AirTag of the Clones: Shenanigans with Liberated Item Finders

Thomas Roth
Leveldown Security
airtag@stacksmashing.net

Fabian Freyer
—
mail@fabianfreyer.de

Matthias Hollick
SEEMOO, TU Darmstadt
mhollick@seemoo.de

Jiska Classen
SEEMOO, TU Darmstadt
jclassen@seemoo.de

***Abstract*—AirTags are the first standalone devices that support Apple's Find My network. Besides being a low-cost item finder, they provide an exciting research platform into Apple's ecosystem security and privacy aspects. Each AirTag device contains a Nordic nRF52832 chip for Bluetooth Low Energy (BLE) and Near Field Communication (NFC) connectivity, as well as Apple's U1 chip for Ultra-wideband (UWB) fine ranging. In this paper, we analyze the AirTag hardware and firmware in detail and present attacks that also affect the whole AirTag ecosystem. After performing a voltage glitching attack on the nRF chip, we extract and reverse engineer the main firmware. We add firmware functionality, change capabilities, and demonstrate cloning AirTags. Moreover, we analyze the protocol used between iPhones and AirTags, unlocking undocumented commands. These commands enable limited firmware instrumentation over-the-air on unmodified AirTag hardware, including playing sound sequences and downgrading the nRF and U1 firmware.**

I. Introduction

Apple's Find My network enables users to locate lost devices while disconnected from the Internet [1]. Initially, this network was only relevant to owners of Apple's mobile devices, such as iPhones, iPads, Watches, or MacBooks. However, almost two years after the Find My release in 2019, Apple added AirTags [2], which are usable as finders for wallets, bikes, and other items. Any nearby Apple device with an Internet connection and precise Global Positioning System (GPS) coordinates reports the item's location, making Find My the offline finding network with the highest sensor density.

AirTags allow research into security and privacy aspects of the overall Apple ecosystem, with a particular emphasis on wireless communication and sensing aspects. As of early 2022, they are the cheapest Apple computing device sold for as little as USD 30. The main points of interest for research are integration into the Find My network [1] and their use of Apple's proprietary UWB chip designated U1 [3]. General concepts of the AirTag integration also apply to other Apple accessories. For example, AirPod firmware updates work similarly to the AirTag's, and AirTag U1 firmware updates are similar to the iPhone's. Thus, despite its focus on AirTags, this paper describes previously undocumented parts of Apple's proprietary ecosystem.

To the best of our knowledge, we are the first to analyze AirTag hardware and firmware in depth. We made some of our results available in advance to the community to enable external research [3], [4], [5]. This paper presents these results in detail and adds new information. Our main contributions are as follows:

- We demonstrate that voltage glitching the nRF chip for firmware extraction is possible with a low-cost setup,
- we reverse-engineer the extracted nRF firmware in detail,
- we modify parts of the nRF firmware for different applications, such as cloning AirTags or changing the NFC contact information,
- we reverse-engineer the wireless protocol between AirTags and iPhones, and
- we demonstrate over-the-air firmware downgrades.

AirTags will remain accessible as a research platform independently of firmware patches since voltage glitching is a hardware vulnerability. Thus, our work forms a basis for future projects. We release our tools for glitching[1] and interacting with AirTags over-the-air[2] to enable follow-up research.

The remainder of this paper is structured as follows. We explain the basic working of Apple's Find My network, AirTag integration, and UWB in Section II. Based on this knowledge, we introduce a threat model in Section III. Then, we explore the AirTag's hardware, glitch the nRF, and modify its firmware in Section IV. Furthermore, we analyze the BLE protocol to send custom commands to AirTags, including firmware downgrades in Section V. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Background

A. Find My Protocol

Find My is Apple's offline finding network, enabling potentially lost devices without an Internet connection to report accurate locations to their owners. Offline devices send BLE advertisements. Once an online device of the Apple ecosystem observes such an advertisement, it reports it along with its own GPS position to Apple's servers. The legitimate owner can query for such reports.

Apple's implementation aims to ensure privacy and anonymity by design [6]. In addition to Apple's claims of these properties, Find My has been reverse-engineered, revealing the underlying cryptographic primitives and implementation details [1]. The amount of advertisements is minimized by only broadcasting while the AirTag is disconnected from the

[1] https://github.com/stacksmashing/airtag-glitcher
[2] https://github.com/seemoo-lab/airtag

owner's iPhone. Advertisements contain a rotating public key, preventing long-term linkage to the owner. Public keys rotate in a predefined sequence, derived from a device-specific master beacon key only known to the owner. A reporting device encrypts its current location with the public key but does not include its identity. Therefore, Apple cannot observe reported locations, only the legitimate owner can search and decrypt reports, and the reporting device does not reveal its identity to the owner. This protocol design is different from other proprietary offline finding networks, where the server knows all locations and can reveal locations of all tracked devices to attackers through a vulnerability or coercion [7], [8].

B. Find My Hardware and Ecosystem Requirements

Standalone trackers are based on GPS and require a cellular connection for location reports. Thus, the underlying hardware is complex and expensive, and its battery life is limited. In contrast, BLE devices powered by a coin-cell type battery may run up to a year. These advantages motivated various other vendors to implement similar BLE-based offline finding networks much prior to Apple [7]. Previous solutions require all participating users to install and continuously run an app, and often, devices cannot be located because no actively reporting device is within BLE range. This is further restricted by limits imposed by the mobile operating system and increased device battery usage. Since Apple enrolls all iPhones to Find My by default, the network of observing and reporting devices is very dense and accurate compared to previous solutions [1].

For vendors lacking Apple's market share and homogenous software landscape implementing a location reporting system with similar capabilities to Find My is very difficult. Apple collaborates with third-party Bluetooth device manufacturers to integrate further certified items into the Find My network, such as e-bikes, headphones, and item finders [9]. Based on the reverse-engineering efforts on Find My, the open-source project *OpenHaystack* enables everyone to build third-party item finders [10]. This project runs on low-cost BLE hardware, such as the nRF51, ESP32, and Raspberry Pi.

C. AirTag Hardware Components

The main chips on the AirTag's Printed Circuit Board (PCB) are depicted in Figure 1. Understanding their functionality is important to follow the remainder of this paper, including security assumptions and reverse engineering tasks.

The nRF52832 chip runs the main firmware application. It supports wireless connectivity with BLE and NFC. BLE is needed for the Find My network and communication with the owner's iPhone, while NFC holds a link to contact the AirTag's owner. The nRF has an internal flash memory with readout protection, called Access Port Protection (APPROTECT) [11]. This protects the configuration information, such as each AirTag's unique serial number and common keys used to verify over-the-air updates. The firmware in the internal flash is not secret and is part of public update files.

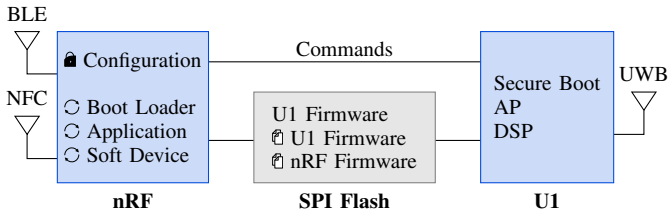


Fig. 1: AirTag main component and storage overview.

A separate chip, the U1, handles UWB. It supports fine ranging on the iPhone 11 and higher, showing the precise distance and direction of a nearby AirTag. Most of the time, U1 is inactive. It regularly wakes up for key management and is enabled during ranging. Compared to U1 chips in the iPhone, the AirTag U1 chip comes in a smaller package with a minimal radio front-end. It only has one transmit and one receive antenna, while the iPhone version has multiple antennas used for direction finding.

The nRF and the U1 chip connect to a separate Serial Peripheral Interface (SPI) flash for storing and reading data. As previously documented, the SPI flash debug pins allow reading its contents [12]. It stores two signed copies of the U1 firmware. Since the U1 bootloader checks the signatures, firmware modified in the SPI flash will not boot. During updates, the nRF firmware is also stored on the SPI flash. However, the nRF boots from internal flash storage.

Furthermore, the nRF connects to various other components, such as an accelerometer and a minimal speaker.

III. Threat Model

Apple did not publish a threat model for AirTags. The AirTag hardware and firmware design and their integration into Apple's ecosystem still allow us to deduce underlying security assumptions. Apple's threat model likely underestimated some attacks, especially stalking, since they improved protections with subsequent updates [13], [14]. In the following, we describe the attacker's capabilities (Section III-A) and potential threats (Section III-B). Even though we are the first to modify the AirTag's firmware through hardware access, there are known vulnerabilities in the Find My ecosystem and less sophisticated hardware attacks (Section III-C).

A. Attacker Capabilities

We consider two types of attackers. An attacker with *physical access* can modify the hardware, e.g., solder wires onto the AirTag's PCB, detach or replace components, or steal an AirTag. Physical access might damage the AirTag's hardware or leave other unwanted traces. Hardware and firmware tampering is often considered out of scope for item trackers [15]. Thus, we also consider an *over-the-air* attacker within physical proximity, only accessing the AirTag via BLE, NFC, or UWB.

We focus on attacks specific to AirTags. The Find My network, including its security, sending valid advertisements, and conditions that violate tracking detection, have been extensively studied [1], [10], [16], [17].

B. Threats and Mitigations

Stalking: The Find My protocol ensures that only an AirTag's legitimate owner can access its location information. Since the AirTag's master beacon key is synchronized via the owner's iCloud account, they can see the AirTag's location from all their devices.

Restricted location data access does not protect against unwanted tracking by an attacker who places an AirTag onto a victim's item without consent. Apple released various firmware updates to improve stalking protections on the physical AirTag. The initial stock firmware only played a sound when the AirTag was away for three days from their owner, and the first update in June 2021 reduced this time to a random time window between $8\,\mathrm{h}$ and $24\,\mathrm{h}$ [14]. This sound plays immediately when the AirTag moves after standing still for a while. In February 2022, Apple announced further increasing the sound level [13].

Stolen Hardware: Once an attacker gains physical access, they can remove an AirTag's battery and steal the item attached to it. Yet, the attacker cannot steal the AirTag. Apple keeps a record of AirTag serial numbers and associated iCloud accounts [13]. During initial pairing, the AirTag connects to an iCloud account. Even after a hard reset [18], the AirTag can only be paired again to the same iCloud account. The iCloud user must remove the AirTag from their account to allow provisioning from another iCloud account, rendering stolen AirTags useless.

Firmware Readout: Apple does not protect firmware against readout, i.e., firmware updates are not encrypted. This is not an AirTag-specific mistake. Most firmware on Apple devices can simply be extracted from update files. There are a few exceptions. However, security researchers repeatedly leaked decryption keys associated with these firmware files. We assume that Apple gave up on "security by obscurity" to protect firmware and only applies basic measures like stripping symbols.

Firmware Modification: Apple protects firmware against modification. Over-the-air firmware updates are signed and verified, and only firmware released by Apple should run on AirTags. On the AirTag, the U1 chip has secure boot [3]. When the firmware on the external SPI flash is modified, the U1 chip no longer boots. The nRF is secured with enabling APPROTECT [11], effectively preventing the firmware from being overwritten by an attacker with physical access. Additionally, the firmware on the nRF verifies over-the-air firmware updates before copying them into the internal flash.

Identity Modification: Each AirTag has a unique serial number printed on the casing below the battery holder. This serial number is exchanged when linking the AirTag to an iCloud account. Apple also uses it for law enforcement, e.g., when a victim discovered an unwanted tracker, Apple can determine the stalker [13]. Serial number modification could tamper with such information. Additionally, stolen, iCloud-locked AirTags could be unlocked by changing the serial number. The nRF flash configuration data holds the AirTag's serial number. Hence, the APPROTECT feature also prevents identity modification.

Identity Extraction: The Find My protocol ensures long-term unlinkability between BLE advertisements. To this end, the master beacon key needs to be protected. An attacker that gains access to the master beacon key of an AirTag can calculate past and future advertisement contents. This leaks an AirTag's identity while the attacker is in over-the-air proximity, and allows access to locations reported to Apple while the AirTag is out of the owner's range. Again, APPROTECT prevents identity extraction.

Outdated Firmware: Firmware updates increase AirTag security, e.g., by improving stalking protections. Updates can also patch Remote Code Execution (RCE) vulnerabilities accessible to an over-the-air attacker. Thus, keeping an AirTag updated is essential.

While the AirTag is connected to its owner's iPhone, the iPhone regularly attempts to update firmware on the AirTag. The AirTag silently updates in the background without indicating this to the owner, and the owner cannot stop this process. Thus, in contrast to other updates within Apple's ecosystem, AirTag firmware updates are enforced without prompting the user.

C. Known Attacks

1) Stalking Protection Bypasses: AirTags were built as item finders. The dense Find My network enables tracking of moving objects, including pets and humans. Thus, they can also stalk people—either by *(i)* observing Find My advertisements and linking them to each other or by *(ii)* attaching a hidden AirTag to their clothes, bicycle, or car.

The Find My protocol protects against *(i)* by rotating the public key in the advertisements. The advertisement rotation time is fixed and $15\,\mathrm{min}$ on iPhones but $24\,\mathrm{h}$ on AirTags. This means that public keys can be linked to each other [17], significantly reducing privacy on AirTags. However, a stalker would still need nearby BLE-enabled devices to observe Find My advertisements on various locations and link them.

Slow public key rotation on AirTags enables a particular form of stalking protection against *(ii)*. If an iPhone user moves and an unknown AirTag travels with them over time, a privacy warning pops up on the user's iPhone, including the route of the AirTag [19]. The third-party software *AirGuard* also enables such warnings for Android users [16]. Shortly after the AirGuard release, Apple also provided an official app for Android users. Interestingly, Apple's tracking detection on iOS and Android misses many situations, e.g., AirTags attached to cars and devices flagged as iPhone [16].

AirTags play a sound when disconnected from the legitimate owner's iPhone upon movement [19]. Tracking alerts were improved with firmware updates [13], [14]. This enables detection if unwillingly attached as in stalking scenario *(ii)*. This additional anti-stalking feature can be bypassed by disconnecting the speaker, and readily modified AirTags were sold [20]. We assume that disconnected speakers are not only used by stalkers. Apple advocates AirTags for finding lost items but not

stolen ones. However, an AirTag with a disconnected speaker also becomes suitable for locating stolen items.

2) Tracking with Other Devices: The Find My protocol has been reverse-engineered in such detail that it is possible to build custom AirTags that integrate into Apple's Find My network [10]. Such implementations could be manipulated to use the shorter 15 min rotation time yet bypass anti-stalking.

Even though AirTags are usable for stalking and Find My implementations could be manipulated, more expensive GPS-based trackers without stalking protections were available and used long before AirTags.

3) Find My for Data Transfer: The nature of encrypted, anonymous reports in Find My enables transferring different data than locations. For example, Find My can transfer arbitrary data from devices that are currently not connected to the Internet [21].

IV. Hardware Analysis

An initial hardware analysis establishes an understanding of the AirTag's components and reveals that the firmware on the nRF can be modified.

A. Fault Injection Setup

nRF microcontrollers have a setting called Access Port Protection (APPROTECT). This setting, stored in the configuration flash of the device, controls the functionality exposed on the Serial Wire Debug (SWD) debug port [11]. When enabled, APPROTECT prevents debuggers from performing memory read operations on the device's flash, RAM, and peripheral memory regions. The nRF allows clearing the APPROTECT setting. However, this causes a chip-erase, purging the entire flash contents. While the firmware can be extracted from over-the-air updates (see Section V-D), none were available when the device launched. Additionally, the nRF flash contains configuration data required to run the device, which prevents running the firmware extracted from over-the-air updates as is. Configuration data access violates most security barriers, as previously outlined in Section III.

Voltage fault injection—or voltage glitching—is a method where the power supply of a chip is changed for a very short period [22]. Depending on the characteristics of the microcontroller this can cause memory and register contents to be corrupted or even allow skipping of entire instructions. The nRF52 family is vulnerable to a voltage-fault injection attack. This bypasses the APPROTECT setting in the boot ROM, enabling an SWD debugger to read out the flash contents. The possibility to glitch application code on the nRF52840 was demonstrated in 2019 by one of the authors of this paper [23]. Later on, other researchers also showed this, including the possibility to bypass APPROTECT, which affects the whole nRF52 family [24], [25]. Their setup is claimed to be homemade and low-cost [24], but the blog post does not provide details. We build a low-cost glitching setup based on the Raspberry Pi Pico, a level shifter, and a MOSFET as shown in Figure 2. Despite rising costs on the electronics market, this setup costs less than 5 € in early 2022.

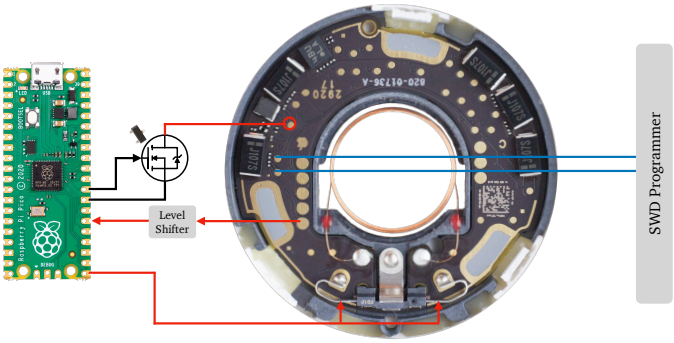


Fig. 2: Low-cost setup for glitching the nRF on the AirTag.

The AirTag power-supply input is directly connected to the Raspberry Pi by soldering a jumper wire to the battery terminals of the AirTag and connecting it to one of the General Purpose Input Outputs (GPIOs) of the Raspberry Pi. The nRF Voltage Common Collector (VCC) is connected to the Raspberry Pi's GPIO ports to detect when exactly in the power-up sequence the nRF is powered on. As the nRF microcontroller operates at 1.8 V, a level-shifter brings this voltage to the 3.3 V levels expected by the Raspberry Pi. To directly affect the nRF CPU core with the fault injection attack, the drain pin of an N-Channel MOSFET is connected to the external bypass capacitor of the nRF core voltage regulator. As shown in Figure 2, this is already exposed as debug pin on the AirTag's PCB, and no modification of existing capacitors is required. The gate of the MOSFET is controlled by one of the Raspberry Pi's GPIOs, while the source is connected to a common ground established between both devices.

When the attack was successful, this shows on the SWD debug port. An SWD programming adapter is connected to the nRF SWD pins. SWD pins are also exposed as test pads on the PCB. Our setup is based on a Segger J-Link, but cheaper SWD programmers are available, e.g., also a separate Raspberry Pi Pico supports SWD via GPIOs [26].

B. Performing the Fault Injection Attack

To perform the attack, firmware for the Raspberry Pi Pico was developed:[3]

1) The firmware receives a delay as well as a glitch pulse length via USB and then provides power to the AirTag.
2) The firmware then waits for the nRF to be turned on by the discrete power-up sequence on the AirTag, which is indicated by the GPIO connected to the level-shifter going to high.
3) Once this occurs, a delay counter starts waiting for the delay received via USB, and then performs the glitch pulse.

[3]The source code is available on https://github.com/stacksmashing/airtag-glitcher, see `AirTag Glitcher.ipynb` for controlling the glitcher's delay and pulse width, and `main.c` for translating glitching commands to pulses on the Pico's GPIOs connected to the AirTag PCB.

4) The computer controlling the setup then attempts to use the SWD debug probe to perform a flash readout. If this succeeds, the attack is successful. Otherwise, the same procedure is restarted.

With this setup, approximately 10 attempts can be performed per second. After an average of approximately 2000 attempts, the attack succeeds, and the firmware of the AirTag can be read out. It is notable that the glitch is still very stable, even though there is a lot of timing variance introduced by the various components. Attacking a new AirTag takes roughly 3.5 min.

C. Reprogramming the nRF

We run different experiments with modified firmware and configuration data, each clarifying different aspects of the integrity and authenticity protections implemented on the AirTag. nRF flash modification is possible after bypassing APPROTECT. It only protects against readout. Even with APPROTECT enabled, an nRF flash can be overwritten with a free erase to unlock procedure. Nonetheless, APPROTECT must be bypassed once with glitching on every AirTag to restore the individual configuration data required to function within Apple's ecosystem.

1) NFC URLs: When an NFC reader scans an AirTag, it sends a URL leading to a site on `found.apple.com`. Using a simple strings analysis of the firmware, this URL can be identified as shown in Figure 3. To determine whether additional firmware integrity protections are implemented on the device, the string is replaced with a link to the YouTube video with the ID *dQw4w9WgXcQ*. The firmware is then flashed back to the device. Afterward, the basic functionality of the AirTag is retained, and it can be located with an iPhone. The modified URL is opened when using an NFC reader, confirming that there are no further nRF firmware protections or checks in place.

```
found.apple.com/airtag?pid=%04x&b=%02x&pt=%04x&fv=%08x&dg
    =%02x&z=%02x
&pi=%s
&bt=%s&sr=%s&bp=%04x
internal error, file radar to (corecrypto | all)
```

Fig. 3: NFC URL contained as string in the AirTag firmware.

2) Cloning AirTags: To determine whether all information exchanged during provisioning of the AirTag is contained in the nRF flash configuration data (and not either in the U1 or the SPI flash), we transfer the flash of one paired AirTag into another AirTag located 800 km away. When turning on the remote AirTag, it appeared at the new position, indicating that all pairing information is contained in the nRF flash. This shows that iCloud provisioning can be bypassed by reflashing a previously dumped firmware and violates the threat model in Section III. Provisioning should prevent re-using a found AirTag without contacting the legitimate owner.

3) Privacy Warning Bypass: Using the data of the previous experiment and comparing the flash contents of different paired AirTags it was possible to identify the unique pairing information. Using this pairing information, we build an AirTag firmware that regularly switches its identity automatically, defeating the system integrated into Apple iOS that alarms the user when an unknown AirTag is detected in proximity for a long time [17]. Note that this is an intended feature of the Find My protocol. For example, iPhones using Find My rotate their public key every 15 min and have a special field marking them as phone instead of a tracker, also hiding them in privacy warnings [16].

4) Custom Sounds: We change the soundset of the AirTag to demonstrate the understanding of the firmware. The AirTag firmware uses a sequencer system, splitting up sound sequences into multiple short sounds. This is visible when loading the firmware as raw 16-bit Pulse-Code Modulation (PCM) audio into a tool such as Audacity, as seen in Figure 4.

Using static analysis, the exact format of the sound sequencer was reverse engineered, and the original sounds were replaced with custom sounds.

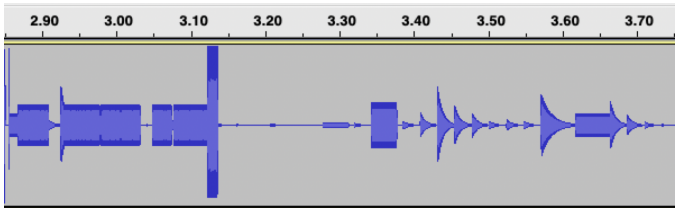


Fig. 4: Sound data area as seen when loaded into Audacity.

5) Accelerometer as Microphone: Accelerometers can be used as microphones [27]. The AirTag PCB contains an unlabeled accelerometer, likely the Bosch 3-axis Mems gyroscope BMA288. Using firmware written from scratch and placing the AirTag in a metal can to improve sound conductance, we attempt using the integrated accelerometer as a microphone. We were not able to achieve enough fidelity to recover audio from the signal reliably, however, other modes of operating the accelerometer might lead to more success.

V. Protocol Analysis

Interacting with AirTags over-the-air unlocks features that are otherwise only accessible after hardware modification. Prior to our work, the AirTag communication protocol was undocumented. In the following, we first document the protocol basics, followed by explaining the analysis approach. Then, we use dynamic instrumentation for sending custom commands and downgrading firmware.

A. BLE Protocol Basics

The Bluetooth specification defines a collection of protocols on all layers [28]. On the physical layer, two different variants exist, with BLE being optimized towards energy consumption and being present in most IoT devices. In contrast to other wireless standards, the Bluetooth specification defines further

protocols up to the application layer, including various options for vendor-specific extensions. The Logical Link Control and Adaptation Protocol (L2CAP) transfers data on top of the physical layer. While it is possible to transfer raw data via L2CAP, GATT is another protocol defined on top and supported by most IoT devices. GATT follows the notion of implementing services, and within a service, a remote device can read and write attributes or subscribe to receive value notifications. Attributes can be public or protected, making them only available to paired devices.

B. Protocol on AirTags

AirTags use various GATT services and a custom protocol on top of raw L2CAP.

1) GATT for Data Transfer: The essential GATT services on AirTags enable data transfer. Attributes are written and acknowledged, following the GATT specification.

After the initial Bluetooth pairing process, AirTags are provisioned to an iCloud account. For this purpose, Apple signs a certificate sent to the AirTag. The provisioning process is based on a GATT service, as shown in Figure 5a. Once an AirTag is provisioned, it is bound to the owner's iCloud account.

Firmware updates also use GATT for data transfer. The firmware update is split into small chunks, and after each write, data reception is acknowledged. A detailed description of the firmware update process follows in Section V-D.

The GATT service additionally exposes a few commands. One of these is the *unauthorized sound* command, which everyone can play on a nearby AirTag that is currently not connected to its owner's iPhone. Lost AirTags and unwanted tracking can be detected with the unauthorized sound.

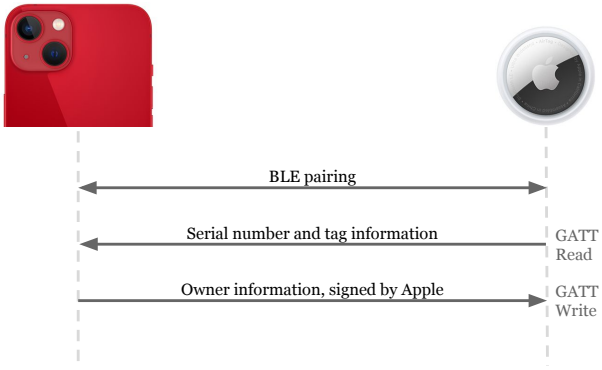
2) Custom L2CAP Commands: The raw L2CAP protocol for AirTags implements commands, as shown in Figure 5b. Each command has a one-byte opcode followed by a payload. If a command was received successfully, it is acknowledged. Some commands require a mutex. Mutexes are also implemented as a command, set before the command requiring a mutex and released afterward.

For example, playing a sound on the AirTag requires the following steps. First, a mutex is set by the iPhone and acknowledged by the AirTag. Then, a sound sequence command is sent to the AirTag, which selects from predefined sound samples within the firmware previously shown in Figure 4. Each sound section consists of 4 bytes: sound index, number of repetitions, offset within sound sample, and pause until next sound. Multiple sounds can be chained within the payload of one command. After transmission, the sound sequence is acknowledged. While the sound is playing, the AirTag regularly sends a status to the iPhone. Finally, the mutex is released.

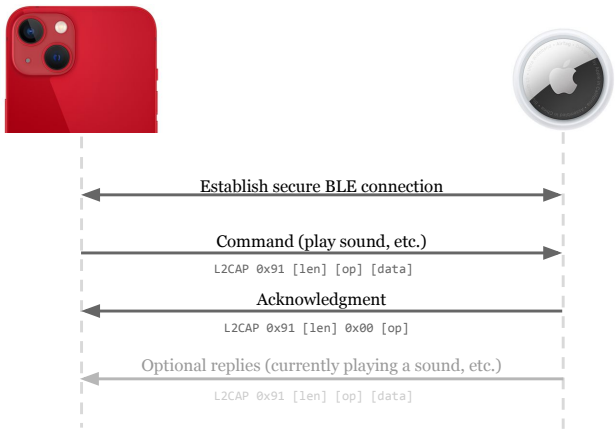
C. Interpreting and Sending Commands

1) Analyzing a Proprietary Bluetooth Protocol on Apple Devices: Generic approaches for BLE protocol analysis are sniffing packets over-the-air or on one of the communicating devices. Over-the-air sniffing requires certain preconditions to break encryption and is supported by `btlejack` [29]. However, `btlejack` cannot break encryption for all pairing modes and often misses packets. Local sniffing on iOS works after installing a Bluetooth debug profile on the iPhone and attaching it to Apple's PacketLogger [30]. PacketLogger has the advantage that it never misses an over-the-air packet. It intercepts packets before encryption and after decryption. We use local sniffing to observe iPhone–AirTag communication, containing all L2CAP and GATT packets. However, sniffing does not reveal the semantics of these packets.

Another approach is analyzing the protocol's implementation within iOS or the AirTag. Static analysis can provide a complete picture of the protocol, even if only a subset of implemented packet types is sent over the air. In contrast, dynamic analysis allows altering and injecting packets within an existing implementation. In the following, we will use



(a) Initial AirTag–iCloud provisioning using a Generic Attribute (GATT) service.



(b) Sending specific commands via Logical Link Control and Adaptation Protocol (L2CAP).

Fig. 5: AirTag BLE protocol implementation.

static and dynamic analysis on a jailbroken iPhone and static analysis of the AirTag firmware.

2) Analyzing and Instrumenting AirTags on iOS: iOS implements the AirTag client within the location services daemon, as shown in Figure 6. The location daemon registers a peripheral handler with the Bluetooth daemon. Afterward, all packets belonging to an AirTag are forwarded to the location daemon for processing. The AirTag is internally called *Durian*, and as part of the core location framework, it is implemented within the Objective-C classes starting with `CLDurian`. Objective-C includes class and method names even within binaries because it is interpreted at runtime. Hence, static analysis tools can reconstruct these, leading to human-readable pseudo-code.

Static analysis can recover all supported command opcodes, including their meaning. The Objective-C method `-[CLDurianTask opcodeDescription]` shown in Listing 1 returns opcode names for a command executed within a task. Since there is no decoder for command payloads, their meaning remains unknown when only analyzing iOS. Instead of directly sending commands, the `CLDurian` implementation wraps their execution into tasks. Each task executes a command. Many task implementations have predefined payloads. For example, the task for playing a sound defines a fixed byte sequence. Static analysis reveals valid payload options for many commands.

FRIDA has an Objective-C API, allowing dynamic instrumentation [31]. Hooks set to Objective-C method names work irrespective of the binary version, as long as it implements the hooked methods. This allows writing hooks for AirTags that work across multiple iOS versions since their introduction in iOS 14.5[4]. We can execute existing tasks using dynamic instrumentation with FRIDA and create our own tasks with custom commands and payloads. We can request status information, play custom sound sequences, and more. Only setting hooks within iOS means that specific parts of the protocol are instrumented. Pairing, provisioning, and establishing connections are implemented within the location daemon and kept intact.

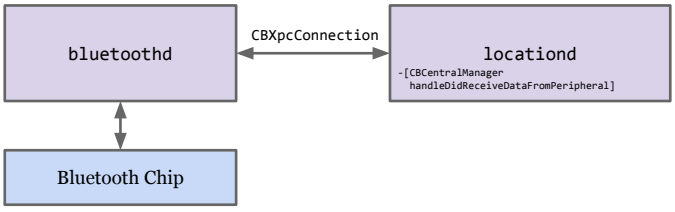


Fig. 6: AirTag implementation as Bluetooth client within the location services daemon `locationd`.

```
// Named function due to Objective-C
-[CLDurianTask opcodeDescription](CLDurianTask *self) {
  int opcode = -[CLDurianCommand opcode](self->_command, "
       opcode");
  return DurianOpcodeDescription(opcode);
}

// Unnamed function
CFString* DurianOpcodeDescription(int opcode) {
  switch (opcode) {
    case 0:
      return CFString("Acknowledge");
    case 1:
      return CFString("Rose Init");  // U1 Ranging
    // ... further commands ...
    case 220:
      return CFString("Stop Unauthorized Sound");
  }
}
```

Listing 1: AirTag opcode implementation within `locationd`.

3) Matching Handlers in the nRF Firmware: The nRF SoftDevice performs BLE signal processing. The SoftDevice is a binary firmware blob supplied by Nordic and not meant to be modified. The nRF SDK provides multiple managers for BLE shown in Figure 7, which are compiled into the firmware and call into the SoftDevice via a Supervisor Call (SVC) and handle interrupt-based events. For example, there is a GATT and L2CAP manager, a security manager for pairings and keys, and a peer manager for active connections.

Some protocol details remain unknown because they are not part of the iOS implementation. For example, the sound sequence is fixed within an iOS task and interpreted by the AirTag. After analyzing the implementation within the AirTag, we know the meaning of the sound sequence payload, previously explained in Section V-B2. By cross-referencing the SVC instructions in the AirTag firmware binary with the nRF SDK, most nRF SDK functions can be identified. This includes functions for registering GATT characteristics and event handlers, called *observers* in the SDK. Wireless communication protocol handlers are identified with this approach. Cross-referencing the SDK is necessary since the firmware contains no symbols and only very few strings.

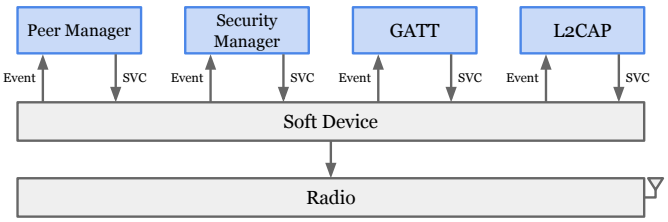


Fig. 7: Relevant Bluetooth layers and their implementation on nRF devices.

4) Reconstructing the nRF Firmware State Machine: Using static analysis, the protocol state machine of the L2CAP handlers was recovered. This was achieved by manual tracing of the data flow from nRF SDK functions to dispatch handlers operating on tables of the state machine (Listing 2 and Listing 3). These tables were then parsed, and graphical representations generated using GraphViz (Figure 8).

[4]We tested our hooks on iOS 14.5–14.8, but once jailbreaks are available, the hooks should also work on newer iOS versions.

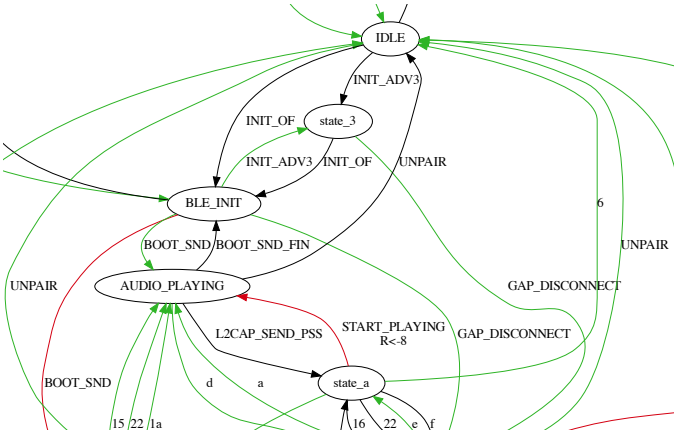


Fig. 8: Excerpt of the recovered state machine showing states relating to sound playback. Green lines represent successful state transitions, red lines failures.

```
struct state_machine_state state_machine_dispatch_table[0xe
    ] = {
  [0x0] = {
    struct state_machine_handlers* handlers = init_state
    uint32_t count = 0x3
  }
  ...
  [0x8] = {
    struct state_machine_handlers* handlers = audio_playing
    uint32_t count = 0x9
  }
  [0x9] = ...
  ...
```

Listing 2: State machine dispatch table containing pointers to the transition tables for each state.

```
struct state_machine_handlers audio_playing[0x9] =
{
  ...
  [0x6] = {
    uint8_t cmd = 0x23
    enum statemachine_state next_state_on_success = STATEA
    enum statemachine_state next_state_on_failure = STATEA
    void (* handler)(int32_t cmd, char data) =
        reply_play_sound_sequence
  }
  [0x7] = {
    uint8_t cmd = 0x24
    enum statemachine_state next_state_on_success = STATE2
    enum statemachine_state next_state_on_failure = STATE2
    void (* handler)(int32_t cmd, char data) =
        audio_playing_boot_snd_fin
  }
  [0x8] = {
    uint8_t cmd = 0x3
    enum statemachine_state next_state_on_success =
        SAME_STATESTATE0
    enum statemachine_state next_state_on_failure =
        SAME_STATESTATE0
    void (* handler)(int32_t cmd, char data) =
        schedule_gap_disconnect_and_start_adv3
  }
  ...
```

Listing 3: State machine transition table for the audio playing state. For each possible transition, the next states and corresponding handlers are given.

D. Firmware Update and Downgrade

Apple accessories paired with an iPhone are updated via iOS. For example, iOS and iPadOS can update AirPods, Apple Pencil, Smart Keyboard, and more. We analyze AirTag updates, but other accessory updates follow the same scheme. Figure 9 shows an overview of the update process.

1) Involved Daemons: Firmware updates require the interplay of multiple daemons with specific tasks. Daemons exchange information using Cross-Process Communication (XPC). Permissions to open mach ports for exchanging XPC messages between daemons are managed by the central launch daemon [32]. If a required daemon is not yet running, the launch daemon invokes it.

The firmware update daemon `fud` is the central component for updating accessories. It regularly scans for updates. Certain events can trigger the `fud` to look for updates. In the case of AirTags, the `searchpartyd` keeps track of nearby devices by observing advertisements. It schedules an update check to `fud` directly 5 min after pairing an AirTag, and repeating every 2.5 h. When the timer expires, `fud` wakes up and starts the AirTag-specific plugin `DurianUpdaterService`. This plugin requires collaboration with various other daemons to run the update. It queries `searchpartyd` if any outdated AirTag is nearby, uses `mobileassetd` for downloading and unpacking the firmware image, queries `fud` for generating valid firmware signatures, and interacts with `locationd` to send Bluetooth data to the AirTag.

2) Firmware Download and Format: Each accessory has a predefined firmware update URL[5]. The XML contents of the firmware information file point to the most recent download link. The update is packed as `.zip`, containing further files. Old download links are still available [33], which allows comparing firmware versions. The initial firmware shipped on the first AirTags generation was never released as online update. As of February 2022, almost one year after the initial AirTag release, only two firmware updates were published.

Update metadata is stored in `BuildManifest.plist`, including a deployment limit and checksums of the U1 firmware. Deployment limits are used for testing purposes. For example, only 1 % of all AirTags receive an update initially. Subsequent firmware updates with higher deployment limits contain the same firmware binary but the version number increases.

The main firmware is stored in `DurianFirmware MobileAsset.bin` as *Super Binary*, an Apple-proprietary format shown in Figure 10. It contains update section names and their offsets within the file. An AirTag update consists of two separate updates for the nRF and the U1 chip. For the nRF, the boot loader, application, and SoftDevice, as well as their signatures are contained separately. Identification of the GATT handlers for the characteristic used for updating (6001, 6003) and static reverse engineering confirms this. On successful

[5]For AirTags, this URL is http://mesu.apple.com/assets/com_apple_MobileAsset_MobileAccessoryUpdate_DurianFirmware/com_apple_MobileAsset_MobileAccessoryUpdate_DurianFirmware.xml.

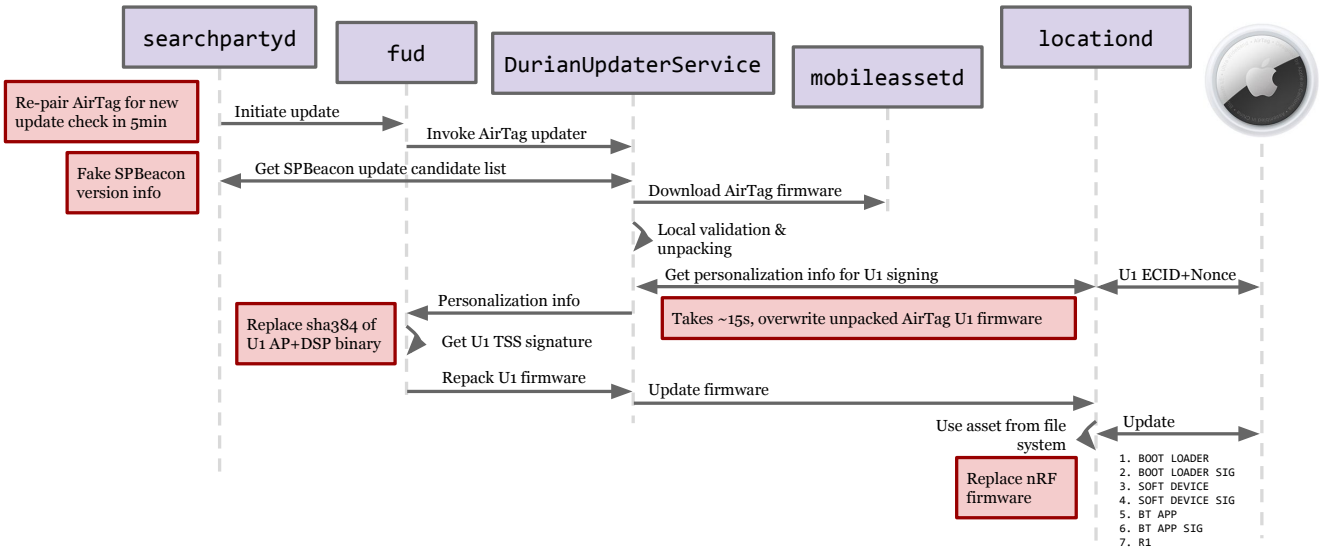


Fig. 9: Firmware update process, which can be manipulated to downgrade firmware. Downgrades to any previously released nRF and any currently signed U1 firmware version are possible, including combinations.

transfer of the update, firmware signatures are checked using a P-256 elliptic curve public key using code from Apple's CoreCrypto framework.

The U1 firmware is included as a single entry in `ftab` format [3], which contains the main U1 application processor firmware and the digital signal processor firmware.

3) Firmware Downgrade: We instrument the firmware update process for firmware downgrades. Requirements are as follows:

- The AirTag is paired to the attacker (e.g., jailbroken iPhone),
- the nRF firmware version is a valid release by Apple but can be outdated, and
- the U1 firmware version is still signed by Apple.

We will utilize a jailbroken iPhone for the firmware downgrade in the following. Regardless, an attacker could reimplement the full AirTag protocol from scratch instead of instrumenting the existing implementation. In this case, the jailbreak is no hard requirement. Nonetheless, firmware downgrades still require that the AirTag is paired with the attacker. The pairing key is protected by iOS and is likely extractable after

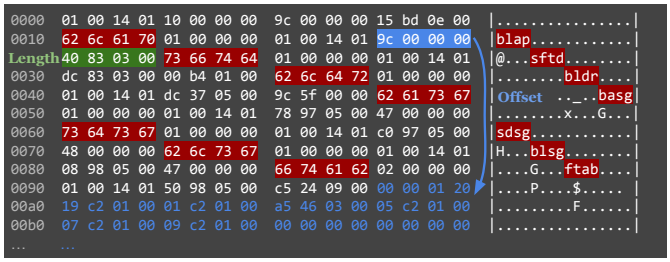


Fig. 10: Super Binary format, containing all nRF firmware sections and the U1 firmware.

manually installing a Bluetooth debug profile [30]. Thus, it is not possible to downgrade firmware on arbitrary AirTags without consent enforced by user interaction.

During the update, nRF firmware is signature-checked against a local public key on the AirTag. Creating a signed firmware is only possible for Apple, assuming that nobody else knows the according private key for signing AirTag nRF firmware. Signatures do not expire. Thus, any nRF firmware for AirTags released by Apple can be flashed over-the-air and passes validation.

U1 firmware updates also involve checking if the provided firmware version is still signed by Apple's Tatsu Signing Server (TSS). When the U1 chip enters firmware update mode, it generates a fresh nonce. The firmware update is personalized to the AirTag's nonce and its unique Exclusive Chip Identification (ECID) by the firmware update daemon. Then, the TSS is asked to sign the U1 firmware. If the firmware is no longer signed or if the nonce and ECID in the signature do not match, the update verification fails locally on the AirTag. The tool `tsschecker` can take an AirTag build manifest and check if it is still TSS-signed [34].

Using FRIDA hooks on a jailbroken iPhone, we can instrument the firmware update process to downgrade firmware as indicated by the red boxes in Figure 9. Removing and then pairing an AirTag again triggers a firmware update in 5 min. The firmware version reported by the AirTag must be older than the provided firmware update, which we overwrite with a hook in the `SPBeacon` Objective-C class that holds information about observed AirTags. The U1 firmware downgrade requires replacing the U1 firmware after Super Binary unpacking by `mobileassetd`. Between unpacking and using the U1 firmware, approximately 15 s pass. During this time window, the U1 firmware can be replaced on the file system without FRIDA hooks. Then, the U1 checksums cached

in the firmware update daemons are replaced by the checksums of the older U1 version. Within the location daemon, we can furthermore replace the nRF firmware sections. It is possible to mix non-matching nRF and U1 versions. However, if the nRF–U1 interface changes in future versions, this might soft-brick an AirTag.

4) Security Impact: The AirTag firmware has no publicly known RCE vulnerabilities. If such bugs were reported in the future, firmware downgrades would allow to execute arbitrary code on AirTags. To the best of our knowledge, updates released so far only contain anti-stalking features on the AirTag but not on the iOS side [13], [14].

Firmware downgrades are nonetheless interesting for research purposes. For example, the behavior of different firmware versions can be compared, such as anti-stalking effectiveness or U1 ranging precision.

VI. Related Work

Find My: The Find My protocol has been reverse-engineered and analyzed in detail [1], [17]. An open-source project implementing the Find My protocol and allowing it to run on low-cost hardware already exists [10]. In contrast to modified AirTags, it is a standalone solution without iOS integration since it does not implement the wireless control protocol used between iPhones and AirTags. It is also missing NFC and UWB.

AirTags: As previously shown by one of the authors of this paper and other researchers, the nRF family is susceptible to voltage glitching [23], [24], [25]. This helped us with building a low-cost glitching setup for AirTags. AirTags were torn down and initially analyzed previously [12]. We use these findings for debug pins, and similarly dump the SPI flash. While in lost mode, AirTags provide contact information via NFC. Phone number sanity checks were missing, leading to the possibility of hijacking iCloud credentials via XSS [35].

Item Finders: Item finders using a BLE offline finding network existed long before AirTags. Their underlying protocols were not optimized towards privacy, allowing for various attacks, including the server side. In the past, vulnerabilities were found within Tile, Nut, TrackR Bravo, iTrackEasy, musegear, and Cube Tracker [7], [8]. Thus, multiple protocols for secure item finders were proposed [7], [15].

nRF and BLE/NFC Security: The nRF chip on the AirTag implements BLE, NFC, and runs the main AirTag application. Even though the nRF SDK ships with plenty of firmware, the only publicly listed Common Vulnerabilities and Exposure (CVE) entry for Nordic Semiconductor relevant to nRF chips is the APPROTECT bypass [24]. *SweynTooth* found multiple bugs associated with the nRF, but only when using the Zephyr Bluetooth implementation instead of the original SoftDevice [36]. Thus, the according CVEs are listed for Zephyr. The underlying Bluetooth specification has multiple issues with BLE pairing and session management [37], [38], [39], [40]. Even though the acronym NFC suggests physical proximity between communication partners, NFC is prone to relay attacks [41].

U1 and UWB Security: UWB is a wireless technology that enables fine and secure ranging. For example, an iPhone can measure the precise distance and direction of an AirTag. UWB is also designed to support secure distance measurements in applications like car keys and payments [42], [43]. This technology is rather new and Apple was one of the early adopters integrating it since the iPhone 11, followed later by other vendors Samsung, Xiaomi, and Google [3], [44], [45]. Since this technology is very recent, only a few analysis tools exist. The basic working principle of Apple's UWB and U1 implementation was reverse engineered [3], and practical distance shortening attacks were demonstrated [46]. Nonetheless, UWB remains a mostly unexplored topic.

VII. Conclusion

AirTags have fundamental implementation issues, such as being vulnerable to firmware downgrades and using an nRF chip that is known to be vulnerable to voltage glitching attacks. These issues are rooted in the underlying hardware and general software architecture, meaning they cannot be patched without tremendous effort. Yet, these issues do not pose a huge risk for end-users. Instead, they enable researchers to instrument the firmware.

The risk for end-users is negligible compared to other threads. For example, an open-source solution implementing Find My on cheap off-the-shelf devices exists, which does not implement anti-stalking features. Also, more expensive GPS-based trackers existed before and never had stalking prevention. Our analysis enables users to understand the risks associated with AirTags.

We demonstrate the research potential by changing the behavior of the BLE and NFC functionality. Furthermore, we can modify internal AirTag behavior such as playing custom sounds and cloning AirTags. Instrumentation of the iPhone–AirTag interface enables some of these features without hardware modification.

Acknowledgment

We thank Colin O'Flynn for the initial hardware analysis of the AirTag. Furthermore, we thank Alexander Heinrich for proofreading this paper and Florian Kosterhon for supporting the U1 analysis. We also thank Lennart Wouters for assisting the AirTag cloning experiment. Additionally, we thank David Hulton for the PCB and U1 hardware analysis. Finally, we thank @doggkruse for the technical discussions and input.

This work has been co-funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

[1] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick, "Who Can Find My Devices? Security and Privacy of Apple's Crowd-Sourced Bluetooth Location Tracking System," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 227–245, 2021.

[2] Apple, "Apple introduces AirTag," https://www.apple.com/newsroom/2021/04/apple-introduces-airtag/, Apr. 2021.

[3] J. Classen and A. Heinrich, "Wibbly Wobbly, Timey Wimey – What's Really Inside Apple's U1 Chip," Presentation at Black Hat USA 2021, Aug. 2021.

[4] J. Classen, F. Freyer, and T. Roth, "Over the Air-Tag: shenanigans with the most over-engineered keyfinder," Presentation at hardwear.io Netherlands 2021, https://hardwear.io/netherlands-2021/speakers/jiska-and-fabian-and-stacksmashing.php, Oct. 2021.

[5] T. Roth, "Hacking the Apple AirTags," Presentation at DEF CON 29, https://www.youtube.com/watch?v=paxErRRsrTU, Aug. 2021.

[6] Apple, "Apple Platform Security," https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf, May 2021.

[7] M. Weller, J. Classen, F. Ullrich, D. Waßmann, and E. Tews, "Lost and Found: Stopping Bluetooth Finders from Leaking Private Information," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 184–194.

[8] D. Heiland and A. Compton, "Multiple Bluetooth Low Energy (BLE) Tracker Vulnerabilities," Rapid7, Inc., October 2016. [Online]. Available: https://blog.rapid7.com/2016/10/25/multiple-bluetooth-low-energy-ble-tracker-vulnerabilities/

[9] Apple, "Apple's Find My network now offers new third-party finding experiences," https://www.apple.com/newsroom/2021/04/apples-find-my-network-now-offers-new-third-party-finding-experiences/, Apr. 2021.

[10] A. Heinrich, M. Stute, and M. Hollick, "OpenHaystack: A framework for tracking personal Bluetooth devices via Apple's massive Find My network," in *WiSec '21: 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Abu Dhabi, United Arab Emirates, 28 June - 2 July, 2021*. ACM, 2021, pp. 374–376. [Online]. Available: https://doi.org/10.1145/3448300.3468251

[11] Nordic Semiconductor, "nRF52832 Product Specification v1.1," https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.1.pdf, Jul. 2016.

[12] C. O'Flynn, "Apple AirTag Teardown & Test Point Mapping," https://colinoflynn.com/2021/05/apple-airtag-teardown-test-point-mapping/, May 2021.

[13] Apple, "An update on AirTag and unwanted tracking," https://www.apple.com/newsroom/2022/02/an-update-on-airtag-and-unwanted-tracking/, Feb. 2022.

[14] J. Clover, "Apple Enhancing AirTags Anti-Stalking Measures With Android App and Shorter Sound Intervals," https://www.macrumors.com/2021/06/03/apple-airtags-anti-stalking-measures/, Jun. 2021.

[15] C. Garg, A. Machiry, A. Continella, C. Kruegel, and G. Vigna, "Toward a Secure Crowdsourced Location Tracking System," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 311–322.

[16] A. Heinrich, N. Bittner, and M. Hollick, "AirGuard–Protecting Android Users From Stalking Attacks By Apple Find My Devices," *arXiv preprint arXiv:2202.11813*, Feb. 2022.

[17] T. Mayberry, E. Fenske, D. Brown, J. Martin, C. Fossaceca, E. C. Rye, S. Teplov, and L. Foppe, "Who Tracks the Trackers? Circumventing Apple's Anti-Tracking Alerts in the Find My Network," in *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, ser. WPES '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 181–186. [Online]. Available: https://doi.org/10.1145/3463676.3485616

[18] Apple, "How to reset your AirTag," https://support.apple.com/en-us/HT212251, Jul. 2021.

[19] ——, "What to do if you get an alert that an AirTag or Find My network accessory is with you," https://support.apple.com/en-us/HT212227, Feb. 2022.

[20] M. Kan, "'Silent AirTags' With Speakers Removed Pop Up on Etsy, eBay," https://www.pcmag.com/news/silent-airtags-with-speakers-removed-pop-up-on-etsy-ebay, Feb. 2022.

[21] F. Bräunlein, "Send My: Arbitrary data transmission via Apple's Find My network," https://positive.security/blog/send-my, May 2021.

[22] J. van Woudberg and C. O'Flynn, *The Hardware Hacking Handbook – Breaking Embedded Security with Hardware Attacks*, San Francisco, 2022.

[23] T. Roth and J. Datko, "Chip.Fail - Glitching the Silicon of the Connected World," Presentation at Black Hat USA 2019, https://chip.fail/chipfail.pdf, Aug. 2019.

[24] LimitedResults, "nRF52 Debug Resurrection (APPROTECT Bypass) Part 1," https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/, Jun. 2020.

[25] ——, "nRF52 Debug Resurrection (APPROTECT Bypass) Part 2," https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass-part-2/, Jun. 2020.

[26] OpenOCD, "Open On-Chip Debugger," https://openocd.org/, Mar. 2022.

[27] D. J. Young, M. A. Zurcher, M. Semaan, C. A. Megerian, and W. H. Ko, "MEMS Capacitive Accelerometer-Based Middle Ear Microphone," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 12, pp. 3283–3292, 2012.

[28] Bluetooth SIG, "Bluetooth Core Specification 5.2," https://www.bluetooth.com/specifications/bluetooth-core-specification, Jan. 2020.

[29] D. Cauquil, "BtleJack: a new Bluetooth Low Energy swiss-army knife," https://github.com/virtualabs/btlejack, 2021.

[30] Apple, "Bug Reporting—Profiles and Logs," https://developer.apple.com/bug-reporting/profiles-and-logs/, Jan. 2022.

[31] O. A. V. Ravnås, "Frida - A world-class dynamic instrumentation framework," https://frida.re/, Feb. 2022.

[32] J. Levin, **OS Internals, Volume I, User Mode*, North Castle, NY, 2020.

[33] The iPhone Wiki, "OTA Updates/AirTag," https://www.theiphonewiki.com/wiki/OTA_Updates/AirTag, Feb. 2022.

[34] tihmstar, "`tsschecker`," https://github.com/tihmstar/tsschecker, Feb. 2022.

[35] B. Rauch, "Zero-Day: Hijacking iCloud Credentials with Apple Airtags (Stored XSS)," https://medium.com/@bobbyrsec/zero-day-hijacking-icloud-credentials-with-apple-airtags-stored-xss-6997da43a216, Sep. 2021.

[36] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sumei, and E. Kurniawan, "SweynTooth: Unleashing Mayhem over Bluetooth Low Energy," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 911–925.

[37] E. Biham and L. Neumann, "Breaking the Bluetooth Pairing: Fixed Coordinate Invalid Curve Attack," http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf, 2018.

[38] Y. Zhang, J. Weng, R. Dey, Y. Jin, Z. Lin, and X. Fu, "Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 37–54.

[39] J. Wu, Y. Nan, V. Kumar, D. J. Tian, A. Bianchi, M. Payer, and D. Xu, "BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[40] J. Classen and M. Hollick, "Happy MitM: Fun and Toys in Every Bluetooth Device," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 72–77.

[41] S. Klee, A. Roussos, M. Maass, and M. Hollick, "NFCGate: Opening the Door for NFC Security Research with a Smartphone-Based Toolkit," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/klee

[42] FiRa Consortium Inc., "FiRa Consortium Website," https://www.firaconsortium.org/, Feb. 2022.

[43] Car Connectivity Consortium, "Digital Key Release," https://carconnectivity.org/press-release/car-connectivity-consortium-publishes-digital-key-release-3-0/, Feb. 2022.

[44] N. Semiconductors, "NXP Secure UWB deployed in Samsung Galaxy Note20 Ultra Bringing the First UWB-Enabled Android Device to Market," https://www.nxp.com/company/about-nxp/nxp-secure-uwb-deployed-in-samsung-galaxy-note20-ultra-bringing-the-first-uwb-enabled-android-device-to-market:NW-SECURE-UWB-SAMSUNG-GALAXY, Aug. 2020.

[45] Google, "Pixel 6 Pro Technical Specifications," https://store.google.com/product/pixel_6_pro_specs, Feb. 2022.

[46] P. Leu, G. Camurati, A. Heinrich, M. Roeschlin, C. Anliker, M. Hollick, S. Capkun, and J. Classen, "Ghost Peak: Practical Distance Reduction Attacks Against HRP UWB Ranging," https://securepositioning.com/ghost-peak/, Nov. 2021.