Lost and Found: Stopping Bluetooth Finders from Leaking Private Information

Mira Weller
Secure Mobile Networking Lab
TU Darmstadt, Germany
mweller@seemoo.de

Jiska Classen
Secure Mobile Networking Lab
TU Darmstadt, Germany
jclassen@seemoo.de

Fabian Ullrich
Secure Mobile Networking Lab
TU Darmstadt, Germany
fullrich@seemoo.de

Denis Waßmann
Secure Mobile Networking Lab
TU Darmstadt, Germany
dwassmann@seemoo.de

Erik Tews
EEMCS Department, SCS Group
University of Twente, Netherlands
e.tews@utwente.nl

ABSTRACT

A Bluetooth finder is a small battery-powered device that can be attached to important items such as bags, keychains, or bikes. The finder maintains a Bluetooth connection with the user's phone, and the user is notified immediately on connection loss. We provide the first comprehensive security and privacy analysis of current commercial Bluetooth finders. Our analysis reveals several significant security vulnerabilities in those products concerning mobile applications and the corresponding backend services in the cloud. We also show that all analyzed cloud-based products leak more private data than required for their respective cloud services.

Overall, there is a big market for Bluetooth finders, but none of the existing products is privacy-friendly. We close this gap by designing and implementing *PrivateFind*, which ensures locations of the user are never leaked to third parties. It is designed to run on similar hardware as existing finders, allowing vendors to update their systems using *PrivateFind*.

CCS CONCEPTS

• **Security and privacy** → **Systems security**; *Software security engineering*; Software reverse engineering; • **Networks** → **Application layer protocols**.

KEYWORDS

Bluetooth, Privacy, Localization, Internet of Things

ACM Reference Format:
Mira Weller, Jiska Classen, Fabian Ullrich, Denis Waßmann, and Erik Tews. 2020. Lost and Found: Stopping Bluetooth Finders from Leaking Private Information. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3395351.3399422

WiSec '20, July 8–10, 2020, Linz (Virtual Event), Austria

ACM ISBN 978-1-4503-8006-5/20/07…$15.00
https://doi.org/10.1145/3395351.3399422

1 INTRODUCTION

Bluetooth finders are small devices that track an item's location. The finder's owner installs a smartphone app that maintains a Bluetooth Low Energy (BLE) connection with the finder. If the connection is interrupted, the app assumes that the corresponding item is lost and alerts the owner—for example, when leaving the house without the wallet. The limited BLE range allows triggering alarms accurately. In case a user is attempting to locate a nearby misplaced item, the app is able to play a sound on a given finder within wireless range. Localization also works in the reverse direction: Upon pressing the finder's button the smartphone plays a sound.

In addition to the sound-based localization, the app saves the smartphone's Global Positioning System (GPS) location and associates it with the finder. Position accuracy depends on the GPS fix and the maximum distance between finder and smartphone. The owner is then able to retrieve the last known location of the finder. Position data is saved locally or on a remote server.

A tracked item can move while it is out of its owner's Bluetooth range. For that reason, some finders allow other users to search for a lost item. All apps are continuously scanning for currently disconnected finders in the background and report these to the server. Once a lost finder is spotted by another user, the owner is informed. Figure 1 depicts the ecosystem architecture that is required to support external position reporting and searching.

Bluetooth finders became popular in 2013 when *Tile* raised $ 2.6 million with a crowdfunding campaign [6]. Since then, various finders were introduced to the market. In this work, we analyze top-ranked finders sold worldwide: *Nut Find3*, *Tile Mate*, *Tile Pro*, *musegear finder*, *Pearl Callstel Key Finder*, *Gigaset g-tag*, and *Cube Tracker*. Additionally, we analyze a couple of lesser known brands based on the *ST17H26* chip that can be managed over various smartphone apps including *iTracing*, *iSearching*, and *FindELFI*. Our analysis uncovers severe security and privacy issues. We reported all issues to the vendors, and most have been fixed by now. However, despite being one of the top-selling finders, *Nut* still has major security issues. We communicated these more than a year ago over various communication channels and contacted two Computer Emergency Response Teams (CERTs), but *Nut* servers are still leaking user data. This demonstrates how even today adequate product security is still being neglected by some low-cost Internet of Things (IoT) vendors.

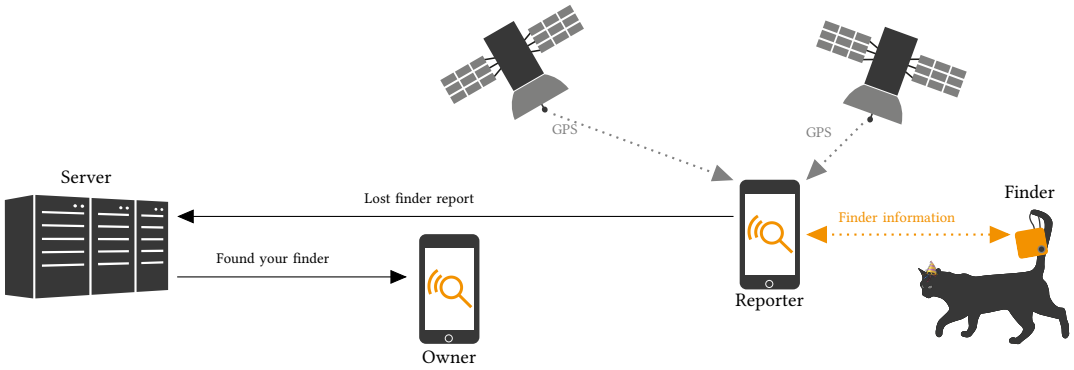


Figure 1: Bluetooth finder ecosystem overview. The finder's owner lost connectivity, but the moving finder is no longer at its last known position. A reporter within Bluetooth range sends their GPS position to the owner via the server.

Despite the variety of available Bluetooth finders, a solution maintaining privacy has yet to be proposed. Therefore, we design and implement the privacy-preserving secure finder protocol *PrivateFind* for an IoT platform similar to those seen on low-budget finders. Our contributions are as follows:

- A comprehensive analysis of features, security, and privacy in popular Bluetooth finder ecosystems.
- Uncovering severe security and privacy breaches in the most popular finders *Tile* and *Nut*.
- Generalized findings of common design flaws in IoT ecosystems that compromise security and privacy.
- Design of a new solution *PrivateFind* supporting all observed features but providing anonymity and privacy.
- Implementation of *PrivateFind* as an open-source project on a low-cost IoT platform similar to existing finder hardware.

This paper is structured as follows. We list previous security research in Section 2. In Section 3, we analyze popular finder features, with a focus on their infrastructure and security. We solve the identified issues while providing all important features with *PrivateFind* in Section 4. Our results are discussed in Section 5. Finally, our work is concluded in Section 6.

2 RELATED WORK

Heiland and Compton discovered and disclosed 12 vulnerabilities in *TrackR Bravo*, *iTrack Easy*, and *Nut* [4]. Their analysis focused on the Bluetooth interface and forging finder identities to access data. We assume that they used the same standard procedure to analyze all Bluetooth finders since the vulnerabilities found are very similar. Their most outstanding findings are: GPS positions of *TrackR Bravo* can be queried by finder identifier (CVE-2016-6540); duplicate registration of the same *iTrack Easy* finder allows to access the device's GPS data (CVE-2016-6543); and *Nut* transmits reusable session tokens without using Transport Layer Security (TLS), thereby allowing Machine-in-the-Middle (MITM) attacks on specific accounts (CVE-2016-6548). The first and second attack require a valid device identifier, meaning that an attacker must either guess it or launch a targeted attack. The third attack's severity depends on the attacker's network position. However, the security analysts did not find any *Tile* issues despite analyzing it. Yet, we were able to uncover security issues and data leaks, detailed in Section 3.2.2. To the best of our knowledge, Heiland and Compton are the only ones who did security research on Bluetooth finders. No more vulnerabilities on the products we analyzed were listed in the Common Vulnerabilities and Exposures (CVE) database.

Only recently, *Apple* added the *Find My* protocol on most of their devices [1]. As of now, *Find My* only runs on powerful devices that have a regularly charged battery. All these devices are associated with a validated *iCloud* account. This enables *Apple* to run a complex encryption scheme. In comparison, the finders in this paper are powered by a button cell for more than a year, and *PrivateFind* does not enforce the privacy-invasive equivalent of an *iCloud* account.

3 ANALYSIS OF EXISTING FINDERS

In Section 3.1 we compare features of top-selling finders. This is followed by a security analysis in Section 3.2. We conclude common features and issues in Section 3.3.

3.1 Feature Comparison

In this section, we compare the features of a large selection of finders. An overview is shown in Table 1. Despite being sold under different names, finders often share the same slightly customized firmware and Application Programming Interface (API). We perform an initial app analysis to avoid buying finders that are hardware or app duplicates by inspecting the app's code structure and API calls. We unpack and analyze the *Android* apps by searching for strings and logic analysis based on *jadx* [20].

3.1.1 Nut. The *Nut* finders support group search for lost devices and silent zones [8]. Group search means that the user can share the finder to friends with a QR code which can then help to locate the finder [7]. *Nut* is ranked *#2* on *Amazon* within the category *GPS, Finders & Accessories*.

The *Android* and *iOS* apps differ a lot. Only the *iOS* app has buttons for a lost finder search, but it never issued a request to the server in practice. An API call requesting lost finders exists in the dissected *Android* app but is never called. We assume some parts of the app were never implemented. Nonetheless, both apps report all found devices and forward them to the *Nut* servers in the background. Locations are uploaded frequently—while the app

Table 1: Finder features compared.

Feature \ App	**Nut Smart Tracker**	**Tile**	**musegear finder, iTrackEasy**	**Cube Tracker**	**keeper**	**iTracing, iSearching, FindELFI**
Device	Nut Find3	Tile Mate, Tile Pro	musegear finder, Pearl Callstel Key Finder	Cube Tracker	Gigaset g-tag	Based on ST17H26
Declare lost	✓	✓	✓	✓	×	×
Reports location to server	yes	yes	yes	yes	?	no
Login	Email/Phone+PW, Social Login	Email+PW, Social Login	Email+PW	Email+PW, Social Login	Email+PW (*Gigaset elements account*)	×

is connected to the finder or when it senses other disconnected finders. Even though locations are sent to the server, the search feature is not available for users.

3.1.2 Tile. The leading finder on *Amazon*, ranked *#1* in *GPS, Finders & Accessories*, is *Tile*. While *Nut* and *Tile* features are similar, *Tile* is more expensive and most features require a monthly subscription. Basic *Tile* features are to let the finder play a sound and to use finders to locate a smartphone. Users can add multiple smartphones to their profile and also associate a limited subset of paired non-*Tile* Bluetooth devices. *Tile* comes with a crowd search feature and claims to have the largest finding community [21]. Once a user account is associated to a *Tile*, the account and the tracker are permanently bound. Losing account access bricks all associated *Tiles*.

The location is reported every few minutes to the servers. In addition to the exact location, each click action in the app sends metadata to the server, such as the currently used Wi-Fi name and MAC address, which also allows inferring a location. Similar data is transferred every few minutes if no action is performed.

3.1.3 musegear & iTrackEasy. Several finders with a similar casing design appear on the first page of the *Amazon* category *GPS, Finders & Accessories* under various product names. One variant of this is the *Pearl Callstel Key Finder*. This type of finder uses the same API hosted on different servers. The user's location is reported to a server every 30 min even in the absence of a finder, which is more privacy-violating than the *Nut* implementation that only works with physically present finders. Also, the user's location is transferred when the user logs in and when a lost finder is reported. When a user marks a finder as lost [10], the server will report its location to the owner once it is found.

3.1.4 Cube Tracker. The *Cube Tracker* is also amongst the most popular finders. The *Cube Pro* has twice the range as the basic *Cube Tracker*. Both finders have a crowd search, a replaceable battery, and a photo trigger function [3]. Also, they can play a sound on the finder and make the phone ring. Similar to *Tile*, the *Cube Tracker* has an online sign-in that can be used to locate a finder.

3.1.5 keeper. We chose the *Gigaset g-tag* for its different technologies and app codebase, although it is not a top-selling finder. The app requires a unique *Gigaset elements account*. Users can edit their profile with the app, but the account is not required for anything—except that it is enforced to have an account. Locations are stored locally in a *Realm* [17] database and seem to be never transmitted to the server. The app's rating is poor because users expect to see their finder's location within their profile.

3.1.6 iTracing, iSearching, FindELFI. We selected the family of *iTracing*, *iSearching* and *FindELFI* finders because they are one of the cheapest top-sellers on *AliExpress*. They look identical and also share the same hardware design. They are all based on the *ST17H26* chip. Due to their simple hardware design, there is no possibility to update the finder's firmware. The apps lack all cloud-based features.

3.2 Security Analysis

We perform a technology and security analysis for all finders. Possible attack vectors strongly depend on server-related features. The general analysis follows the same categories. In addition to these categories, we perform a device-specific analysis. This analysis includes app and cloud services. An overview of the common analysis results is shown in Table 2, and the individual security issues are discussed in the corresponding subsections.

The common analysis steps and categories are as follows. Depending on the *TLS certificate validation* scheme, MITM traffic analysis requires installing a root certificate to the smartphone, or a server certificate needs to be replaced within the app. For MITM traffic analysis, we use *mitmproxy* and *Burp Suite* [9, 15]. The category *API authentication* summarizes how the app authenticates with the server. Typically, such API authentication credentials can be exfiltrated from the app. *Firmware location and encryption* refers to firmware updates if the finder supports it. Updates are forwarded by the app to the finder because it only communicates via Bluetooth. The app caches the update in some location, ideally encrypted. In some cases, we were able to download firmware updates directly from the server or to decrypt the updates once they were downloaded by the app. Ideally, the Over-the-Air (OTA) update keeps the firmware encrypted at all times until it is decrypted and verified on the finder. If the app is *obfuscated*, analyzing or changing its functions is more complicated. Non-obfuscated apps come with function names. All apps contained debug strings, which made de-obfuscation easier. The *package name* and class structure of *Java* applications as well as *API endpoints*, hint to a shared codebase between differently branded products.

3.2.1 Nut. The *Nut* ecosystem implements a rich set of features but comes with the most security issues.

Traffic Interception. The first step for analysis of the app is an MITM attack on TLS to inspect the app's behavior. Instead of checking TLS certificates, the app ignores any `CertificateException` within the class `X509TrustManager`. Thus, even an obvious MITM attack with invalid certificates is possible without installing new certificates to a victim's smartphone. Anyone with access to the same network can sniff and manipulate *Nut* app traffic.

Table 2: Common finder analysis results.

Analysis \ App	Nut Smart Tracker	Tile	musegear finder, iTrackEasy	Cube Tracker	keeper	iTracing, iSearching, FindELFI
TLS cert validation	ignored	yes	pinning	pinning	yes	n/a
API authentication	no	AWS credentials	Client certificate	AWS credentials	Client certificate	Basic authentication
FW location	Server via HTTP	Amazon AWS via HTTPS	n/a	App	App	n/a
FW encryption	App	none	n/a	none	Finder	n/a
App obfuscation	✓	×	×	✓	×	✓
Package name	com.nut. blehunter	com. thetileapp	com.antilost.finder, com.antilost.app3	com.blueskyhomesales.cube, com.shenzhen.android.cube	com.gigaset.elements. android.app.gtag2	com.fb.antiloss, com.lenzetech.antilost, com.zoqin.findelfi
API endpoint	https://api.find. nutspace.com/, https://qa-find. nutspace.com/	https://production. tile-api.com, https://locations-prod.tile-api.com	https://api.musegear. net:8092/CommApiEx, https://api.ieasytec. com:8092/CommApiEx	AWS	https://api.gigaset-elements.de/api/v1/, https://im.gigaset-elements. de/identity/api/v1/	-
Individual issues	Leakage of all user and location data	MQTT data leakage and ring control	Static TLS keystore, registration XSS	Outdated software, fake account registration, prototype pollution internal server error	—	Duplicate apps, usage statistics
Disclosure	10/7/2018, unfixed	5/2/2019	5/13/2019	1/28/2020	—	—

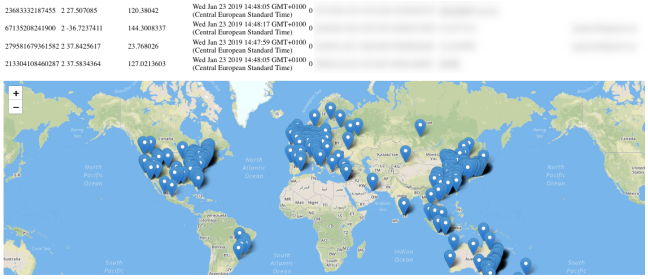


Figure 2: Anonymized snapshot most recent *Nut* reports.

Firmware Leakage. All *Nut* firmware, including unreleased product series, can be retrieved from the server. They are encrypted, but before installing an OTA update to a finder it is locally decrypted in the app with a static key.

Group Sharing. A group sharing link allows location information access to other *Nut* users. We are able to generate arbitrary group share links for known finders of other users without using the app's QR code group share feature [7]. Share links are supposed to be generated locally inside the app. They contain the finder's `deviceUUID`, an `expirationTime`, and something internally called `hmac`. This `hmac` is not what the name suggests since it is missing an encryption key. Instead, it is generated only from known values:

```
hmac=sha1(userUUID|deviceUUID|expirationTime)
```

This share link is used to request a `shareRecordUUID` from the server. In the next step, a `shareRecordUUID` can be used to obtain a `shareRecord`. This `shareRecord` contains the share's `userUUID` as well as the `userUUID` of all users the finder is shared with. Moreover, some privacy-concerning details such as the last known position of a finder are included.

This results in two vulnerabilities. First, an attacker can create share links for all devices with a known `deviceUUID` and `userUUID`. Second, an attacker can generate a new share link and invite themself even if the previous share expired or was revoked by the owner since `deviceUUID` and `userUUID` leak with a share link.

Retrieving All Lost Finder Reports. All users of the *Nut* app report lost finders. Only lost finders appear during a Bluetooth device scan, connected finders are invisible. Besides the lost finder's ID and GPS position, a report also contains the reporter's identity, including their mail address, phone number, and password hash.

An undocumented endpoint in the API leaks all recently seen devices to the public, which represents a severe privacy issue. The endpoint is contained in the app, but is never accessed during normal app operation. *Nut* did not close this issue more than a year after reporting it. Thus, we do not reveal the actual API call. Figure 2 illustrates an anonymized example of the attack's impact. We limited the request to the server to the last few lost finders. Even though the API function requires the user's `device_id` as an argument, it is ignored.

Responsible Disclosure. We wrote *Nut* an email on October 7, 2018. On January 18, 2019 we sent them 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 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 called the number listed on their Facebook site but were not able to contact them. We were also able to obtain the email address of one of the developers—we sent them an email in Chinese on April 5, 2019. CERT/CC recommended us to publicly request CVEs disclose our findings. The missing certificate validation in the app was assigned CVE-2019-16252. However, server-side issues in custom applications, as it is the case for the remaining issues, are not eligible for a CVE.

3.2.2 Tile. Cloud features in the *Tile* ecosystem are more secure than in the *Nut* ecosystem. The notification backend is based on Message Queuing Telemetry Transport (MQTT) and properly using TLS and authentication. Yet, specific implementation details make their infrastructure vulnerable.

Firmware Leakage. Firmware is available on the server in plaintext. A valid link for the associated finder type can be extracted during the firmware update process.

Public Static Credentials for MQTT Server. The app uses an MQTT server to allow remotely ringing the phone via another device. Credentials for MQTT access are requested over a public API endpoint. This way, credentials are not stored inside the app—and could change regularly. However, the MQTT credentials returned by the API are static and the same for all users. Thus, there is insufficient authentication towards the MQTT backend and no user separation.

Controlling Other Users' Phones. Once connected to the MQTT server, a request to ring any connected phone can be issued. A valid request must contain a `tileUUID`, but no further access control is performed on requests. A `tileUUID` can leak over various ways. For example, it can be requested from any nearby *Tile* via Bluetooth, and remote attackers can extract it with the attack described next.

User Data Leakage. In MQTT, data is exchanged by publishing it to topics to which clients can be subscribed. Topic subscriptions allow for wildcards by default [14, p. 57]. *Tile* blacklisted multi-level wildcards represented with `#`, but did not include blacklist single-level wildcards with `+`. When subscribing to the wildcard topic `+`, an attacker can receive all MQTT messages, including messages meant for all *Tiles* and system messages. Specific messages that can be intercepted include regular status messages of the type `CONTROL_STATUS_CHANGED` sent by the phones, which include the user's mail address, the user's `UUID`, client `UUID`, and `tileUUID`. Combined with the previous knowledge, the `tileUUID` can be used to ring the user's phone. An attacker can also subscribe to the wildcard topic `$sys/#`, the so-called sys topics. The broker responds with its uptime, program name, and version (`Erlang MQTT Broker 2.2`), as well as real-time notifications of other clients connecting and disconnecting, including their client `UUID`, user name, and IP address.

Responsible Disclosure. We disclosed all issues with detailed explanations to *Tile* on May 2, 2019. They replied within one day and applied fixes in a timely manner.

3.2.3 musegear & iTrackEasy. Attacks found in the *musegear* finder family are rather weak. However, they indicate design flaws that might originate from insufficient security testing.

Keystore. For customization, the app allows the vendor to configure different server locations. Since servers use individual TLS certificates, the app also has a local keystore with trusted certificates. The password to protect this keystore is static and the same in both apps, *musegear*, and *iTrackEasy*. This enabled us to modify the app and run a MITM attack for further analysis.

Registration Link. After creating an account, the user receives an email with a registration confirmation link. It is possible to include Cross-Site Scripting (XSS) contents in the link to change the website's appearance in the browser.

Responsible Disclosure. We informed *MS kajak7 UG* and *KKM Company Limited*, the companies behind *musegear* and *iTrack Easy*, on May 13, 2019. *musegear* reacted within less than an hour to our first contact attempt, and we discussed all findings.

3.2.4 Cube Tracker. While we could not find any severe data leakage within the *Cube* ecosystem, there are a lot of minor security issues. Their infrastructure is hosted on Amazon Web Services (AWS), and all traffic passes an *Nginx* server. Further services running in the backend are *AngularJS*, *ExpressJS*, *Mongoose*, and *MongoDB*.

Firmware Leakage. The firmware is stored locally in the app. It is neither signed nor encrypted.

Query Existing Users. The login website displays different error messages for login failures versus non-existing users.

Fake Account Generation. Instead of confirming an email on account generation, a user can call the API endpoint `PUT /users/<id>/edit`. This allows the attacker to create an endless amount of accounts without the mail overhead.

Profile Picture. The profile picture can be set to an external URL. However, as far as we observed its usage, this picture is only presented to the user themself.

Server-side JavaScript Prototype Pollution. It is possible to alter *JavaScript* values by sending malicious JSON payloads to *Mongoose*. For example, we were able to pass a payload that turns the `User` type into a generic `Object` type. When `User`-related methods are then called on the `Object`, this causes an internal error of type `500` because the method is not found. Note that this happens only within that request, it does not escalate into the whole web service. We cannot exclude that remote code execution becomes possible via this issue as we do not have access to the server's implementation, but assume that this is very unlikely.

Outdated Software. The *Nginx* webserver indicates `nginx/1.13.7` in the header, which is an outdated version with multiple publicly known CVEs. Also, the *AngularJS* version 1.3.20 is out of support.

Responsible Disclosure. We contacted *Cube Tracker* on January 28, 2020, and also sent a second mail containing more details. They promptly confirmed the reception, but communication did not continue later, likely due to COVID-19.

3.2.5 keeper. We could not find anything remarkable. The app does not have any connected functionality besides the account, which is never used except for logging in.

3.2.6 iTracing, iSearching, FindELFI. Even though the apps for finders based on the *ST17H26* chip do not report locations to the server according to what we observed, their apps lack privacy and user-friendliness.

Usage Statistics. During installation, the app connects once to a server to transmit usage statistics. The statistics only contain finder information but no user account.

App Duplicates. There seem to be around 15 copies of the original app in *Android's Google Play Store*. In the best case, it is just customization for different resellers. However, it might be that some copies contain malware or leak location data, even though *Google* regularly checks for malware in apps.

3.3 Analysis Results

The hardware implementation of all Bluetooth finders is rather simple. Finders themselves are not aware of locations or user data. Logic to find lost items is implemented in the app and cloud. A common architecture issue is that the cloud is retrieving GPS data

from all users in plaintext. No matter if there is a vulnerability in the implementation or not, users need to trust the cloud operator to keep their location information private. Most apps continuously transmit finder locations to the cloud, regardless of their lost state. In the *Tile* and *Cube* ecosystems, which feature a Web platform where users can view their device locations from anywhere, this is reasonable. In contrast, *Nut* finders are bound to a smartphone installation anyway. Location traces are very privacy-sensitive as they reveal a lot about the user's habits—and, thus, could be sold for marketing purposes and similar. Our security analysis shows that even the leading vendors fail to protect location and user data in their clouds from being extracted by external attackers.

We claim that existing Bluetooth finders are privacy-invasive by design. In most situations, GPS data could be stored locally inside the app. Only in lost mode, reporters need to transmit location information to the owner—which can be end-to-end encrypted. Minimizing private data transferred and stored in plaintext reduces the risk of data leakage if there are security issues.

4 PrivateFind: AN OPEN, SECURE, AND ANONYMOUS FINDER SOLUTION

In the following, we describe *PrivateFind*, which features a similar architecture and hardware design as existing commercial products. *PrivateFind* enables finder crowd search without leaking private data. It prevents data leakage by design, as the server never sees any GPS locations in plaintext. Moreover, it enables anonymous usage of the crowd search ecosystem. It aims at preventing tracking by both the server infrastructure as well as other users. *PrivateFind* is publicly available on https://github.com/seemoo-lab/privatefind.

We define two setup variants with different security and privacy guarantees in Section 4.1. An anonymous lost finder reporting system is introduced in Section 4.2. Reports are independent of the setup procedure and, thus, compatible with both variants. We discuss protocol design decisions in Section 4.3. The hardware used for our open-source prototype and the corresponding *Android* app is shown in Section 4.4.

4.1 Setup Procedure

The setup procedure establishes an end-to-end encryption key and identifiers, proves that the owner currently owns the finder, and comes in two variants.

End-to-end Encryption Key. During the setup, the finder and the smartphone establish a pre-shared key `e2e-key`. The `e2e-key` can be reset by running the setup procedure again. Keys are individual per device; any leaked key will only compromise one finder. The `e2e-key` is symmetric due to the finder's hardware limitations.

After the setup, this key never leaves the finder and the smartphone. However, the finder can receive plaintext messages and encrypt them with this key, such that only the smartphone can decrypt it. This mechanism is used to hide GPS locations of reports from the server by using the finder to encrypt reports about itself. Moreover, generating reports requires the physical presence of the respective finder. Based on these reporting properties, the reporter can stay anonymous when sending reports to the server.

Initial and Randomized Identifier. Moreover, the finder reveals its fixed identifier $\texttt{id}_{init}$ during setup. This identifier never changes, even if the finder is reset by repeating the setup procedure. The format of this identifier is implementation-specific, e.g., it could be a 256 bit random value. Both, the smartphone and the finder, use it in conjunction with the `e2e-key`, to derive randomized identifiers in fixed time intervals:

$\texttt{id}_{rand,n+1} = \texttt{hmac(e2e-key, id}_{rand,n}\texttt{)}$

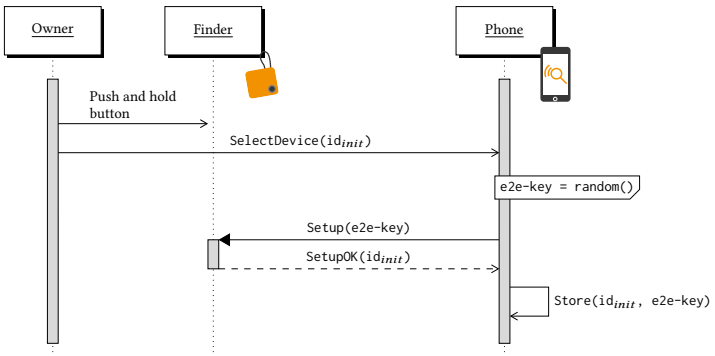
While $\texttt{id}_{init}$ never leaves the finder and the smartphone (but is known to the server in the manufacturer-verified setup), the randomized $\texttt{id}_{rand}$ can be requested by nearby devices if the owner's smartphone lost the connection and if the finder internally can confirm that it has not seen its owner since a while. Thus, *PrivateFind* further increases privacy by only revealing $\texttt{id}_{rand}$ if necessary. The finder does not appear in scan results while it has a connection to its owner, and it can choose to refuse excessive identity requests, even on the randomized and regularly changing identifier.

Proving Ownership. The setup mode that resets `e2e-key` and leaks the unique identity $\texttt{id}_{init}$ can only be performed by the finder's owner. To enter the setup mode, the owner pushes and holds the finder's button. Only after pressing the button like this, the finder enters setup mode. Otherwise, it will not accept a reconfiguration. Once the setup is finished, the setup mode must be reactivated again by pressing the button if needed. We assume an attacker with physical access who wants to steal an item attached to a finder could as well remove the finder's battery or shield it in tinfoil. This is similar to the security assumptions made by other finder products.

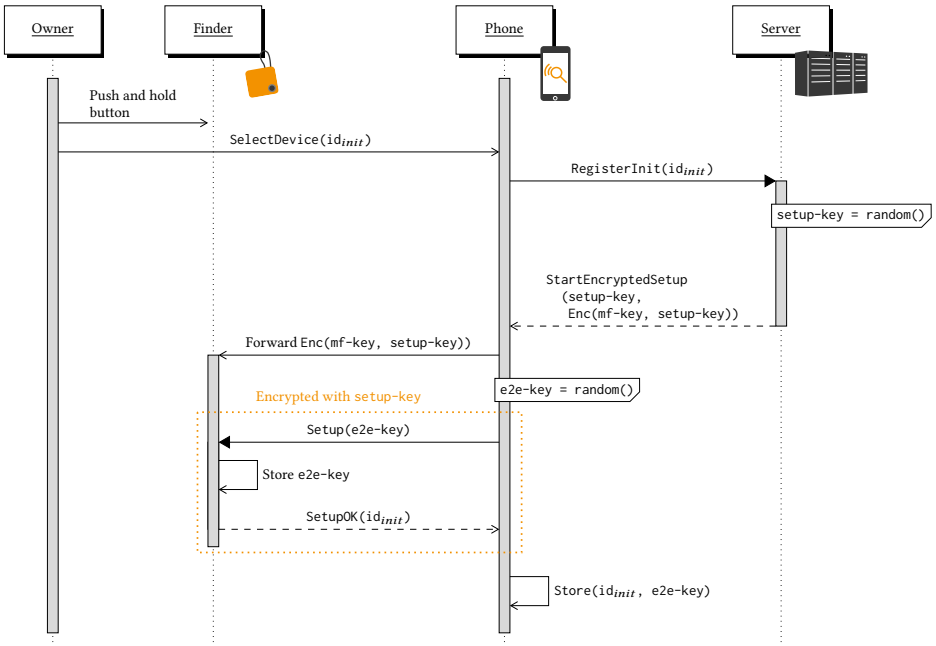
Setup Variants. While both setup variants establish an `e2e-key`, exchange the $\texttt{id}_{init}$, and ensure ownership by physical access, they slightly differ. The local variant in Section 4.1.1 features a mode that enables compatibility between different finder manufacturers. The manufacturer-verified variant in Section 4.1.2 enables the manufacturer to verify a finder's identity and provides further security to the Bluetooth communication. While we recommend using the local variant to improve privacy, manufacturers might prefer the manufacturer-verified variant, as it is closer to the ecosystems provided by existing products. Both variants do not require the user to register a personal account.

4.1.1 Local Setup. The local setup, which is shown as a sequence diagram in Figure 3a, ensures that the user has physical access to a finder. Moreover, the setup exchanges `e2e-key` and $\texttt{id}_{init}$. There is no third party involved in the local setup and there is no registration with a server. Note that in this local variant $\texttt{id}_{init}$ can also be reset to a random value during the setup, as there are no external dependencies on it.

4.1.2 Manufacturer-Verified Setup. The manufacturer-verified setup increases security at the cost of privacy. It provides the same security properties as the local setup. On top, it enables the manufacturer to validate that the finder is indeed one they created, and adds manufacturer-verified encryption to the Bluetooth setup. Even though finder validation makes the server able to identify a finder during setup, messages containing precise GPS location data will always be end-to-end encrypted between the finder and the user who registered a finder with their smartphone. Thus, also



(a) Local setup without proof of ownership.



(b) Manufacturer-verified setup that proofs ownership of a legitimate device to the manufacturer.

Figure 3: Initial setup and registration variants.

this protocol variant ensures more privacy than any finder ecosystem we encountered in the wild. The overall process is depicted in Figure 3b.

Manufacturing Requirements. The manufacturer might want to verify their manufactured devices. To this end, the manufacturer installs an individual manufacturing key `mf-key` on each finder, which is associated with its identifier $\texttt{id}_{init}$. The `mf-key` is the root of trust between server and finder. Each finder has an individual `mf-key`, which is never transmitted in plaintext but can be identified by its $\texttt{id}_{init}$.

Verification During Setup. The finder notifies the server that $\texttt{id}_{init}$ is now active. *PrivateFind* does not enforce any account registration, but in case the manufacturer would like to add this as a property, they could add a `mf-key`-based setup challenge in this step. Note that, depending on the remaining implementation, this would also add reporter identities to the encrypted location reports.

Encrypting Bluetooth Communication. We assume communication between app and server is not compromised since it can be protected with TLS [18]. The server certificate must be validated and ideally is pinned [19]. Encryption methods employed by TLS are not feasible on a low-cost finder device. Hence, the server generates a temporary random `setup-key`, which is used to encrypt the setup procedure. The communication during the setup, encrypted with `setup-key`, is marked with an orange dotted box in Figure 3b. The `setup-key` is transferred to the app in two formats: plaintext and encrypted. The plaintext `setup-key` is for the app; the app does

not have the mf-key to decrypt the encrypted one. The app forwards the encrypted setup-key to the finder via Bluetooth. Only a legitimate finder can decrypt it with its mf-key.

This possibility of securing the registration procedure with the mf-key improves security. We do not consider Bluetooth encryption to be sufficient. Since the finder neither has a display nor a keyboard, *Just Works* pairing is applied. Assuming BLE 4.0 or 4.1 on at least one of the involved devices, this mode is susceptible to passive MITM attacks due to weak encryption methods [2, p. 277]. Even in BLE 4.2 and higher, *Just Works* can be actively eavesdropped, as stated in the Bluetooth 5.2 specification [2, p. 274]. Using the mf-key as an additional root of trust mitigates this MITM risk.

4.2 Privately Reporting Lost Finders

Based on the initial setup information, finders can be located, as shown in the sequence diagram in Figure 4. This mechanism does not leak the lost finder's location to the server, and the reporter's identity is not revealed.

A lost finder can be found by anyone because it will be discoverable in Bluetooth scanning if it is not connected to the owner's smartphone. The reporter can send a message to a lost finder that contains the current GPS position. The finder answers this with an encrypted message that can only be decrypted by the owner with e2e-key. The finder only answers this message if it has not seen the owner for a few minutes to ensure that no unnecessary reports are generated. Finder and owner can use a pseudo-random sequence, if available on the finder, to prevent a replay of old locations. The current *PrivateFind* implementation prevents this with a counter and an authenticated encryption mode. The owner can look up reports by its calculated recent list of $\texttt{id}_{rand}$.

4.3 Implementation Variants

In the following, we discuss implementation variants and point out the current state of the *PrivateFind* implementation.

Identity Randomization. The finder should randomize its MAC address as $\texttt{mac\_addr}_{rand}$ to prevent tracking by nearby devices. Thus, the finder changes its address in regular intervals—otherwise, the randomization of the finder's identifier $\texttt{id}_{rand}$ could be bypassed. This MAC address randomization feature is already included in the Bluetooth specification to ensure privacy for BLE [2, p. 3064ff]. Usually, this address changes every 15 min, but the interval can be lowered to increase privacy. The interval of the MAC address change and $\texttt{id}_{rand}$ update should be synchronized to prevent tracking via asynchronous changes.

Note that the current *PrivateFind* code release is not using MAC address randomization, because it requires MAC addresses for a simplified, non-randomized finder identification during location reports.

Report Delivery Modes. PrivateFind uses *direct delivery* of reports to an owner. For direct delivery, the owner messages the server with the currently valid $\texttt{id}_{rand}$ or a sequence of recently valid identities.

An alternative approach would be *broadcast delivery*. Since reports are encrypted, locations would not leak. This form of broadcast delivery hides the fact which finder was lost and found at which point in time from the server. However, someone who lost an item needs to observe all reports, which causes a lot of traffic and leaks recently valid identities.

In the current *PrivateFind* implementation, a reporter is not getting any feedback. Reporters cannot verify if a finder exists. This increases security against attackers trying to leak valid identities.

Verifying Reporters and Reports. PrivateFind does not verify a reporter's identity on the server-side to increase privacy. Owners can check locally if a report was indeed created by their lost finder if it decrypts correctly and contains a valid number of a pseudo-random sequence. Moreover, anonymous location reports can be displayed differently inside the app than locations observed by the user. This helps the user to check themself if the reported location seems legit.

The server could verify if the reported finder is real by sending a challenge similar to the one in the setup. The challenge can again make use of the mf-key shared between server and finder, but with the reporter as a relay. Such a challenge would make it impossible

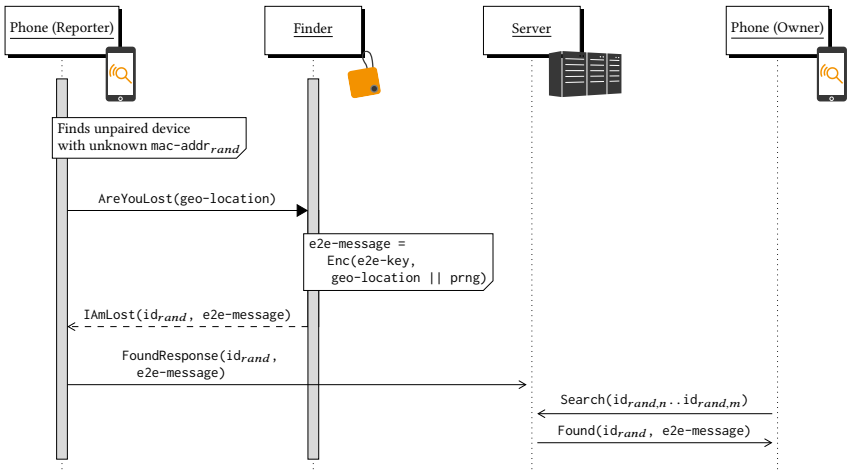


Figure 4: Anonymous report of witnessed device.

to replay reports already on the server-side. A disadvantage of this approach is an increased load on the server for a property the owner can also verify locally. Moreover, it would require the server to ask for $\texttt{id}_{init}$ to look up the according `mf-key`, which would deanonymize reports.

The server could also restrict usage of their ecosystem to those users who legitimately purchased a finder. For this, the manufacturer-verified setup needs to be extended as follows. The server can send a challenge bound to a finder's `mf-key` that can only be answered by a legitimate finder. If this challenge is answered correctly, the server can either confirm a user's account registration with this, or issue an account-less access token. Later, when users report lost finders or search for them, these actions can be bound to that account or token. While the GPS locations in such an approach remain encrypted, this always reveals the user's or finder's identity. However, even this extension would still be more privacy-preserving than existing approaches that all share GPS locations to the server. The current *PrivateFind* code release has an option for such a token.

Deciding if a Finder is Lost. A finder that lost the connection to the owner's smartphone appears in Bluetooth scan results of other smartphones. This might create a lot of reports in crowded places with bad connection quality. Hence, the reporter asks the finder if it has been disconnected for a while, instead of reporting it immediately. Yet, this still imposes a privacy issue. There is a possibility that an owner lost the connection but still knows the item's location, e.g., if the smartphone battery is empty. The report includes the reporter's IP address, which can be used to make a raw guess on the finder's geolocation.

Our *PrivateFind* implementation enables users to opt-out from receiving reports by setting a flag in the finder that disables the generation of reports. As long as this flag is set, the finder will never answer to an `AreYouLost` message.

The server can keep track of owners who report their finder as lost and propagate this information to reporters. Even though this enables all users to see which finders are currently lost, the public information is not associated with an IP address and the included $\texttt{id}_{rand}$ is randomized.

Metadata on the Server. Users still need to trust the server in some means. The server can estimate the geolocation by IP addresses. Message timing also allows it to guess whether reporter and owner are nearby and know each other. However, in *PrivateFind* this information is only transferred if the finder confirms to be lost. In contrast to the default behavior in most commercial products, the amount of reports is minimized and additionally anonymized. In general, users being concerned about geolocations leaking by IP addresses should use anonymization techniques such as Tor [16]. Users could be selfish and not send reports for anything they found but still profit from other reporters if they are concerned about leaking information when reporting.

Identity Export. A user might use a finder with multiple phones or backup the finder association in case of phone loss, the `e2e-key` could be exported with a QR code. Without export, only one smartphone at a time is supported. To replace the smartphone, the finder must run through the setup procedure again, which resets the `e2e-key`.

4.4 PrivateFind Implementation

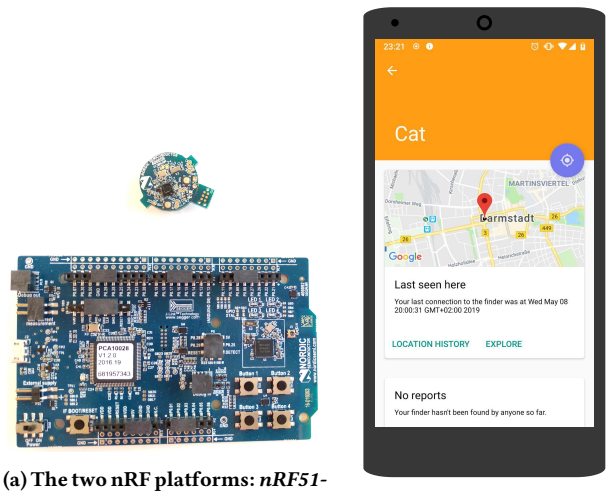
In the following section, we detail how we realized *PrivateFind* in hardware, firmware, and as an *Android* app.

Hardware Platform. After disassembling common Bluetooth finders, we became aware of the *nRF51822 Bluetooth Smart Beacon Kit* [13]. Its diameter is as small as 20 mm, and it comes with all the important features. During development, we used the *Bluetooth Low Energy Development Kit for the nRF51 Series* [11]. It is based on the same chip but meant for development, which means the board comes with additional input and output possibilities and is easier to flash—limitations for firmware running on the chip stay similar.

The development platform already comes with a basic finder example, which triggers an alarm on Bluetooth connection loss. Only the *PrivateFind* setup, registration, and report procedures had to be implemented.

Hardware Optimization. The development platform already offers some encryption methods, such as Advanced Encryption Standard (AES). Encryption in the lost finder reporting is Authenticated Encryption with Associated Data (AEAD), e.g., AES-CTR with a random nonce for the associated data and SHA256-HMAC for authentication. Depending on the hardware platform, different encryption methods can be used.

Android Application. The app runs in the background and maintains a Bluetooth connection with the finder established. This is implemented using the *Android* BLE library by *Nordic Semiconductor* [12]. The last known position of a finder is saved locally and can be displayed as shown in Figure 5b. Once the connection is lost, it plays a sound on the smartphone or on the finder, according to the user's preferences. Moreover, the app searches for other lost finders in the background and reports these. To preserve battery on the smartphone, we use the hardware offloading of the Bluetooth controller if supported.



(a) The two nRF platforms: *nRF51-DK* (bottom) and *nRF51822* (top). **(b) Mobile app.**

Figure 5: *PrivateFind* implementation.

5 DISCUSSION

Crowd Search. Distributed item search requires a large user base with the app installed and running. For example, finders of three different brands were marked as lost, and the crowd search feature was not able to find them near a busy train station in Germany with approximately 250 000 people passing by in 2017 [5]. A lost item can only be found within communities that commonly use finders. This problem gets worse with the diverse finder market since reporting systems are not compatible at all. We observed that online shop ratings of the same finders differ a lot, depending on the platform and region where they are sold, and assume this is due to the distinct communities using these finders.

Privacy Policies. Besides the technical problems we uncovered in common Bluetooth finders, their basic concept is already privacy-invasive depending on how data is handled exactly. Thus, we did a check on the privacy policies of finders with online functionality.

Nut There are two different privacy policies for *Nut*. One is publicly available on their website but only refers to data their webserver collects while browsing their sites, and another one that is displayed when installing the app. We consider the policy in the app to be more relevant. Yet, the app privacy policy of *Nut* is surprisingly short and written in very vague terms. The policy does not even state that *Nut* is collecting location data. We assume that such a policy is not acceptable under the EU General Data Protection Regulation (GDPR).

Tile Two privacy policies were in place at the same time before our report. When a user tried to create an account in the app, a privacy policy from 2016 was shown. However, on the website of *Tile*, a new policy from 2018 is shown [22]. The policy of the app states that the location of the user is transmitted periodically. It also states that *Tile* may collect information for statistical purposes that cannot be traced back to an individual user anymore. In general, we consider the privacy policy of *Tile* to be well-written and understandable.

musegear The privacy policy by the *musegear* app looks very similar to *Tile*. Their privacy policy states that location data should be regularly saved on the device, but does not mention the regular transmission of this data. Also, their policy is hard to read due to formatting mistakes.

CubeTracker The privacy policy of *Cube Tracker* is surprisingly well-written, but still contains a placeholder where the tax identifier of the company should be inserted. It allows *Cube Tracker* to share data with third parties and also to update the policy, and the user will be notified about significant changes. Also, it does not differentiate which data is collected by the app locally and which data is shared with the cloud service and third parties.

To summarize, the privacy policies of all tested Bluetooth finders with connected features do not accurately reflect what the finders and corresponding apps are doing. Therefore, we would like to encourage the manufacturers to update their privacy policies. A general issue is that the privacy policies in the apps differ from the ones on the websites, and the policies are subject to change. This makes the privacy policies very opaque in addition to missing details on how data is processed and stored.

6 CONCLUSION

We conducted a comprehensive analysis of the most popular Bluetooth finders currently on the market and analyzed their security and privacy. None of the market-leading products is designed in a privacy-friendly way, and several of them have serious security flaws on multiple levels:

- All products tested were designed and implemented without a focus on privacy.
- Some of the tested products forced the user to create an account for a cloud service not immediately required for using the product. Other products automatically reported the user's location to a cloud service without an observable reason.
- Some of the products had minor security vulnerabilities, such as insufficient protection of the API against unauthorized communication. Furthermore, some products omitted proper TLS certificate validation of the backend services.
- A few products had serious security vulnerabilities in their corresponding backend that enabled attackers to obtain access to private data of other users.
- One vendor ignored our reports about those weaknesses for more than a year, despite multiple attempts to get in touch with them over several communication channels. Until today, the corresponding cloud service leaks private customer information that can be accessed easily.

Some implementations are highly suspicious, and we cannot rule out that this data may be collected for other purposes. Many of those security and privacy problems can be fixed easily in the app and the corresponding backend services, for example by not sending any unnecessary data to the cloud service. We decided on a more thorough approach and designed and implemented the privacy-friendly Bluetooth finder *PrivateFind* that prevents location leakage to the cloud service. Users are able to report a lost and found Bluetooth finder anonymously, and we believe our system achieves a more advanced security and privacy standard than any commercial system we analyzed. Our system provides the same features as most commercial products and runs on the same or comparable hardware. This shows that privacy-friendly and secure Bluetooth finders can be built without increasing expenses for the hardware and without a loss of features for the user.

ACKNOWLEDGMENTS

We thank Max Maass and Nils Ole Tippenhauer for the discussion about the *PrivateFind* protocol, Vanessa Hahn for the graphics design, and Matthias Hollick for his support.

This work has been 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] Apple. 2019. Set up Find My on your iPhone, Mac, and other devices. https://support.apple.com/en-us/HT210400.
[2] Bluetooth SIG. 2020. Bluetooth Core Specification 5.2. https://www.bluetooth.com/specifications/bluetooth-core-specification.
[3] Cube Tracker. 2020. Cube Tracker – Instructions. https://cdn.shopify.com/s/files/1/0257/8998/8936/files/cube_tracker_instructions_EN.pdf.

[4] Deral Heiland and Adam Compton. 2016. Multiple Bluetooth Low Energy (BLE) Tracker Vulnerabilities. https://blog.rapid7.com/2016/10/25/multiple-bluetooth-low-energy-ble-tracker-vulnerabilities/
[5] Jan-Keno Janssen, Michael Link, and Stefan Porteck. 2017. Verliermeinnicht: Neun Bluetooth-Tags im Test.
[6] Natasha Lomas. 2013. Tile Grabs $2.6M Via Selfstarter For Its Lost Property-Finding Bluetooth Tags Plus App. https://techcrunch.com/2013/07/24/tile-grabs-2-6m-via-selfstarter-for-its-lost-property-finding-bluetooth-tags-plus-app/
[7] NutTag Australia PTY LTD. 2019. Group Share. https://nuttag.com.au/blogs/user-guide/group-share
[8] NutTag Australia PTY LTD. 2019. Silent Zones - Region (iOS). https://nuttag.com.au/blogs/user-guide/silent-zones
[9] Mitmproxy Project. 2020. mitmproxy - an interactive HTTPS proxy. https://mitmproxy.org/.
[10] musegear.net 2019. *Musegear Finder User Manual.* musegear.net. https://musegear-finder.net/wp-content/uploads/2019/01/Bedienungsanleitung_original.pdf.
[11] Nordic Semiconductor. 2019. Bluetooth Low Energy Development Kit for the nRF51 Series. https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF51-DK
[12] Nordic Semiconductor. 2019. Nordic BLE Library. https://github.com/NordicSemiconductor/Android-BLE-Library
[13] Nordic Semiconductor. 2019. nRF51822 Bluetooth Smart Beacon Kit. https://www.nordicsemi.com/Software-and-Tools/Reference-Designs/nRF51822-Beacon-Kit
[14] OASIS. 2014. *MQTT Version 3.1.1.* Technical Report. OASIS. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.
[15] PortSwigger. 2020. Burp Suite - Cybersecurity Software from PortSwigger. https://portswigger.net/burp.
[16] Tor Project. 2019. Tor Project | Anonymity Online. https://www.torproject.org
[17] Realm. 2019. Realm Database. https://realm.io/products/realm-database
[18] E. Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3.* RFC 8446. RFC Editor.
[19] David Sounthiraraj, Justin Sahs, Garret Greenwood, Zhiqiang Lin, and Latifur Khan. 2014. SMV-Hunter: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014.* The Internet Society. https://www.ndss-symposium.org/ndss2014/smv-hunter-large-scale-automated-detection-ssltls-man-middle-vulnerabilities-android-apps
[20] Maddie Stone. 2020. Android App Reverse Engineering 101. https://maddiestone.github.io/AndroidAppRE/.
[21] Tile. 2019. How it Works. https://www.thetileapp.com/en-us/how-it-works
[22] Tile. 2019. Privacy Policy. https://www.thetileapp.com/en-us/privacy-policy