Cross-protocol attacks: weaponizing a smartphone by diverting its Bluetooth controller

Romain Cayre, Géraldine Marconato, Florent Galtier, Mohamed Kaâniche, Vincent Nicomette, Guillaume Auriol

▶ **To cite this version:**

Romain Cayre, Géraldine Marconato, Florent Galtier, Mohamed Kaâniche, Vincent Nicomette, et al.. Cross-protocol attacks: weaponizing a smartphone by diverting its Bluetooth controller. 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Jun 2021, Abu Dhabi, United Arab Emirates. 10.1145/3448300.3468258 . hal-03355664

HAL Id: hal-03355664
https://laas.hal.science/hal-03355664

Submitted on 27 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

POSTER: Cross-protocol attacks: weaponizing a smartphone by diverting its Bluetooth controller

Romain Cayre
Géraldine Marconato
APSYS.Lab, APSYS, Toulouse, France
firstname.lastname@airbus.com

Florent Galtier
Mohamed Kaâniche
CNRS, LAAS, Toulouse, France
firstname.lastname@laas.fr

Vincent Nicomette
Guillaume Auriol
Univ de Toulouse, INSA, LAAS
Toulouse, France
firstname.lastname@laas.fr

ABSTRACT

In this paper, we focus on a new type of wireless attacks, named cross-technology pivoting attacks. The main objective of these attacks is to divert the transceivers of compromised devices dedicated to a given protocol to allow them to communicate through another protocol, taking advantage of some similarities in their modulation schemes. The main contribution of this work consists in demonstrating the practical feasibility of pivoting attacks from off-the-shelf devices implementing the Bluetooth 5.0 specification. To our knowledge, this attack has not been explored so far in the state of the art.

ACM Reference Format:
Romain Cayre, Géraldine Marconato, Florent Galtier, Mohamed Kaâniche, Vincent Nicomette, and Guillaume Auriol. 2021. POSTER: Cross-protocol attacks: weaponizing a smartphone by diverting its Bluetooth controller. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The rapid and massive expansion of connected objects in our daily life raises new security concerns. Indeed, most of these devices use heterogeneous and vulnerable decentralised wireless communication protocols such as Bluetooth or Zigbee, and some of them are commonly used in mobile application scenarios. This situation allows attackers to explore new offensive scenarios, taking advantage of the massive and pervasive dissemination of these wireless technologies to compromise vulnerable devices and networks. Among those, cross-technology pivoting attacks allow a compromised device to be used as an intermediary to attack other kind of devices. Given that many IoT devices are intended for mobile use, the existence of this new type of attack significantly increases the attack surfaces of the targeted networks. As an example, an employee's smartwatch may be compromised in a public space using a Bluetooth Low Energy (BLE) exploit, and then used as an intermediary to compromise its company Zigbee network when the employee is in range.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this work, we explore the feasibility of diverting a *BCM4375* Bluetooth controller embedded in a Samsung Galaxy S20 smartphone to implement this new kind of pivoting attacks, resulting in a wide attack surface. Second we describe how we successfully implemented such an attack to target three different wireless technologies: Zigbee, Enhanced ShockBurst and Mosart.

2 FIRMWARE REVERSE ENGINEERING

2.1 InternalBlue framework

The *InternalBlue* framework[4] takes advantage of some vendor-specific commands and allows to easily dump, analyse and patch firmware embedded in Bluetooth controllers from Broadcom and Cypress, which are common in the wild. First, it allowed us to dynamically instrument the firmware to understand its internals. Second, we used it to patch some specific functions to integrate our customized receive and send primitives and to add support for new protocols. Note that the use of this framework requires root access on the smartphone as it needs to send arbitrary commands to the Bluetooth controller using the Host-Controller Interface. In the specific case of Samsung Galaxy S20, we also had to replace one of the patched official Broadcom file by an older one, as some of these patches removed support of some specific commands used by *InternalBlue*, as mentioned in the framework documentation[3].

2.2 Methodology

We focused our analysis on a recent bluetooth controller, the *BCM4375* chip from *Broadcom*. This chip is embedded in many smartphones, such as Samsung Galaxy S10 or S20. We have chosen this specific chip for the following reasons:

- it supports Bluetooth 5 and especially the LE 2M physical layer, which is needed to implement Zigbee and Enhanced ShockBurst support,
- it is compatible with *InternalBlue*, which considerably facilitates the process of firmware reverse engineering and patching.

We also analysed the firmware of the CYW20735 IoT develoment board. Indeed, the symbols associated to this firmware are already known, allowing to easily identify the main functions and to understand their behaviour.

We have partially reverse engineered these firmwares, especially the functions linked to the RF hardware configuration and to BLE scanning and advertising tasks. This process was conducted using both static and dynamic analysis with *IDA Pro* and *InternalBlue*. We also identified several common functions that are present in both firmwares: this allowed us to take advantage of the known symbols

of the CYW20735 firmware to infer relevant information about the BCM4375 firmware.

2.3 Diverting scanning and advertising tasks

We have focused our work on the Bluetooth Low Energy stack, and more particularly on the features related to advertisements, such as scanning or advertising. Indeed, these features do not require the establishment of a connection as a prerequisite to send and receive packets. Therefore, they are good candidates to be diverted, in order to implement receive and send primitives for other wireless protocols. They are implemented in the firmware as tasks, consisting of several functions and callbacks.

First, we identified the main functions linked to the configuration of the RF hardware, the reception callback (`extendedScanTaskRxDone`) used by the scanning task and the transmission function (`extendedAdvTaskProgHw`) used by the advertising task. Second, we modified some specific instructions in these functions to redirect the execution flow to our own code stored in RAM, allowing us to 1) configure the RF hardware; 2) gain direct access to the raw demodulator output thanks to a memory mapped register and 3) gain indirect access to the modulator input by storing our complete packet into an advertisement packet payload, using Packet-in-Packet attack [2].

2.4 RF hardware configuration

In order to implement our reception and transmission primitives to support other wireless protocols, we must be able to perform the following operations:

- choose an arbitrary preamble,
- choose an arbitrary frequency,
- select a 2Mbps data rate,
- receive data from the demodulator output,
- send data to the modulator input.

We mainly identified the configuration function linked to the setup of *LE 2M* physical layer (`le2m_program2MAdvMode`). The BLE *access address* being used as a pattern to match the beginning of a BLE packet, we used it to select an arbitrary preamble to synchronize with packets from other wireless protocols using a 2Mbps data rate. The whitening operation was configured using a specific function (`bcsulp_setupWhitenning`) which has been modified to disable this feature, allowing us to manipulate the demodulator output and the modulator input without requiring additional data processing. Two different registers are used to select the frequency, one being used by the scanning task and the other one by the advertising task. However, both of them allow to select an arbitrary frequency in the 2.4 to 2.5 GHz band by providing an offset from 2402 MHz, specified in MHz (as an example, selecting 2410 MHz implies to write a 8MHz offset to one of these registers).

We were able to implement both a reception and a transmission primitive by diverting these features, allowing us to handle arbitrary GFSK packets using a 2Mbps data rate. We then used these primitives to add support for several non-native protocols, such as Zigbee, Mosart and Enhanced ShockBurst.

2.5 Host/Controller communication

We introduced these new offensive capabilities directly in the Bluetooth Controller by patching its firmware with *InternalBlue*. However, they have to be handled from the smartphone, also known as Host. Therefore, we had to find a way to establish a communication between the Controller and the Host to build a relevant API.

The Bluetooth specification describes an interface named *Host Controller Interface* (HCI), allowing Host to Controller communication using commands and Controller to Host communication using events. We identified two functions allowing to allocate a buffer describing an event message (`bthci_event_AllocateEventAndFillHeader`) and send it to the Host (`bthci_event_AttemptToEnqueueEventToTransport`) : we mostly used them to send the received packets to the smartphone. We also discovered that HCI commands are handled using an array of function pointers: the command identifier is used to calculate an index, allowing to call the corresponding function into the firmware. We found two unused command identifiers and stored our own functions' addresses at the right places in this array, allowing us to expose a simple API that can be used to control the receiver mode and transmit a given packet.

These modifications allowed us to implement a user-friendly API, which can then be easily manipulated from the smartphone using the HCI. We are currently working on an experimental Android application allowing to interact with the controller to trigger the new offensive capabilities we added: we monitor HCI events by parsing in real time the Bluetooth HCI snoop log and we can also send commands to the controller by writing the raw command message directly to /dev/ttySAC1. We plan to release both the patches and the application as open source software.

3 PROTOCOLS SUPPORT

3.1 Zigbee

We implemented Zigbee protocol support using the WazaBee attack[1]. This attack describes how to divert a Bluetooth Low Energy chip to implement reception and emission primitives for 802.15.4-based protocols by taking advantage of similarities between Gaussian Frequency Shift Keying (GFSK) and Offset Quadrature Phase Shift Keying (O-QPSK) modulation schemes. We added a correspondence table in the firmware, allowing to map each Zigbee symbol to the corresponding GFSK demodulated binary sequence. We also added helper functions allowing to automatically perform this conversion when a Zigbee packet is received or sent. Every Zigbee packet starts with a 4 bytes-long preamble which is composed of zeros: as a consequence, we generated the GFSK bytes sequence corresponding to the zero symbol and used it as preamble to synchronise the receiver with Zigbee frames. Selecting a specific Zigbee channel is straightforward, as the offset we have to write in the frequency selection register depends directly on the Zigbee channel number.

We performed experiments to evaluate our primitives. We built a custom Zigbee network composed of multiple XBee nodes: the coordinator receives packets transmitted every second by end devices, extracts a numeric value from the payload and displays it on a graph. This setup simulates a sensors network, with the coordinator acting as a visualizer and the end devices acting as sensors. We first used our primitives to passively monitor traffic to identify the network PanID and nodes' addresses. Then, we injected a fake configuration

to perform a denial of service attack targeting a specific sensor and spoofed it by transmitting fake data to the visualizer.

3.2 Mosart

Mosart is a proprietary protocol commonly used by wireless mice and keyboards from various manufacturers. It is based on a GFSK modulation scheme using a 1 Mbps datarate. A Mosart packet consists of a 2-byte preamble (0x5555), a 4-byte address, a variable length payload and a 2-byte CRC. One of the major issues we encountered in implementing this protocol is related to the RF hardware of BCM4375 chip: even though BLE natively supports a physical layer using 1 Mbps data rate, the firmware does not expose any function to select an arbitrary access address if 1 Mbps data rate is used. We assume that the access address used in LE 1M advertising mode is hard-coded in the RF hardware and cannot be easily changed from the firmware, which complicates the implementation of the reception primitive. However, as we mentioned in subsection 2.4, the access address can be chosen arbitrarily if the *LE 2M* physical layer is used. We solved this problem by using *LE 2M* and duplicating every bit: as an example, the 2-byte preamble 0x5555 at 1 Mbps becomes 0x33333333 with 2 Mbps data rate. We have implemented helper functions to select one bit over two in the demodulator output to decode a received Mosart packet and to duplicate each bit of the sent packets before their transmission to the modulator input. It should be noted that other simple transformations are also performed in these functions, allowing to deal with scrambling and endianness.

We evaluated these two primitives by implementing several attacks from *MouseJack*[5], a set of vulnerabilities targeting wireless keyboards and mice. Indeed, the Mosart protocol does not use encryption, so we were able to implement a wireless keylogger allowing to passively collect keystrokes and inject arbitrary keystrokes or mouse events to a vulnerable Mosart dongle.

3.3 Enhanced ShockBurst

Enhanced ShockBurst is a proprietary protocol using a GFSK modulation at 2Mbps, used by many keyboard or mouse protocols, such as Logitech Unifying.

Each packet starts with a preamble of 0xAA or 0x55, followed by a 5-byte address, a payload and a CRC. Since the modulation scheme it uses is identical to the one used in BLE, it is quite easy to implement the primitives described above to communicate using this protocol. Some minor differences, such as the endianness, can be easily solved with a simple transformation applied to the modulator input and the demodulator output. Synchronizing the receiver with Enhanced ShockBurst packets is straightforward if the address is known, as we can use its first bytes as preamble. Without prior knowledge of this address, we first configure our receiver with an arbitrary preamble to get large demodulated buffers, in which we then search for valid packets to extract the embedded addresses.

Using this approach, we could get the address of a Logitech wireless mouse, and then sniff its communications or inject malicious packets to trigger mouse clicks or arbitrary movements. Let us note that M. Newlin identified multiple critical vulnerabilities in *MouseJack*[5] that could be triggered using these primitives, allowing a fake device to be paired with a dongle without user interaction or to inject unencrypted keystrokes. Most of these issues were supposed to be fixed, but during our experiments, we encountered recent devices which are still vulnerable to some of them.

4 EXTENSION

We believe that this work could be extended to many other wireless protocols. As an example, *ANT+* protocol, a proprietary protocol commonly used by sports-oriented devices, has been reverse engineered by T. Szakaly[6] and seems to be based on a variant of *Enhanced ShockBurst* protocol: therefore, implementing support for the *ANT+* protocol should be straightforward. Similarly, in our previous research we noted that many proprietary protocols used by drones and wireless keyboards and mice seem to be based on similar modulation schemes, making them good candidates to be implemented using our primitives.

This work has been focused on implementing new offensive capabilities on a BCM4375 chip. However, the *InternalBlue* framework can be used to inteoperate with several other chips from Broadcom and Cypress, and we noted in our experiments, that the firmwares generally share a significant amount of code. Therefore, it would probably be straightforward to implement this work on other chips from the same manufacturers. Note that since we are relying primarily on *LE 2M* physical layer, the candidate chips must provide a minimal Bluetooth 5 support.

5 CONCLUSION

In this article, we presented an approach allowing to divert a Bluetooth chip embedded in a smartphone to communicate over different wireless protocols, demonstrating the feasibility of cross-protocol attack strategies on a standard mobile phone. This is critical from a security perspective, as this attack strategy does not require any expensive or specific equipment, takes advantage of the ubiquity of Bluetooth devices and is mobile. For example, compromising an employee's smartphone could lead an attacker to pivot on different other protocols used by a company to carry out passive or active attacks, such as injecting keystrokes on a distant computer, or inserting a malicious node in a ZigBee network. As the impact of such attack scenarios may be critical, we therefore believe we should rethink existing defense strategies in wireless environments, to take possible stealthy pivoting attacks into account.

REFERENCES

[1] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, and Géraldine Marconato. 2021. WazaBee: attacking Zigbee networks by diverting Bluetooth Low Energy chips. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Taipei (virtual), Taiwan. https://hal.laas.fr/hal-03193299

[2] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro, and Ryan Speers. 2011. Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios.. In *WOOT*. 54–61.

[3] SEEMOO Lab. 2018. InternalBlue repository. https://github.com/seemoo-lab/internalblue/.

[4] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. 2019. InternalBlue - Bluetooth Binary Patching and Experimentation Framework. *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services* (Jun 2019). https://doi.org/10.1145/3307334.3326089

[5] Marc Newlin. 2016. MouseJack : White Paper. https://github.com/BastilleResearch/mousejack/blob/master/doc/pdf/DEFCON-24-Marc-Newlin-MouseJack-Injecting-Keystrokes-Into-Wireless-Mice.whitepaper.pdf.

[6] Tamas Szakaly. 2016. Help, I've got ANTs !!! https://media.defcon.org/DEFCON24/DEFCON24presentations/DEFCON24-Tamas-Szakaly-Help-I-got-ANTS.pdf.