distortions:

$$\begin{bmatrix} RX_{Bob} \\ RX_{Eve} \end{bmatrix} = \begin{bmatrix} H_{AB} \\ H_{AE} \end{bmatrix} \begin{bmatrix} H_{AB} \end{bmatrix}^{-1} \begin{bmatrix} \text{Data} \end{bmatrix} \tag{1}$$

For OB, this concept is extended with a transmitter having two antennas, one transmitting the same signal as above, and another transmitting noise to an orthogonal channel. This noise does not harm Bob, but Eve receives a higher amount of noise:

$$\begin{bmatrix} RX_{Bob} \\ RX_{Eve} \end{bmatrix} = \begin{bmatrix} H_{AB} \\ H_{AE} \end{bmatrix} \begin{bmatrix} H_{AB} \\ \perp H_{AB} \end{bmatrix}^{-1} \begin{bmatrix} \text{Data} \\ \text{Noise} \end{bmatrix} \tag{2}$$

The dimension of the channels increases with the number of antennas, but the overall system is similar. Mostafa and Lampe first introduced an OB variant for VLC by using zero-forcing and artificial noise [ML14]. They extend their work to a massive array with thousands of LEDs [ML15]. Zaid et al. improve the model to achieve a higher secrecy rate [Zai+15]. Le Minh et al. combine multiple CSI matrices to one filter, which makes it harder to guess [LM+14].

FRIENDLY JAMMING A friendly jammer sends a pseudo-random signal only known by legitimate receivers, as illustrated in Figure 36. Eavesdroppers cannot subtract the jamming signal from the data, resulting in a poor SNR. In contrast to ZFBF, the channel H_{AB} is not required to be known or measured by trusted parties. Confidential communication only requires sufficient coverage by the friendly jammer. As a side effect, parties without knowledge of the pseudo-random jamming signal also cannot communicate.

Chow et al. create a trusted communication zone with four data and four jamming LEDs, which emit random binary signals [Cho+15]. As a proof of concept, the reception outside a trusted communication zone is compared, which shows that jammers can restrict the zone.

KEY-BASED ENCRYPTION In RF PLS, it is possible to derive symmetric encryption keys based on channel reciprocity [Jan+09]. There is no reciprocal channel in VLC, as PDs are receive only and LEDs are transmit only. However, it is possible to broadcast pre-existing encryption keys with VLC.

Okuda et al. build a system that regularly changes standard Wi-Fi keys and broadcasts key updates using VLC [Oku+11]. Ho, Duan, and Chen propose a central instance continuously streaming key material through VLC, which is used to secure Wi-Fi transmissions [HDC15]. Both approaches lower the effective Wi-Fi communication range to that of VLC, as shown in Figure 37. While eavesdropping VLC is possible as previously stated in Chapter 9, an eavesdropper is now required to be at a location where VLC and Wi-Fi can be received. In most cases, VLC range is shorter than Wi-Fi range.

10.4.2 *Localization and Authentication*

Location information is a useful second authentication factor. In scenarios that require physical presence, a location or distance proof can be included. For example, to unlock a door with a key card, the key card should be next to the door and contain the correct key material. Localization and authentication in VLC systems can be classified into systems with and without data transmission, and systems that recognize or ignore multipath effects.

LOCALIZATION BY KNOWN PATTERNS Gu et al. detect and compensate multipaths in an OOK-based system [Gu+16]. Aminikashani, Gu, and Kavehrad perform positioning with high data throughput [AGK15]. They compare the positioning performance of OFDM to OOK. Since OFDM is using more bandwidth and has a higher sampling rate, more multipath effects can be detected, resulting in higher precision. Both papers simulate empty rooms; multipaths would be less predictable in a real environment.

Kuo et al. propose a practical system locating smartphones with multiple LEDs at known locations, transmitting one out of 16 tones [Kuo+14]. Their method is based on optical Angle of Arrival (AoA) information and multiple optimizations, allowing for additional data transmissions.

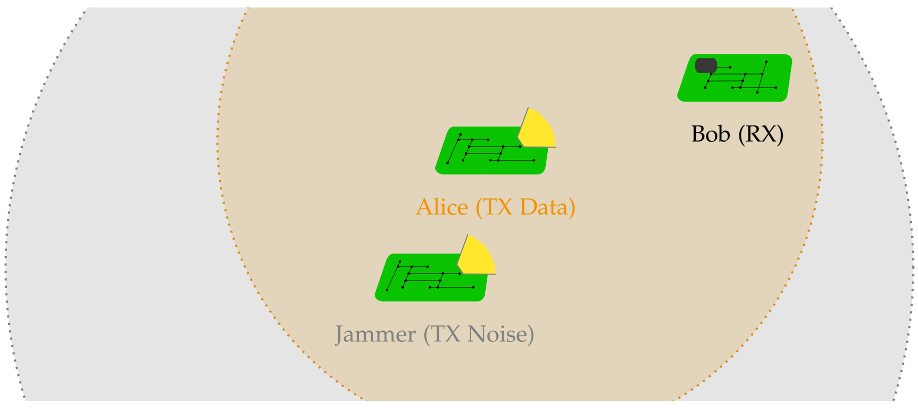


Figure 36: Friendly jamming.

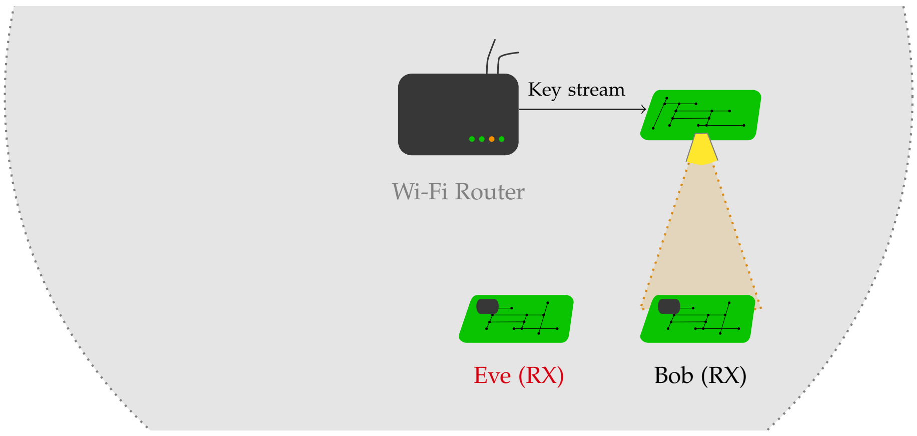


Figure 37: Key streaming to reduce Wi-Fi range to VLC.

10.4.3 *Integrity*

In the following, solutions that enable integrity on the PHY are referenced. Note that some of the previously mentioned confidentiality solutions also enable integrity. For example, if a key is streamed via VLC to protect Wi-Fi confidentiality as in [HDC15], injecting Wi-Fi signals without knowledge of that key is also impossible. Thus, only one remaining approach is listed in the following.

INTEGRITY BY POLARIZATION Javidi and Nomura implement an optical verification system that adds a mask of 128×128 linear polarization filters to a transmitted image [JN00]. Normal cameras do not interpret polarization. Hence, unaware receivers cannot recognize a pattern. The modified camera has a reference polarization mask, enabling a correlation to the encoding polarization mask, to check if the transmitted image has the same polarization.

In addition to this specific optical verification scenario, polarization filters add another degree of freedom to the system that could be used for message integrity verification or encryption in general. This is the opposite effect of the missing phase information in VLC.

10.5 ATTACK SCENARIOS

Attackers might hold more abilities than considered in the initial design phase of a PLS mechanism. Even in a theoretically secure system, users might fail to use it properly and thus violate security assumptions (Section 10.5.1). Especially PLS mechanisms that rely on assumptions about the attacker's hardware and other physical constraints might break in a slightly different attack model, i.e., if the attacker has more sensitive PDs than the intended receivers (Section 10.5.2). Moreover, attackers can exploit additional information, such as partially known plaintexts (Section 10.5.3). Furthermore, active attackers who tamper with ongoing transmissions are missing in many attack models (Section 10.5.4). In the following, we categorize these attacker capabilities and describe the corresponding threats.

10.5.1 *User Failures*

Many approaches assume a trusted zone within which users can eliminate or identify eavesdroppers. This trusted zone can be limited by certain view angles as well as blockage by objects. If there are view angle variations introduced by users that are hard to predict, the eavesdropper needs receivers at multiple locations to ensure reception [Zha+14]. Yet, users might fail at appropriately changing the angle. Just relying on blockage without taking further actions can become an issue: light propagates through windows and can be eavesdropped on by an invisible spy next door, as previously described in Chapter 9. Building environments that are non-eavesdroppable is challenging and not practical for everyday scenarios. Hence, we need to assume that users make mistakes in detecting eavesdroppers within a trusted zone.

10.5.2 *Better Equipment*

Related work often assumes that attackers have similar hardware as the intended receivers. However, eavesdroppers might invest in more advanced hardware to achieve a sufficient SNR outside of a trusted zone. Additionally, multiple synchronized eavesdropping receivers can purge coding scheme effects [SLH14; Tip+13]. Therefore, we assume an "arms race" between legitimate transceivers and eavesdroppers, which cannot be won by either side—only an increase defense and attack efforts.

10.5.3 *Additional Information*

An attacker can include additional information sources that were not considered during design. Such information can be but is not limited to: known or partially known plaintext, the location of transceivers, and an estimation of the CSI. Note that the CSI entropy in VLC systems is lower than in RF systems because the phase information is missing—this is similar to the entropy reduction in legacy Wi-Fi systems if the signal strength is considered only [Jan+09]. We, therefore, assume that attackers gain a context-specific advantage over legitimate receivers by exploiting additional information sources.

10.5.4 *Active Attackers*

Active attackers can interfere with ongoing transmissions. By default, receivers adjust to the strongest received signal and reject remaining information as noise. Hence, signals can be overridden by injecting a stronger signal. We assume transmissions can be blocked or spoofed.

10.6 EVALUATION

In this section, we follow the structure from the previously introduced security mechanisms in Section 10.4. We discuss attacks on confidentiality (Section 10.6.1), localization (Section 10.6.2), and integrity (Section 10.6.3). We provide an overview of the attack success in Section 10.6.4.

10.6.1 *Confidentiality*

Many existing RF PLS attacks are not considered when designing visible light security. We apply these attacks to the VLC domain and also introduce an optimized attack for an OOK-based jamming scheme.

WYNER'S WIRETAP CHANNEL The main problem of Wyner's wiretap channel is the unknown eavesdropper's position. Certainly, the eavesdropper has at least a slightly different channel. However, it remains unknown how different this channel is and how much information the eavesdropper is missing in a practical setup. For example, the view angle dependent approaches by Zhang et al. assume that the user properly performs an angle variation [Zha+14]. Checking if an eavesdropper is nearby might not be trivial in IoT environments. When manually changing the view angle to exclude eavesdroppers with a limited set of receivers, the user needs to perform rotations correctly. User mistakes, i.e., failing in moving the screen in a secure manner, and unlimited hardware resources are not considered. However, an eavesdropper armed to the teeth might be more easily spotted by the user. Thus, the overall approach still features good security.

As already shown in recent work for RF communication, approaches based on ZFBF and OB are vulnerable to various attacks. An eavesdropper equipped with the same quantity of antennas as the transmitter can perform a known-plaintext attack [SLH14]. Even though the visible light channel only contains light intensities and no phase information, the same aspects hold true for VLC. Solving Equation 2 by using partially known plaintext reveals the unknown filter H_{AB} that allows equalizing the OB. In noisy environments, multiple rounds of training might be required to achieve good performance. This attack affects the approaches from Mostafa and Lampe, Zaid et al., and Le Minh et al. [ML14; Zai+15; LM+14]. Introducing a second channel as in [LM+14] still results in a filter of the same dimensions and does not add further security. Increasing the amount of transmitting LEDs as in [ML15]—by using cameras as arrays of thousands of receivers—only increases the efforts to launch this attack but cannot fundamentally prevent it.

FRIENDLY JAMMING Friendly jamming approaches are vulnerable to eavesdropping with additional antennas, as shown in the setup depicted in Figure 38. An attacker can place two antennas with equal distances towards a jammer, resulting in equal channels $H_{JE_1} = H_{JE_2}$. However, the data channels become different: $H_{DE_1} \neq H_{DE_2}$. When subtracting the signals received on both antennas, the attacker receives the following:

$$\begin{aligned} & RX_{E1} - RX_{E2} \\ & = (H_{JE_1} TX_J + H_{DE_1} TX_D) - (H_{JE_2} TX_J + H_{DE_2} TX_D) \\ & = (H_{DE_1} - H_{DE_2}) TX_D \end{aligned} \tag{3}$$

The original data can be reconstructed from the data signals with a small time and amplitude offset. If the jammer has a synchronization sequence, one can even assume known channels for oblivious antenna positions. This makes it possible to reconstruct data despite the presence of a strong jamming signal, as shown in [Tip+13].

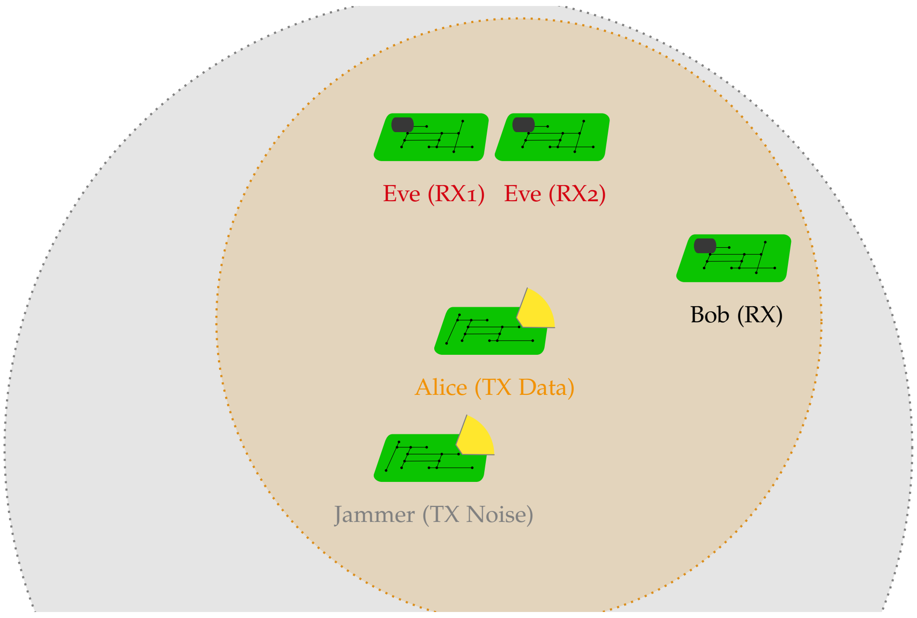


Figure 38: Attack on friendly jamming.

Friendly jamming with OOK as in [Cho+15] causes a single PD to receive the following intensity levels I_n:

$$RX_E = \begin{cases} I_0 & \text{if } TX_{Jam} = 0 \text{ and } TX_{Data} = 0 \\ I_1 & \text{if } TX_{Jam} = 1 \text{ and } TX_{Data} = 0 \\ I_2 & \text{if } TX_{Jam} = 0 \text{ and } TX_{Data} = 1 \\ I_3 & \text{if } TX_{Jam} = 1 \text{ and } TX_{Data} = 1 \end{cases} \tag{4}$$

An eavesdropper can always separate the two intensity levels I_0 and I_3. If the distances to the jammer or data source are different, I_1 and I_2 also become different for a single PD eavesdropper. This means that for OOK friendly jamming attacks, a single PD is sufficient, which is even worse than in the original attack on friendly jamming [Tip+13], and corresponds to the proposed "secure" scheme in Figure 36.

KEY-BASED ENCRYPTION If the system proposed by Ho, Duan, and Chen is streaming a key with the same length as the plaintext, this is similar to the Vernam OTP and proven information-theoretically secure [HDC15; Ver19]. Even though the Wi-Fi communication range is lowered to the range of VLC, one should have eavesdroppers in mind that are still in range but not detected by users or extend their range by using better hardware. Nevertheless, considering only the information-theoretically secure range restriction, this is an interesting and secure VLC building block.

10.6.2 *Localization and Authentication*

A general problem in localization are active attackers. As soon as they obtain the underlying secret information identifying a location, they can likely spoof arbitrary locations.

LOCALIZATION BY KNOWN PATTERNS Systems transmitting known illumination patterns require honest users and non-modified location beacons. Otherwise, active attackers knowing which illumination pattern belongs to which location can reply incorrect location information and even install their own transmitters to provide false locations. This sketch is similar to the *Skyhook* attack by Tippenhauer et al., a Wi-Fi localization system relying on publicly available access points and their location database [Tip+08]. Fake access points can impersonate access points providing location information.

LOCALIZATION BY RANDOM PATTERNS Random illumination patterns are more secure against impersonation. Attackers are required to capture and reply to such a random pattern. While attackers located slightly outside the reception area could improve their reception with better hardware, this only means that some location accuracy is lost. The located device then needs to send the received pattern back to the central random pattern instance to get its actual location, adding more dependencies to the system. To the best of our knowledge, such a system has not been implemented yet for VLC.

Distance bounding is also a randomness based solution, with honest entities proving their minimum distance by challenge-response [HK05]. Challenges are random, and responses depend on the key of a user. Thus, the distance proof is associated with an identity. The challenge-response round-trip time determines the distance between the location prover and verifier. Distance bounding can also be used in the VLC domain. In contrast to random illumination patterns, only users with a previously established key can proof their location. While this reduces attack vectors, not all application scenarios support previously established keys.

10.6.3 *Integrity*

Since integrity and confidentiality can be achieved similarly on the PHY, similar attack vectors exist for both. Hence, we only discuss attacks on polarization-based approaches in the following.

INTEGRITY BY POLARIZATION Approaches based on polarization appear promising, such as adding polarization information [JN00]. In general, polarization variations are harder to eavesdrop than intensity variations. Moreover, polarization can change or get lost on NLOS paths, because reflections—even on plane surfaces in incidence direction—depolarize except for horizontal and vertical linear polarization. Circular polarization might change into elliptic polarization. Besides, lateral scattering on rough surfaces reduces the polarization factor, which is very likely to appear in real environments [BS87].

Polarization can be eavesdropped with the more advanced hardware setup shown in Figure 39. Linear polarization can be in the range from 0–90°, and the better the match between the light polarization and filter polarization, the more light will pass. To detect polarization in a single polarized light source, an attacker requires one receiver with a polarization filter at 0° and another at 90°. Linear polarization angles in between will cause both filters to receive a reduced amount of light, which can be mapped to the angle. In case that not only the polarization but also the brightness is varied, an additional PD is required to compensate for the brightness changes that are not introduced by polarization. This polarization detection setup requires multiple polarization filters and PDs, but it is not too costly for an attacker.

After eavesdropping the polarization, an active attacker can block the original signal and spoof a signal containing the correct polarization. The attacker needs to adjust a polarization filter to the same polarization as Alice. Eve introduces a small time delay, depending on her processing time, as well as her polarization filter adjustment speed. Either way, delays could also be introduced by a larger distance between Alice and Bob or other signal processing operations, and thus are hard to detect.

10.6.4 *Attack Robustness*

We summarize the attack robustness in Table 6. Three attack robustness categories categorize the vulnerability of a VLC system. Already one successful attacker type makes an approach insecure. Attackers having better equipment can break most approaches, or at least lower their security to some extent, depending on a hardware competition

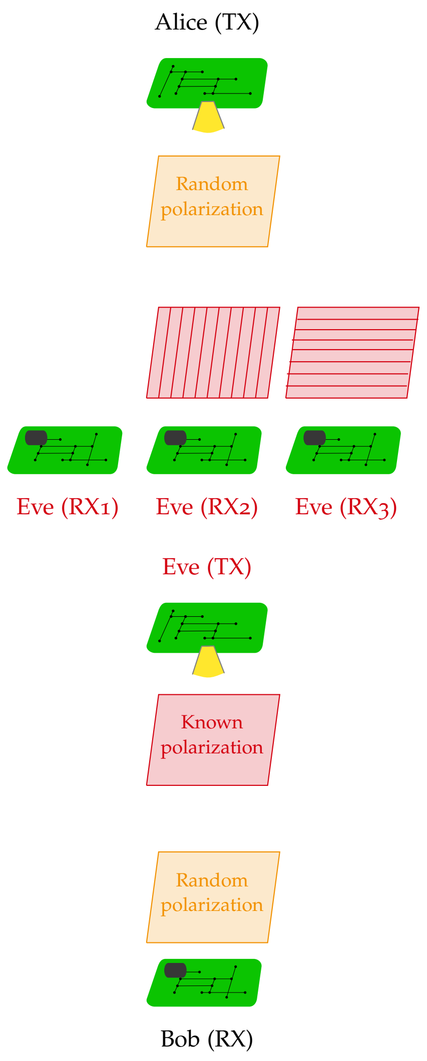


Figure 39: Eavesdropping and spoofing the polarization.

between attack and defense. Attackers having additional information, such as plaintext or a location database, can break the entire system design. Active attackers often need complicated hardware installations and operate within more constraints, but also can be very successful. As long as security does not depend on user interaction, it is not prone to user failures.

Table 6: Security against attackers.

	C: Wiretap channel	C: Jamming	C: Keys	A: Known patterns	A: Random patterns	I: Polarization
User failures	×	✓	~	✓	✓	✓
Better equipment	×	×	~	~	~	×
Additional information	×	✓	✓	×	✓	✓
Active attackers	✓	✓	✓	×	✓	×

Legend
× Attack available.
~ Attack possible, but not completely breaking the mechanism.
✓ Attacker not successful in this setting.

10.7 CONCLUSION

This chapter shows that the actual security for existing VLC PLS approaches needs to be reconsidered. Simply adapting PLS mechanisms from RF technologies for VLC is insufficient. Changing propagation characteristics and different hardware designs must be considered to avoid opening new vulnerabilities. Our attack model provides an intuition on how to design secure visible light applications on the PHY. In our evaluation, we outline VLC specific attacks, such as anti-jamming in OOK systems and eavesdropping on polarized light. With this contribution, we aim to raise the awareness of vulnerabilities in VLC systems and support future work in designing stronger security solutions.

MY CONTRIBUTION

This chapter is a survey of discussions on how to properly build security into VLC on the PHY. While our result was that VLC is not very suitable to build PLS despite its short transmission range, the reasoning for this is still valuable and interesting. Existing research on that topic mainly followed the approach to switch the PHY from RF to VLC and keep everything else unchanged. With the PHY changes introduced by VLC, this leads to insecure solutions. My main contribution to this part was to survey all existing related work to search for interesting or vulnerable approaches, as well as analyzing their security.

Section 10.3 is adapted from [CSH16, Sec. 1–2]. Section 10.4 is based on [CSH16, Sec. 3]. Section 10.5 is adapted from [CSH16, Sec. 4]. Section 10.6 is based on [CSH16, Sec. 5]. Section 10.7 is based on [CSH16, Sec. 6]. All figures are re-drawn.

Part V

DISCUSSION AND CONCLUSIONS

We discuss the results and additional interesting aspects in Chapter 11. Finally, we conclude this work in Chapter 12.

11 DISCUSSION AND FUTURE WORK

In this section, we reflect on the implications of the findings of this thesis. We discuss the future research directions that follow from our results. We apply the same structure as in Chapter 1 to review our goals, approaches, and contributions.

11.1 PUBLIC SECURITY AWARENESS

ACCEPTANCE OF SURVEILLANCE A topic out of the scope of this thesis is the psychological aspect of users continuing using devices that are insecure and may enable surveillance. There is a break-even point where devices become that useful and convenient that users no longer care about their privacy. The smartphones everybody carries around nowadays allow telecommunication providers tracking movements continuously. While it would be possible to apply k-anonymity to those datasets [GF15] or disable location logging completely, legislation requires telecommunication providers to log precise information and share it with law enforcement [BM18]. Nonetheless, most people choose to use a smartphone and always be connected. The continually growing Internet of Things (IoT) market also shows that users prioritize the functionality, e.g., their vacuum cleaners and fitness trackers, over their privacy. Giving up privacy towards an almost invisible party—the vendor and probably further governmental organizations—has become the norm.

11.2 VENDOR SUPPORT AND FIXES

TESTING METHODOLOGY When we responsibly disclose issues to a company, this happens quite late within the product's development cycle. The product has likely passed several internal test stages, but the details of these stay unknown to us external testers. For us, it is often impossible to differentiate if a bug should have previously been found internally, or if internal tests were not in place at all. During responsible disclosure, we informally asked how security testing has been done within the company. Answers vary but are interesting nonetheless: "We hired some nerd from Chaos Computer Club (CCC) two years ago.", "Red teaming is new to us, but we recently hired a new team of three people.", "We organized a hackathon and have an open bug bounty program.", or "We already had external professional penetration testers who took a look at our product."

These citations are anonymous on purpose, and also cover statements from companies with whom we interacted but are not part of this thesis. A study considering factors as company size, code complexity, bug severity, and security budget would be interesting. Such a study would probably be easy to perform by companies that offer external penetration tests, though they are usually bound to Non-Disclosure Agreements (NDAs).

Besides, we found that our testing capabilities are comparably advanced. For example, due to bugs we uncovered with *Frankenstein* [Rug19], we know that *Broadcom* and

Cypress are limited to over-the-air testing and do not have an emulation-based fuzzing toolchain. Based on our findings and knowledge, it would be possible to develop further automated testing frameworks for connected products and collaborate with industry to get these into their device testing process.

RESEARCH TOOLS *InternalBlue*, our Bluetooth research tool, is under active development. For a tool with such a narrow use case, it has surprisingly many clones and other *Github* activities [MC19]. *InternalBlue* fills a gap in Bluetooth testing tools and is used by a technical audience.

The initial version of *InternalBlue* was used to demonstrate the Key Negotiation of Bluetooth (KNOB) attack, as there were no other affordable and functional tools available to test this attack in practice [ATR19]. It is important to note that *InternalBlue* is not capable of working as Machine-in-the-Middle (MITM). Instead of attacking an ongoing communication between two other devices, it enables altering traffic on a device that is legitimately involved in information exchange. Thus, it can only be used to test if a device, which is currently connected to a test device, is vulnerable to the KNOB or Elliptic Curve Diffie-Hellman (ECDH) fixed coordinate attacks [Eli18] but not to carry them out on active connections. We published Proof of Concepts (PoCs) to test for these vulnerabilities without endangering any users, and we know that these tests are used by industry.

As future work, we aim at integrating *InternalBlue* and *Frankenstein* to provide an even more powerful Bluetooth testing toolchain. This can be expanded on the host side as well as for over-the-air testing. For example, as of now, *InternalBlue* supports testing of *Broadcom* chips on *Android*, *Linux*, *macOS*, and *iOS* devices. With *Frankenstein*, it can be used in reverse direction to test these operating systems' host implementations. Moreover, *Frankenstein* might be attachable to a Software-Defined Radio (SDR) for over-the-air testing with a fully controllable Bluetooth stack.

COMPANY STRUCTURES AND RESPONSIBILITIES Acquisitions and outsourcing can lead to severe bugs and low-quality code maintenance. Mature products require proper documentation and experienced developers. However, this kind of knowledge can get lost during acquisitions, or is never established when development is outsourced. The decision of acquisition or outsourcing does not fit into any of the phases in the life cycle. This can happen at any point in time and severely break the quality and the security of a product.

One example for acquisitions is *Broadcom*, who sold their IoT related Bluetooth branch to *Cypress* in 2016 [Cyp16]. As of now, in late 2019, *Cypress* is in the process of acquisition by *Infineon* [Inf19]. *Cypress* has at least two different Bluetooth codebases, which are the ones acquired from *Broadcom*, as well as their own Bluetooth Low Energy (BLE) only series, which is shipped in low-power IoT devices. *Infineon* was at least shipping Bluetooth chips in the past [Inf03], but less successful. Thus, there will be two or three technically different Bluetooth stacks within one company that have to be maintained. In practice, this leads to bugs that should be found during initial testing of a device but are missed. For example, *CVE-2019-18614* is a heap overflow that prevents a significant part of classic Bluetooth from working at all on the *CYW20735* evaluation board.

Outsourcing can lead to even worse problems. During responsible disclosure with one of the Bluetooth finder vendors, we uncovered that their app was violating their EU General Data Protection Regulation (GDPR) statements: they based those statements on their observations of the app, as they did not have the source code for it. This uncontrollable app then connected to their server infrastructure. They were aware of the general problem and already working on an internal app rewrite to gain control over the ecosystem they are running.

A recent study surveyed factors that led to security misconfigurations [Die+18]. The study considers the operator's perspective and human factors. It investigated root causes for unfixed known issues, and finds that sufficient structural and procedural mitigations already exist but often are not in place in practice. Based on their results, they propose six immediate action items, which include documentation, clear responsibilities, and processes and procedures. Further surveys on human and operator factors would be exciting, for example: Do employees think that product quality was maintained during and after an acquisition? Does a company's website list a security contact? Does this security contact answer if confronted with a plausible (real or fake) responsible disclosure request? Such questions could be answered by asking employees directly or by testing their responsible disclosure process.

11.3 USER SUPPORT AND FIXES

REPLACING COMMERCIAL APPS *Pebble* launched as a *Kickstarter* project to provide a smartwatch that does not depend on an external, commercial ecosystem. In February 2017, *Fitbit* acquired *Pebble* and merged existing devices into their ecosystem [Hea17]. Users who bought a *Pebble* smartwatch for their privacy started sharing their data with *Fitbit*. In November 2019, *Google* acquired *Fitbit* [Ost19]. Users who entrusted *Fitbit* with their health data now have to share this data with *Google* instead. As *Google* also owns a lot of other data, they could now correlate fitness records with further information about the users, even retrospectively. *Fitbit* and *Google* claim that they will not merge this data—however, this would be technically feasible. This example shows that it is advantageous for users to not share their data with any vendor, despite the burden of not using the officially supported app.

Thus, an open research question remains on how to develop alternate IoT apps that store information locally or on trusted devices [Dat19], and how to encourage users to use these apps. Given that commercial app developers have an interest in user data and more resources, this problem can only be solved by highly motivated users and an open-source community tackling this issue. New legislation that enforces IoT vendors providing their data in open formats would support this process.

11.4 REGULATIONS, SPECIFICATIONS, AND STANDARDIZATION IMPROVEMENTS

PRACTICAL PHYSICAL-LAYER SECURITY Considering their impact, contributions on Physical-Layer Security (PLS) are twofold. On the one hand, PLS can solve fundamental problems in secure device-to-device communication. On the other hand, most works that propose PLS mechanisms are theoretical. To implement these, SDRs are required, since off-the-shelf hardware typically does not decode the required wave properties.

Once proven to work in practice, PLS mechanisms can be built into the hardware. This enables light-weight security solutions in IoT environments.

This highly theoretical and otherwise SDR dependent deployment chain raises two issues: First, theoretical PLS mechanisms are often not secure in practice. They are proven to be secure but under simplistic assumptions, such as a single antenna attacker. Second, even suitable PLS mechanisms require years to become readily available in commodity hardware. The most wide-spread solution is distance bounding, but more sophisticated mechanisms rarely seen in the wild.

In general, it is possible to implement PLS on off-the-shelf smartphones. Depending on the wireless chip, binary patching can be used to send raw In-phase and Quadrature (IQ) samples [Sch18]. Binary patching and Physical Layer (PHY) changes are tailored to specific wireless chips. Transferring well-working PLS solutions into actual products requires collaborations with industry and standardization institutes. In contrast, the audio chips in smartphones can send and receive audio waves without binary patching [Put19]. Both technologies can be used to prototype practical PLS solutions and improve their adaption in widely used products in the future.

INDIVIDUAL SOLUTIONS VERSUS OPEN STANDARDS The IoT ecosystem architecture of various vendors is similar. Yet, they differ a lot in their concrete implementations. Hardware is highly customized for its specific purpose. On top, vendors often use proprietary protocols. Internals are not publicly documented, and without extensive analysis and reverse engineering, security issues remain undiscovered. An initial comparison as part of a Master's thesis within our group that includes our findings on *Fitbit* and *Neato* found that penetration testing guides do not cover most of our specific security issues [Kaz19].

We encourage vendors to use standard protocols. Obscurity of proprietary solutions does not only complicate adversarial testing, but it also prevents vendors from using standard tools for internal testing. In general, open standards experience better testing by the public. An attacker who wants to break a specific system will invest the time reverse engineering requires. Security by obscurity is not sufficient against such attackers.

From a research perspective, widely applied and open protocols are the most relevant. Scientific knowledge and industrial needs can be merged in this area. For example, researchers can apply formal methods to proof protocol security [CS07]. The effort of scientifically verifying protocols and their implementations only pays off for popular standards but not individual solutions.

Many topics of this discussions implicate that vendors should be more open to the scientific community, but science should also be more open to industry. Only this way, relevant problems in actual products can be solved together. Moreover, governmental organizations have responsibility in leading the market to produce privacy-providing, secure products.

12 CONCLUSIONS

In this thesis, we analyzed various security aspects throughout a product's life cycle of Internet of Things (IoT) devices. We tested the security of recent IoT devices after their release to the public, enhanced patching technologies for end-of-life products, and checked experimentally if claims on upcoming technologies hold.

Part i introduced the product life cycle and challenges associated with it.

In Part ii, we analyzed multiple complex IoT ecosystems regarding their architecture and security. Their technology stacks differ a lot, despite having many goals and communication paradigms in common. Because of their common architecture, similar attack models apply. In general, IoT vendors underestimate the damage that can be done by modified or copied devices. Thus, attackers who successfully mimic any component in an ecosystem are potent. For example, information stored on *Vorwerk/Neato* vacuum cleaning robots enables attackers access to the cloud infrastructure and spawning new devices. At least, *Fitbit* limits this damage to individual devices with device-specific keys. Our results show that trust assumptions made during the initial design of a product matter a lot—trust defines the impact of a single component attack on the whole ecosystem, as well as the complexity of bugfixes in production.

When a vendor decides to not patch certain bugs, binary patching enables modification of firmware without having its source code. This technique becomes interesting once a product reaches the end of life or if the users are interested in features that the vendor will not release. With proper reverse engineering efforts, binary patching turns closed-source lower-level implementations almost into open-source projects. These efforts are worth the results, as they allow device analysis and integrating functionality into those devices that are not available on the market. In Part iii, we applied binary patching in two very different systems, specifically, *Fitbit* fitness trackers and *Broadcom* Bluetooth chips. We extended the *Fitbit* fitness trackers to expose their raw accelerometer readings. Moreover, we are the first to make Bluetooth accessible on commodity hardware through binary patching on both, classic Bluetooth and Bluetooth Low Energy (BLE). For Bluetooth, we demonstrated that binary patching could be used for security analysis of the firmware itself and of remote devices. External researchers already made use of our tools, and we already used them to analyze and increase BLE connection reliability.

While binary patching can fix end-of-life products, security can already be enhanced before there is any concrete product. Security mechanisms can be put in place in Physical Layer (PHY) specifications, as shown in Part iv. As new PHY technologies emerge, their different properties change what can be realized with Physical-Layer Security (PLS). However, most implementations fail in practice and do not consider realistic attack scenarios. We have experimentally demonstrated that Visible Light Communication (VLC) is still susceptible to eavesdropping, despite being blocked by many objects. Moreover, VLC is less suitable for most traditional PLS approaches because the waveform does not contain any phase information.

When designing an ecosystem, security should be considered in all phases of the product's life cycle. Proper security requires a full-stack approach, going from the hardware itself up to servers and applications, and our findings cover all these aspects. Our results and following responsible disclosure had direct impact on the security of various popular products on the market.

[Spö+20] Michael Spörk, Jiska Classen, Carlo Alberto Boano, Matthias Hollick, and Kay Römer. "Improving the Reliability of Bluetooth Low Energy Connections." In: *International Conference on Embedded Wireless Systems and Networks (EWSN)*. 2020.

[Ada19] Adafruit. *Bluefruit LE Sniffer—Bluetooth Low Energy (BLE 4.0)—nRF51822*. 2019. URL: `https://www.adafruit.com/product/2269`.

[And19a] Android Open Source Project. *Android Security Bulletin—May 2019*. May 2019. URL: `https://source.android.com/security/bulletin/2019-05-01`.

[And19b] Android Open Source Project. *Hearing Aid Audio Support Using Bluetooth LE*. 2019. URL: `https://source.android.com/devices/bluetooth/asha`.

[Ang19] Hugues Anguelkov. *Reverse-engineering Broadcom Wireless Chipsets*. Quarkslab, 2019. URL: `https://blog.quarkslab.com/reverse-engineering-broadcom-wireless-chipsets.html`.

[ATR19] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. "The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR." In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019. ISBN: 978-1-939133-06-9. URL: `https://www.usenix.org/conference/usenixsecurity19/presentation/antonioli`.

[App19a] Apple. *About the security content of iOS 13*. 2019. URL: `https://support.apple.com/en-us/HT210606`.

[App19b] Apple. *About the security content of macOS Mojave 10.14.6, Security Update 2019-004 High Sierra, Security Update 2019-004 Sierra*. July 2019. URL: `https://support.apple.com/en-us/HT210348`.

[Blu19a] Bluetooth SIG. *Bluetooth Core Specification 5.1*. Jan. 2019. URL: `https://www.bluetooth.com/specifications/bluetooth-core-specification`.

[Blu19b] Bluetooth SIG. *GATT Services*. 2019. URL: `https://www.bluetooth.com/specifications/gatt/services/`.

[Blu19c] Bluetooth SIG. *The Link Manager Version Parameter*. Bluetooth, 2019. URL: `https://www.bluetooth.com/specifications/assigned-numbers/link-manager`.

[Bra19] Mathias Brandt. *Statistik der Woche: Wer für DSGVO-Verstöße zahlte*. 2019. URL: `https://www.heise.de/tr/artikel/Statistik-der-Woche-Wer-fuer-DSGVO-Verstoesse-zahlte-4601473.html`.

[CCC19] CCC Event Blog. *36C3 content teams running full steam*. 2019. URL: `https://events.ccc.de/2019/10/29/36c3-content-teams-running-full-steam/`.

[Cla19a] Jiska Classen. *All Wireless Communication Stacks are Equally Broken*. Leipzig: 36. Chaos Communication Congress, 2019.

[Cla19b] Jiska Classen. *Bluetooth H4 Broadcom Wireshark plugin from the InternalBlue project*. `https://github.com/seemoo-lab/h4bcm_wireshark_dissector`. 2019.

[Cla19c] Jiska Classen. *Vacuums in the Cloud*. Karlsruhe: OWASP, 2019.

[CE19] Jiska Classen and Johannes Eger. *Smart Vacuum Cleaners as Remote Wiretapping Devices*. Wien: Easterhegg, 2019. URL: `https://media.ccc.de/v/eh19-157-smart-vacuum-cleaners-as-remote-wiretapping-devices`.

[CH19] Jiska Classen and Matthias Hollick. "Inside Job: Diagnosing Bluetooth Lower Layers Using Off-the-Shelf Devices." In: *12th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec). Replicability label*. 2019. DOI: `10.1145/3317549.3319727`. **Part of this thesis.**

[CM19a] Jiska Classen and Dennis Mantz. *Bluetooth, Does it Spark Joy?* Heidelberg: TROOPERS, 2019. URL: `https://www.troopers.de/troopers19/agenda/smsm3l/`.

[CM19b] Jiska Classen and Dennis Mantz. *Playing with Bluetooth*. Darmstadt: MRMCD, 2019. URL: `https://media.ccc.de/v/2019-185-playing-with-bluetooth`.

[CM19c] Jiska Classen and Dennis Mantz. *Reversing and Exploiting Broadcom Bluetooth*. Montreal: REcon, 2019. URL: `https://cfp.recon.cx/reconmtl2019/talk/EQTRGU/`.

[CS19] Jiska Classen and Kristoffer Schneider. *Nexmon for Bluetooth*. `https://github.com/seemoo-lab/nexmon/tree/bluetooth-wip`. 2019.

[Cox19] Kate Cox. *Equifax to pay USD 575M for data breach, promises to protect data next time*. 2019. URL: `https://arstechnica.com/tech-policy/2019/07/equifax-to-pay-575m-for-data-breach-promises-to-protect-data-next-time/`.

[Dam19] Damien Cauquil. *Bluetooth Low Energy Swiss-army knife*. `https://github.com/virtualabs/btlejack`. 2019.

[Dat19] Databox Project. *EPSRC Project on Privacy-Aware Personal Data Platform*. 2019. URL: `https://www.databoxproject.uk`.

[Dul19] Thomas Dullien. *Rashomon of disclosure*. 2019. URL: `http://addxorrol.blogspot.com/2019/08/rashomon-of-disclosure.html`.

[Dwo19] Patrick Dworski. "A Study on Proprietary Communication Protocols Used in TETRA Hardware Components." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2019.

[Ege19] Johannes Eger. "Analyzing Firmware and Cloud Security of a Premium IoT Ecosystem." Supervised by Fabian Ullrich and Jiska Classen. Master thesis. TU Darmstadt, 2019.

[Eur19] European Telecommunications Standards Institute. *CYBER; Cyber Security for Consumer Internet of Things; ETSI TS 103 645 V1.1.1*. 2019.

[Exp19] Express Logic, Inc. *ThreadX G4.0c Documentation*. 2019. URL: http://www.ece.ualberta.ca/~cmpe490/documents/ghs/405/threadxug_g40c.pdf.

[Fit19a] Fitbit. *Fitbit Sensors API*. https://dev.fitbit.com/reference/device-api/sensors/. 2019.

[Fit19b] Fitbit. *Q2'19 Earning Summary*. 2019.

[Gie19] Dennis Giese. "Security Analysis of the Xiaomi IoT Ecosystem." Master thesis. TU Darmstadt, 2019.

[GP19] Xiling Gong and Peter Pi. "Exploiting Qualcomm WLAN and Modem Over The Air." In: *DEF CON 27*. 2019. URL: https://media.defcon.org/DEF%20CON%2027/DEF%20CON%2027%20presentations/DEFCON-27-Xiling-Gong-Peter-Pi-Exploiting-Qualcomm-WLAN-and-Modem-Over-The-Air.pdf.

[Gre19] Great Scott Gadgets. *Ubertooth One—Your BLE Hacking Tool*. 2019. URL: https://www.attify-store.com/products/ubertooth-one-your-ble-hacking-tool.

[Gre19] Andy Greenberg. *The Clever Cryptography Behind Apple's 'Find My' Feature*. WIRED, 2019. URL: https://www.wired.com/story/apple-find-my-cryptography-bluetooth/.

[Gro19] Carolin Gross. "A researcher's guide to the Fitbit Ionic smartwatch." Supervised by Jiska Classen and Daniel Wegemer. Master thesis. TU Darmstadt, 2019.

[Inf19] Infineon. *Infineon to acquire Cypress, strengthening and accelerating its path of profitable growth*. 2019. URL: https://www.infineon.com/cms/en/about-infineon/press/press-releases/2019/INFXX201906-074.html.

[Jis19] Jiska Classen and Daniel Wegemer. *Fitbit Firmware Modifications*. https://github.com/seemoo-lab/fitness-firmware. 2019.

[Kaz19] Philip Kazmeier. "Applicability of Penetration Testing Guides for the Internet of Things." Supervised by Max Maass and Fabian Ullrich. Master thesis. TU Darmstadt, 2019.

[Kor19] Joxean Koret. *Diaphora*. 2019. URL: https://github.com/joxeankoret/diaphora.

[Lin19] Linux Kernel. *linux/drivers/bluetooth/btbcm.c:btbcm_patchram*. 2019. URL: https://github.com/torvalds/linux/blob/master/drivers/bluetooth/btbcm.c#L108.

[MC19] Dennis Mantz and Jiska Classen. *InternalBlue Broadcom Bluetooth Experimentation Framework*. https://github.com/seemoo-lab/internalblue. 2019.

[Man+19] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. "InternalBlue - Bluetooth Binary Patching and Experimentation Framework." In: *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2019. DOI: 10.1145/3307334.3326089. **Part of this thesis.**

[May19] René Mayrhofer. *Disclosing Proof-of-Concept (PoC) exploits for vulnerabilities: A defender's point of view*. 2019. URL: `https://www.mayrhofer.eu.org/post/vulnerability-disclosure-is-positive/`.

[Mei+19] Richard Meister, Jiska Classen, Muhammad Saad Saud, Marcos Katz, and Matthias Hollick. "Practical VLC to WiFi Handover Mechanisms." In: *Co-Wireless*. 2019.

[Mü19] Uwe Müller. "PowerPC Binary Patching and dissecting of TETRA Base Station." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2019.

[New19] Lily Hay Newman. *You Can Jailbreak Your iPhone Again (But Maybe You Shouldn't)–Apple reintroduced a previously fixed bug in iOS 12.4, which has led to a jailbreak revival*. 2019. URL: `https://www.wired.com/story/ios-jailbreak-new/`.

[Nor19a] Nordic Semiconductor. *nRF5 SDK Documentation*. 2019. URL: `https://developer.nordicsemi.com/nRF5_SDK/doc/`.

[Nor19b] Nordic Semiconductor. *nRF8001 Bluetooth Chip Product Specification*. `http://www.nordicsemi.com/eng/nordic/download_resource/17534/16/6078997/2981`. 2019.

[Ost19] Rick Osterloh. *Helping more people with wearables: Google to acquire Fitbit*. Google, Nov. 2019. URL: `https://blog.google/products/hardware/agreement-with-fitbit`.

[Put19] Florentin Putz. "Secure Device Pairing Using Short-Range Acoustic Communication." Supervised by Flor Álvarez and Jiska Classen. Master thesis. TU Darmstadt, 2019.

[QV19] Nguyen Anh Quynh and Dang Hoang Vu. *Unicorn Engine*. 2019. URL: `https://github.com/unicorn-engine/unicorn`.

[Rug19] Jan Ruge. "Dynamic Bluetooth Firmware Analysis." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2019.

[Sch19] Sabine Schiner. *Hacker im Wohnzimmer - Wissenschaftler der TU Darmstadt decken Schwachstellen bei Vorwerk-Saugroboter auf*. 2019. URL: `https://www.echo-online.de/panorama/wissenschaft/wissenschaft/wissenschaftler-der-tu-darmstadt-decken-schwachstellen-bei-vorwerk-saugroboter-auf_20434863`.

[Sta19] Statista. *Unit sales of the Apple iPhone worldwide from 2007 to 2018 (in millions)*. 2019. URL: `https://www.statista.com/statistics/276306/global-apple-iphone-sales-since-fiscal-year-2007/`.

[Ste19] Steffen Kreis, Johannes Riedel, Tobias Krichel, Jiska Classen. *Fitbit Open Source Android App*. `https://github.com/seemoo-lab/fitness-app`. 2019.

[Sti19] Stiftung Warentest. *Nur zwei Staubsauger-Roboter reinigen gut*. 2019. URL: `https://www.test.de/Saugroboter-im-Test-4806685-0/`.

[Tro19] Kirsten Tromnau. *Die Spione im eigenen Haus*. 2019. URL: `https://www.swr.de/swraktuell/radio/netzagent/Die-Spione-im-eigenen-Haus,av-o1150171-100.html`.

[UC19] Fabian Ullrich and Jiska Classen. *Vacuum Cleaning Security—Pinky and the Brain Edition*. Las Vegas: DEF CON 27, 2019. URL: `https://www.defcon.org/html/defcon-27/dc-27-speakers.html#jiska`.

[Ull+19] Fabian Ullrich, Jiska Classen, Johannes Eger, and Matthias Hollick. "Vacuums in the Cloud: Analyzing Security in a Hardened IoT Ecosystem." In: *The 13th USENIX Workshop on Offensive Technologies (WOOT)*. 2019. **Part of this thesis.**

[Vor19] Vorwerk. *Vorwerk Unternehmenspräsentation 15. Oktober 2019*. 2019. URL: `https://corporate.vorwerk.de/fileadmin/data/master_corporate/04_Presse/Publikationen/Vorwerk_Unternehmenspraesentation_2019.pdf`.

[WAR19] WARP. *WARP Project*. 2019. URL: `http://warpproject.org`.

[Wal19] Tim Walter. "Fuzzing the Linux Bluetooth Stack." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2019.

[WGC19] Daniel Wegemer, Carolin Groß, and Jiska Classen. *A Security Researchers Guide into the Fitbit Ecosystem*. Las Vegas: IoT Village at DEF CON 27, 2019. URL: `https://www.iotvillage.org/`.

[Wel+19] Max Weller, Jiska Classen, Fabian Ullrich, Erik Tews, and Matthias Hollick. "Lost and Found: Stopping Bluetooth Finders from Leaking Private Information." In: *Under submission*. 2019. **Part of this thesis.**

[Zep19] Danny Zepeda. *Here's how long Apple supports older iPhone models—The iPhone 5s got six years of yearly iOS updates before it was cut off*. 2019. URL: `https://www.imore.com/apples-continual-software-support-iphones-major-reason-theyre-popular`.

[Zer19] Zerodium. *Our Exploit Acquisition Program*. 2019. URL: `https://zerodium.com/program.html`.

[Zyn19] Zynamics. *BinDiff*. 2019. URL: `https://www.zynamics.com/bindiff.html`.

[pan19] pancake. *radare2*. 2019. URL: `https://github.com/radare/radare2`.

[BM18] Ulf Buermeyer and Andre Meister. *Funkzellenabfrage: Die alltägliche Rasterfahndung unserer Handydaten*. Leipzig, 2018. URL: `https://media.ccc.de/v/35c3-9972-funkzellenabfrage_die_alltagliche_rasterfahndung_unserer_handydaten`.

[Cla18a] Jiska Classen. *Pinky & Brain are Taking Over the World with Vacuum Cleaners*. Darmstadt: MRMCD, 2018. URL: `https://media.ccc.de/v/2018-124-pinky-brain-are-taking-over-the-world-with-vacuum-cleaners`.

[Cla18b] Jiska Classen. *Praktische IoT-Sicherheit am Beispiel von Wearables und Smart Home*. Frankfurt: Deka Bank, 2018.

[Cla18c] Jiska Classen. *ma1lm4n.mp4*. Leipzig: 35. Chaos Communication Congress, 2018. URL: `https://media.ccc.de/v/35c3-9566-lightning_talks_day_2#t=219`.

[CM18] Jiska Classen and Dennis Mantz. *Dissecting Broadcom Bluetooth*. Leipzig: 35. Chaos Communication Congress, 2018. URL: `https://media.ccc.de/v/35c3-9498-dissecting_broadcom_bluetooth`. **Talk in a lecture hall with 5000 seats and Bluetooth live demos.**

[CW18a] Jiska Classen and Daniel Wegemer. *Create your own Fitness Tracker Firmware*. Montreal: REcon, 2018. URL: `https://recon.cx/2018/montreal/schedule/events/118.html`.

[CW18b] Jiska Classen and Daniel Wegemer. *Hacking your Fitbit*. Würzburg: Easterhegg, 2018. URL: `https://media.ccc.de/v/TNYPFB`.

[Cla+18a] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. "Anatomy of a Vulnerable Fitness Tracking System: Dissecting the Fitbit Cloud, App, and Firmware." In: *PACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*. 2018. **Part of this thesis.**

[Cla+18b] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. *Demo. Modified Fitbit Firmware: Reach your Daily Goals within Seconds*. Singapore, 2018. URL: `http://ubicomp.org/ubicomp2018/program/demo-schedule.pdf`.

[Die+18] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. "Investigating System Operators' Perspective on Security Misconfigurations." In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 1272–1289.

[Eli18] Eli Biham and Lior Neumann. *Breaking the Bluetooth Pairing: Fixed Coordinate Invalid Curve Attack*. 2018. URL: `http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf`.

[Elv18] Simon Elvery. *My devices are sending and receiving data every two seconds, sometimes even when I sleep*. 2018. URL: `https://www.abc.net.au/news/2018-11-16/datalife-i-spied-on-my-phone-and-here-is-what-i-found/10496450`.

[Fit18] Fitbit. *What's changed in the latest Fitbit device update?* 2018. URL: `https://help.fitbit.com/articles/en_US/Help_article/1372`.

[Han18] Matthias Hanreich. "Security Analysis and Firmware Modification of Fitbit Fitness Trackers." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2018.

[Hes18] Hessenschau. *Kommunikation trotz Netzausfall*. 2018. URL: `https://www.hessenschau.de/tv-sendung/hessenschau---ganze-sendung,video-77832.html`.

[Huk18] Wolfram Huke. *Spion im Wohnzimmer*. 2018. URL: `https://www.mdr.de/video/mdr-videos/c/video-175524.html`.

[Kie18a] Jason Kielpinski. *Security in a Vacuum: Hacking the Neato Botvac Connected, Part One*. 2018. URL: `https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2018/march/security-in-a-vacuum-hacking-the-neato-botvac-connected-part-1/`.

[Kie18b] Jason Kielpinski. *Security in a Vacuum: Hacking the Neato Botvac Connected, Part Two.* 2018. URL: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/april/security-in-a-vacuum-hacking-the-neato-botvac-connected-part-2/.

[Lib] *LibTomCrypt*. https://github.com/libtom/libtomcrypt. 2018.

[Liu18] John Liu. *Europe's GDPR causes malfunction of smart home devices: Report.* 2018. URL: https://www.asmag.com/showpost/26577.aspx.

[Man18] Dennis Mantz. "InternalBlue - A Bluetooth Experimentation Framework Based on Mobile Device Reverse Engineering." Supervised by Matthias Schulz and Jiska Classen. Master thesis. TU Darmstadt, 2018.

[MG18] Amrit Mundra and Hong Guan. *Secure Boot on Embedded Sitara Processors.* 2018. URL: http://www.ti.com/lit/wp/spry305a/spry305a.pdf.

[Neu18] Sven Neubauer. "Angriffsanalyse einer TETRA-Basisstation." Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2018.

[Pla18] Marco Plaue. "Sicherheit funkferngesteuerter Rangierlokomotiven." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2018.

[RE18] Bill Ray and Jon Erensen. "Market Share Analysis: Wireless LAN, Bluetooth, GPS and NFC Semiconductors, Worldwide, 2017." In: *Gartner, Inc.* G00360119 (2018).

[Sch18] Matthias Schulz. "Teaching Your Wireless Card New Tricks: Smartphone Performance and Security Enhancements Through Wi-Fi Firmware Modifications." PhD thesis. Technische Universität, 2018.

[Tre18] Moritz Tremmel. *DoS-Angriff auf Bluetooth-Chips von Broadcom.* 2018. URL: https://www.golem.de/news/sicherheitsluecke-dos-angriff-auf-bluetooth-chips-von-broadcom-1901-138454.html.

[Ull18a] Fabian Ullrich. "Analysing and Evaluating Interface, Communication, and Web Security in Productive IoT Ecosystems." Supervised by Jiska Classen and Max Maass. Master thesis. TU Darmstadt, 2018.

[Ull18b] Fabian Ullrich. *Nello (nicht ganz) allein zu Haus.* Darmstadt: MRMCD, 2018. URL: https://media.ccc.de/v/2018-123-nello-nicht-ganz-allein-zu-haus.

[Wil18] Elliot Williams. *35C3: Finding Bugs In Bluetooth.* 2018. URL: https://hackaday.com/2018/12/30/finding-bugs-in-bluetooth/.

[Wol18] Markus Wolsiffer. *Datenschutz bei Wearables - Wie sicher sind meine Daten bei Smartwatch und Co.?* 2018. URL: https://www.zdf.de/verbraucher/volle-kanne/datenschutz-bei-wearables-102.html.

[Zsc18] Peter Zschunke. *Forscher warnen vor Bluetooth auf älteren Smartphones.* 2018.

[Ada17] Adafruit. *Adafruit nRF8001.* 2017. URL: https://github.com/adafruit/Adafruit_nRF8001/tree/master/utility.

[Ahm17] Muneeb Ahmed. "Improving a Linux Device Driver for Visible Light Communication." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

[Apv17] Axelle Apvrille. “Research on Fitbit Flex.” In: Available at: http://www.fortiguard.com/events/1869/research-on-fitbit-flex, 2017.

[Ato17] Atollic. *TrueSTUDIO*. https://atollic.com/truestudio/. 2017.

[Ay17] Serafettin Ay. “Detecting WiFi Covert Channels.” Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

[Che+17] Joe Chen, Daniel Steinmetzer, Jiska Classen, Edward Knightly, and Matthias Hollick. “Pseudo Lateration: Millimeter-Wave Localization Using a Single RF Chain.” In: *Wireless Communications and Networking Conference*. IEEE. 2017.

[Cla17] Jiska Classen. *Leaking and Modifying Fitbit Data*. Frankfurt: Continental AG, 2017.

[CW17a] Jiska Classen and Daniel Wegemer. *Doping your Fitbit*. Leipzig: 34. Chaos Communication Congress, 2017. DOI: 10.5446/34791. URL: https://media.ccc.de/v/34c3-8908-doping_your_fitbit.

[CW17b] Jiska Classen and Daniel Wegemer. *Leaking and Modifying Fitbit Data*. Darmstadt: MRMCD, 2017. URL: https://media.ccc.de/v/3T9E8Y.

[Dan17] Dany. “Fitbit Flex: switching between encrypted and unencrypted mode.” In: Available at: https://www.freelists.org/post/galileo/Fitbit-Flex-switching-between-encrypted-and-unencrypted-mode, 2017.

[Fer+17a] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. “Breaking Fitness Records without Moving: Reverse Engineering and Spoofing Fitbit.” In: *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. Springer, Cham. 2017.

[Fer+17b] Hossein Fereidooni, Tommaso Frassetto, Markus Miettinen, Ahmad-Reza Sadeghi, and Mauro Conti. “Fitness Trackers: Fit for Health but Unfit for Security and Privacy.” In: *IEEE International Workshop on Safe, Energy-Aware, & Reliable Connected Health (CHASE workshop: SEARCH)*. 2017.

[Fit17] Fitbit. *Fitbit Ultra Setup*. https://www.fitbit.com/de/setup/ultra. 2017.

[GNU17] GNU Operating System. *GDB: The GNU Project Debugger*. https://www.gnu.org/software/gdb/. 2017.

[Hea17] Brian Heater. *Fitbit reveals it paid $23 million to acquire Pebble's assets*. Verizon Media, 2017. URL: https://techcrunch.com/2017/02/22/fitbit-pebble-acquisition/.

[Hex17] Hex-Rays. *IDA Pro*. https://www.hex-rays.com/. 2017.

[IDC17] IDC. *Worldwide Quarterly Wearable Device Tracker*. 2017.

[Jü17] Jannik Jürgens. “TETRA Security Analysis by Fuzzing.” Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

[Kor17] Tim Kornhuber. “Implementation of a physical layer for visible light communication using the OpenVLC platform.” Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2017.

[Kos17] Felix Kosterhon. “Absicherung von SCADA-Protokollen.” Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2017.

[Kü17] Michael Kümpel. “Implementierung des unteren MAC-Layers für die Open-VLC-Hardware.” Supervised by Jiska Classen. Bachelor thesis. TU Darmstadt, 2017.

[McE17] Brian McEvoy. *34C3: Fitbit Sniffing and Firmware Hacking*. 2017. URL: `https://hackaday.com/2017/12/29/34c3-fitbit-sniffing-and-firmware-hacking/`.

[Mei17] Richard Meister. “Design and Evaluation of a Hybrid SDR Testbed For Visible Light Communication and Wi-Fi.” Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2017.

[MWZH17] Cristina Mihale-Wilson, Jan Zibuschka, and Oliver Hinz. “About User Preferences and Willingness to Pay for a Secure and Privacy Protective Ubiquitous Personal Assistant.” In: (2017).

[STM17a] STMicroelectronics. *STM32L141UC datasheet*. 2017. URL: `www.st.com/resource/en/datasheet/stm32l151cc.pdf`.

[STM17b] STMicroelectronics. *STM32L141UC reference manual*. `www.st.com/resource/en/reference_manual/cd00240193.pdf`. 2017.

[Sch17] Maarten Schellevis. *Maarten committed 3601372: Add a utility to decrypt older dumps*. `https://bitbucket.org/benallard/galileo/commits/3601372658e5e6da271300656d4ec503c5c87ddc`. 2017.

[SWH17] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. *Nexmon: The C-based Firmware Patching Framework*. `https://nexmon.org`. 2017.

[iFi17] iFixit. *Fitbit Flex teardown*. `https://www.ifixit.com/Teardown/Fitbit+Flex+Teardown/16050`. 2017.

[CSH16] Jiska Classen, Daniel Steinmetzer, and Matthias Hollick. “Opportunities and Pitfalls in Securing Visible Light Communication on the Physical Layer.” In: *Proceedings of the 3rd Workshop on Visible Light Communication Systems*. ACM. 2016. **Part of this thesis.**

[CSM16] Eric Clausing, Michael Schiefer, and Maik Morgenstern. “AV-TEST Analysis of Fitbit Vulnerabilities.” In: `https://www.av-test.org/fileadmin/pdf/avtest_2016-04_fitbit_vulnerabilities.pdf`, 2016.

[Cou16] Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC*. 2016.

[Cyp16] Cypress. *Cypress to Acquire Broadcom’s Wireless Internet of Things Business*. 2016. URL: `https://www.cypress.com/news/cypress-acquire-broadcom-s-wireless-internet-things-business-0`.

[GDS16] Rohit Goyal, Nicola Dragoni, and Angelo Spognardi. “Mind the Tracker You Wear: A Security Analysis of Wearable Health Trackers.” In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. Pisa, Italy, 2016.

[Gu+16] Wenjun Gu, Mohammadreza Aminikashani, Peng Deng, and Mohsen Kavehrad. "Impact of multipath reflections on the performance of indoor visible light positioning systems." In: *Journal of Lightwave Technology* 34.10 (2016).

[Kwi16] Jan-Pascal Kwiotek. "TETRA Fuzzing." Supervised by Jiska Classen. Master thesis. TU Darmstadt, 2016.

[Pfe16] Martin Pfeiffer. "Location Privacy of Digital Trunked Radio." Supervised by Jiska Classen and Robin Klose. Master thesis. TU Darmstadt, 2016.

[Pfe+16] Martin Pfeiffer, Jan-Pascal Kwiotek, Jiska Classen, Robin Klose, and Matthias Hollick. "Analyzing TETRA Location Privacy and Network Availability." In: *Proceedings of the 6th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM. 2016.

[PwC16] PwC. *The Wearable Life 2.0*. `https://www.pwc.com/ee/et/publications/pub/pwc-cis-wearables.pdf`. 2016.

[RCT16] Mahmudur Rahman, Bogdan Carbunar, and Umut Topkara. "Secure Management of Low Power Fitness Trackers." In: *IEEE Transactions on Mobile Computing* 15.2 (2016).

[Rei16] Hugo Reinaldo. "Hello Quark! Fitbit firmware reversing (Lessons learned)." In: *AlligatorCon* (2016).

[Sch+16] Maarten Schellevis, Bart Jacobs, Carlo Meijer, and Joeri de Ruiter. "Getting access to your own Fitbit data." MA thesis. Radboud University, 2016.

[SCH16a] Daniel Steinmetzer, Jiska Classen, and Matthias Hollick. "Exploring Millimeter-Wave Network Scenarios with Ray-tracing based Simulations in mmTrace." In: *IEEE Infocom 2016 Poster Presentation*. IEEE. 2016.

[SCH16b] Daniel Steinmetzer, Jiska Classen, and Matthias Hollick. "mmTrace: Modeling Millimeter-wave Indoor Propagation with Image-based Ray-tracing." In: *Millimeter-wave Networking Workshop*. IEEE. 2016.

[AGK15] Mohammadreza Aminikashani, Wenjun Gu, and Mohsen Kavehrad. "Indoor positioning in high speed OFDM visible light communications." In: *arXiv preprint arXiv:1505.01811* (2015).

[Apv15] Axelle Apvrille. "Fitness Tracker: Hack In Progress." In: Available at: `https://hackinparis.com/data/slides/2015/axelle_aprville_hackinparis.pdf`, 2015.

[Cho+15] Chi-Wai Chow, Yang Liu, Chien-Hung Yeh, Chung-Yen Chen, Chao-Nan Lin, and Dar-Zu Hsu. "Secure communication zone for white-light LED visible light communication." In: *Optics Communications* 344 (2015).

[Cla15a] Jiska Classen. *Building and Breaking Wireless Security*. Hamburg: 32. Chaos Communication Congress, 2015. URL: `https://media.ccc.de/v/32c3-7119-building_and_breaking_wireless_security`. **Talk in a lecture hall with 3000 seats.**

[Cla15b] Jiska Classen. *Wireless Physical Layer Security*. Darmstadt: MRMCD, 2015. URL: `https://media.ccc.de/v/MRMCD15-7011-wireless_physical_layer_security`.

[Cla+15a] Jiska Classen, Johannes Braun, Florian Volk, Matthias Hollick, Johannes Buchmann, and Max Mühlhäuser. "A Distributed Reputation System for Certification Authority Trust Management." In: *Proceedings of IEEE Trust-Com*. Vol. 1. IEEE. 2015.

[Cla+15b] Jiska Classen, Joe Chen, Daniel Steinmetzer, Matthias Hollick, and Edward Knightly. "The Spy Next Door: Eavesdropping on High Throughput Visible Light Communications." In: *Proceedings of the 2nd International Workshop on Visible Light Communications Systems*. ACM. 2015. **Part of this thesis.**

[CSH15] Jiska Classen, Matthias Schulz, and Matthias Hollick. "Practical Covert Channels for WiFi Systems." In: *IEEE Conference on Communications and Network Security*. IEEE. 2015.

[For15] Forbes. *Fitbit Disputes Claim Fitbit Trackers Can Be Hacked And Infect PCs*. 2015.

[GF15] Marco Gramaglia and Marco Fiore. "Hiding Mobile Traffic Fingerprints with GLOVE." In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM. 2015, p. 26.

[Hed15] Johan Hedberg. *Release of BlueZ 5.36*. 2015. URL: `http://www.bluez.org/release-of-bluez-5-36/`.

[HDC15] Siu-Wai Ho, Jialong Duan, and Chung Shue Chen. "Location-based information transmission systems using visible light communications." In: *Transactions on Emerging Telecommunications Technologies* (2015).

[Li+15] Tianxing Li, Chuankai An, Andrew T. Campbell, and Xia Zhou. "HiLight: Hiding Bits in Pixel Translucency Changes." In: *SIGMOBILE Mob. Comput. Commun. Rev.* 18.3 (2015).

[LMGGB15] F Javier Lopez-Martinez, Gerardo Gomez, and José María Garrido-Balsells. "Physical-Layer Security in Free-Space Optical Communications." In: *IEEE Photonics Journal* (2015). ISSN: 1943-0655. DOI: `10.1109/JPHOT.2015.2402158`.

[ML15] A. Mostafa and L. Lampe. "Pattern synthesis of massive LED arrays for secure visible light communication links." In: *International Conference on Communication Workshop (ICCW)*. 2015.

[Nea15] Neato. *Programmer's Manual v3.1*. 2015. URL: `https://github.com/jero%65nterheerdt/neato-serial/blob/master/XV-ProgrammersManual-3_1.pdf`.

[Org15] Organisation Internationale de Normalisation. *IEC 60601-1-11:2015: Medical electrical equipment – Part 1-11: General requirements for basic safety and essential performance – Collateral standard: Requirements for medical electrical equipment and medical electrical systems used in the home healthcare environment*. 2015.

[Ste+15] Daniel Steinmetzer, Joe Chen, Jiska Classen, Edward Knightly, and Matthias Hollick. "Eavesdropping with Periscopes: Experimental Security Analysis of Highly Directional Millimeter Waves." In: *IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2015.

[Tho15] Simon Thomsen. *Extramarital affair website Ashley Madison has been hacked and attackers are threatening to leak data online*. 2015. URL: https://www.businessinsider.com/cheating-affair-website-ashley-madison-hacked-user-data-leaked-2015-7?IR=T.

[Yan+15] Nan Yang, Lifeng Wang, Giovanni Geraci, Maged Elkashlan, Jinhong Yuan, and Marco Di Renzo. "Safeguarding 5G wireless communication networks using physical layer security." In: *IEEE Communications Magazine* 53.4 (2015).

[Zai+15] Hajar Zaid, Zouheir Rezki, Anas Chaaban, and Mohamed Slim Alouini. "Improved achievable secrecy rate of visible light communication with cooperative jamming." In: *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2015.

[Bra+14] Johannes Braun, Florian Volk, Jiska Classen, Johannes Buchmann, and Max Mühlhäuser. "CA Trust Management for the Web PKI." In: *Journal of Computer Security* 22.6 (2014).

[Cla14] Jiska Classen. "Reputation Systems for Trust Management in the Web PKI." Supervised by Johannes Braun and Florian Volk. Master thesis. TU Darmstadt, 2014.

[Cyr+14] Britt Cyr, Webb Horn, Daniela Miao, and Michael Specter. "Security Analysis of Wearable Fitness Devices (Fitbit)." In: *Massachusets Institute of Technology* (2014).

[Kuo+14] Ye-Sheng Kuo, Pat Pannuto, Ko-Jen Hsiao, and Prabal Dutta. "Luxapose: Indoor Positioning with Mobile Phones and Visible Light." In: *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*. MobiCom. ACM, 2014. DOI: 10.1145/2639108.2639109.

[LM+14] Hoa Le Minh, Anh T Pham, Zabih Ghassemlooy, and Andrew Burton. "Secured communications-zone multiple input multiple output visible light communications." In: *Globecom Workshops (GC Wkshps), 2014*. IEEE. 2014.

[ML14] Ayman Mostafa and Lutz Lampe. "Physical-layer Security for Indoor Visible Light Communications." In: *IEEE International Conference on Communications*. 2014. DOI: 10.1109/ICC.2014.6883837.

[QHK14] Yijun Qiao, Harald Haas, and Edward Knightly. "Demo: A Software-defined Visible Light Communications System with WARP." In: *1st ACM Workshop on Visible Light Communication Systems*. 2014.

[Sch+14] Stefan Schmid, Josef Ziegler, Giorgio Corbellini, Thomas R. Gross, and Stefan Mangold. "Using Consumer LED Light Bulbs for Low-cost Visible Light Communication Systems." In: *Proceedings of the 1st ACM MobiCom Workshop on Visible Light Communication Systems*. Maui, Hawaii, USA: ACM, 2014. ISBN: 978-1-4503-3067-1. DOI: 10.1145/2643164.2643170. URL: http://doi.acm.org/10.1145/2643164.2643170.

[SLH14] Matthias Schulz, Adrian Loch, and Matthias Hollick. "Practical Known-Plaintext Attacks against Physical Layer Security in Wireless MIMO Systems." In: *NDSS*. 2014.

[TC14] Yee-Yong Tan and Wan-Young Chung. *Mobile health–monitoring system through visible light communication*. Bio-medical materials and engineering, 2014.

[Wik14] Wikileaks. *Weeping Angel—Smart TV Surveillance*. 2014. URL: `https://wikileaks.org/ciav7p1/cms/page_12353643.html`.

[Zha+14] Bingsheng Zhang, Kui Ren, Guoliang Xing, Xinwen Fu, and Cong Wang. "SBVLC: Secure Barcode-based Visible Light Communication for Smartphones." In: *Proceedings IEEE INFOCOM*. 2014. DOI: `10.1109/INFOCOM.2014.6848214`.

[ZP14] Wei Zhou and Selwyn Piramuthu. "Security/privacy of wearable fitness tracking IoT devices." In: *Proceedings of the 9th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE. Barcelona, Spain, 2014.

[Fre13] Louis E Frenzel. "Millimeter Waves Will Expand the Wireless Future." In: *Electronic Design* 04/2013 (2013).

[Lom13] Natasha Lomas. *Tile Grabs $2.6M Via Selfstarter For Its Lost Property-Finding Bluetooth Tags Plus App*. Verizon Media, 2013. URL: `https://techcrunch.com/2013/07/24/tile-grabs-2-6m-via-selfstarter-for-its-lost-property-finding-bluetooth-tags-plus-app/`.

[Pas13] Pasternack Enterprises. *Pasternack 60 GHz Transmit/Receive (Tx/Rx) Development System (PEM003-KIT)*. `http://www.pasternack.com/60-ghz-development-systems-category.aspx`. 2013.

[RCB13] Mahmudur Rahman, Bogdan Carbunar, and Madhusudan Banik. "Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device." In: *Proceedings of the 13th Privacy Enhancing Technologies Symposium (PETS)*. Bloomington, Indiana, USA, 2013.

[Tip+13] Nils Ole Tippenhauer, Luka Malisa, Aanjhan Ranganathan, and Srdjan Capkun. "On limitations of friendly jamming for confidentiality." In: *Symposium on Security and Privacy (SP)*. IEEE. 2013.

[ALK12] Narendra Anand, Sung-Ju Lee, and Edward Knightly. "Strobe: Actively securing wireless communications using zero-forcing beamforming." In: *INFOCOM*. IEEE. 2012.

[RRL12] Sridhar Rajagopal, Richard D Roberts, and Sang-Kyu Lim. "IEEE 802.15.7 visible light communication: modulation schemes and dimming support." In: *IEEE Communications Magazine* 50.3 (2012).

[The12] The Institute of Electrical and Electronic Engineers, Inc. "IEEE standard 802.11-2012." English. In: *IEEE Standard for Information Technology* (2012).

[LSK11] Cen B. Liu, Bahareh Sadeghi, and Edward W. Knightly. "Enabling Vehicular Visible Light Communication Networks." In: *Proceedings of the Eighth ACM International Workshop on Vehicular Inter-networking*. VANET. Las Vegas, Nevada, USA: ACM, 2011. ISBN: 978-1-4503-0869-4. DOI: `10.1145/2030698.2030705`. URL: `http://doi.acm.org/10.1145/2030698.2030705`.

[Oku+11] K. Okuda, M. Murata, T. Nakamura, W. Uemura, and T. Yamamoto. "Proposal and development of encryption key distribution system using visible light communication." In: *IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)*. 2011.

[Oss11] Ossmann, Michael and Spill, Dominic. *Project Ubertooth: Open Source Wireless Development Platform Suitable for Bluetooth Experimentation*. 2011. URL: `https://github.com/greatscottgadgets/ubertooth`.

[Pov11] Gordon Povey. *Top 10 Visible Light Communications Applications*. 2011. URL: `http://visiblelightcomm.com/top-10-visible-light-communications-applications/`.

[The11] The Institute of Electrical and Electronic Engineers, Inc. *IEEE standard for local and metropolitan area networks–part 15.7: Short-range wireless optical communication using visible light.* Std 802.15.7, 2011.

[Daw+10] Maurice Dawson, Darrell Norman Burrell, Emad Rahim, and Stephen Brewster. "Integrating Software Assurance into the Software Development Life Cycle (SDLC)." In: *Journal of Information Systems Technology and Planning* 3.6 (2010).

[Jan+09] Suman Jana, Sriram Nandha Premnath, Mike Clark, Sneha K Kasera, Neal Patwari, and Srikanth V Krishnamurthy. "On the effectiveness of secret key extraction from wireless signal strength in real environments." In: *Proceedings of the 15th annual international conference on Mobile computing and networking*. ACM. 2009.

[Mic09] Michael Ossmann and Dominic Spill and Mark Steward. *Bluetooth, Smells Like Chicken*. DEFCON 17. 2009. URL: `https://www.youtube.com/watch?v=9WAmMwUyzMc`.

[Tip+08] Nils Ole Tippenhauer, Kasper Bonne Rasmussen, C Popper, and Srdjan Capkun. "iPhone and iPod location spoofing: Attacks on public WLAN-based positioning systems." In: *System Security Group, ETH Zürich Univ., Zürich, Switzerland, Tech. Rep* 599 (2008).

[CS07] Richard Chang and Vitaly Shmatikov. "Formal Analysis of Authentication in Bluetooth Device Pairing." In: *FCS-ARSPA07* (2007).

[HH07] Konstantin Hypponen and Keijo MJ Haataja. ""Nino" man-in-the-middle attack on bluetooth secure simple pairing." In: *3rd IEEE/IFIP International Conference in Central Asia on Internet*. IEEE. 2007.

[SB07] Dominic Spill and Andrea Bittau. "BlueSniff: Eve Meets Alice and Bluetooth." In: *WOOT* 7 (2007).

[HK05] Gerhard P Hancke and Markus G Kuhn. "An RFID distance bounding protocol." In: *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM)*. IEEE. 2005.

[Sha05] Shaked, Yaniv and Wool, Avishai. "Cracking the Bluetooth PIN." In: *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*. ACM. 2005.

[Info3] Infineon. *Infineon reports Second Quarter and First Half Year Results for Fiscal Year 2003*. 2003. URL: `https://www.infineon.com/cms/en/about-infineon/press/press-releases/2003/95690.html`.

[JN00] Bahram Javidi and Takanori Nomura. "Polarization encoding for optical security systems." In: *Optical Engineering* 39.9 (2000).

[Cou93] Council of the European Union. *Council Directive 93/42/EEC of 14 June 1993 concerning medical devices*. 1993.

[STS91] Bahaa EA Saleh, Malvin Carl Teich, and Bahaa E Saleh. *Fundamentals of photonics*. Vol. 22. Wiley New York, 1991.

[BS87] P. Beckmann and A. Spizzichino. *The scattering of electromagnetic waves from rough surfaces*. 1987.

[Wyn75] A. D. Wyner. "The Wire-tap Channel." In: *Bell Systems Technical Journal* 54.8 (1975).

[Ver19] Gilbert Sandford Vernam. *Secret signaling system*. US Patent 1,310,719. 1919.

ERKLÄRUNG ZUR DISSERTATIONSSCHRIFT

gemäß § 9 der Allgemeinen Bestimmungen der Promotionsordnung der Technische Universität Darmstadt vom 12. Januar 1990 (ABI. 1990, S. 658) in der Fassung der 8. Novelle vom 1. März 2018

Hiermit versichere ich, Jiska Dorothee Classen, die vorliegende Dissertationsschrift ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Eigenzitate aus vorausgehenden wissenschaftlichen Veröffentlichungen werden in Anlehnung an die Hinweise des Promotionsausschusses Fachbereich Informatik zum Thema "Eigenzitate in wissenschaftlichen Arbeiten" (EZ-2014/10) in Kapitel "*Previously Published Material*" auf Seiten vii bis xiii gelistet. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. In der abgegebenen Dissertationsschrift stimmen die schriftliche und die elektronische Fassung überein.

Darmstadt, 6. Dezember 2019

Jiska Dorothee Classen