Tracking You from a Thousand Miles Away! Turning a Bluetooth Device into an Apple AirTag Without Root Privileges

Junming Chen
George Mason University

Xiaoyue Ma
George Mason University

Lannan Luo
George Mason University

Qiang Zeng*
George Mason University

Abstract

Apple's Find My network, leveraging over a billion active Apple devices, is the world's largest device-locating network. We investigate the potential misuse of this network to maliciously track Bluetooth devices. We present `nRootTag`, a novel attack method that transforms computers into trackable "AirTags" without requiring root privileges. The attack achieves a success rate of over 90% within minutes at a cost of only a few US dollars. Or, a rainbow table can be built to search keys instantly. Subsequently, it can locate a computer in minutes, posing a substantial risk to user privacy and safety. The attack is effective on Linux, Windows, and Android systems, and can be employed to track desktops, laptops, smartphones, and IoT devices. Our comprehensive evaluation demonstrates `nRootTag`'s effectiveness and efficiency across various scenarios.

1 Introduction

Bluetooth trackers, such as AirTag [7] and Tile [61], utilize crowd-sourced finding networks, allowing owners to locate their trackers precisely. A tracker sends advertisement messages over Bluetooth Low Energy (BLE), while nearby devices (i.e., finders) that are in the same finding network as the tracker receive the messages and then report their locations (latitude/longitude coordinates) to a cloud. A tracker does not need a GPS module but relies on the GPS modules of such finders to report locations. Among various tracking networks, Apple's Find My network is the largest, leveraging over a billion active iPhones and other Apple devices. This extensive network makes it more likely that a lost AirTag will be detected and its location reported compared to other trackers. Therefore, this work is focused on Find My network.

Prior work, `OpenHayStack` [31], reported that devices like the ESP32 could be turned into trackers leveraging Apple's Find My network without requiring Apple's approval. (Note that this is different from Apple-certified third-party trackers, which participate in the Find My network through a paid membership in the Apple program [6].) `OpenHayStack` requires root privileges (or a device that does not distinguish root and non-root processes at all) to turn a device into a tracker [29]. However, root privilege escalation usually requires non-trivial efforts in the attacking scenario and does not always succeed. Thus, `OpenHayStack` only works in limited scenarios. In contrast, we aim at a widely applicable attack method that can turn a remote computer into a tracker without depending on root privileges.

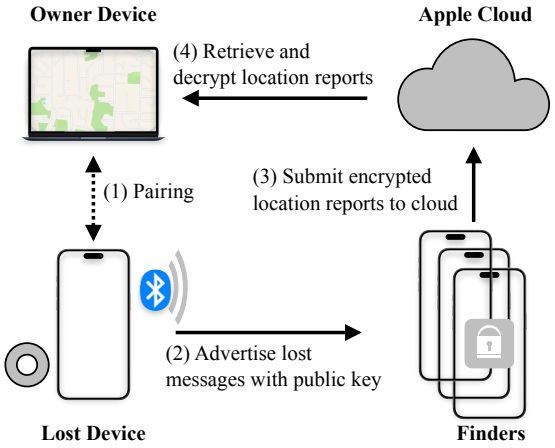


Figure 1: Overview of Find My offline finding.

Figure 1 illustrates the overview of Find My offline finding. (1) Through pairing, an AirTag shares the public/private key information with the owner's device. (2) When the AirTag is separated from the paired device, it advertises its public key via BLE advertisements, known as *lost messages*. (3) Nearby Apple devices, referred to as *finders*, generate encrypted location reports and send them, along with the hashed public key, to the Apple Cloud. (4) The Apple Cloud allows anyone to use a hashed public key to retrieve the associated location reports, which can only be decrypted using the correct private key. To ensure anonymity, finders do *not* authenticate whether a lost message is sent from an Apple device [31].

*Corresponding author.

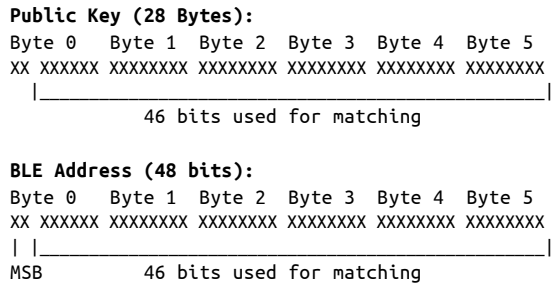


Figure 2: Relationship between the public key and the BLE advertising address.

The BLE specification limits an advertisement payload to 31 bytes, which is insufficient for a lost message. To cope with the tight limit, the Find My protocol stores a part (46 bits) of the public key in the BLE advertising address field. As illustrated in Figure 2, if the least significant 46 bits of a BLE address are equal to the least significant 46 bits of the first 6 bytes in a public key, we say they *match*. These 46 bits are referred to as the *critical* part of the advertising address or public key. To craft a lost message, `OpenHayStack` first prepares a public/private key pair, and then modifies the advertising address to match the public key. Modifying the advertising address, however, requires root privileges.

Instead of modifying the advertising address, we propose to search for a public/private key pair that matches the advertising address and then send lost messages advertising the public key. However, the Find My specification [5] mandates the use of a *random static* address for advertising lost messages, whereas our investigation reveals that devices usually do *not* use such addresses for advertising. This seems to create a barrier to our strategy of using the existing address for advertising lost messages.

Nevertheless, our further experiments reveal that all types of BLE addresses are allowed by the Find My implementation for advertising lost messages. Building on this finding, we present a novel attack method, `nRootTag`, which turns a computer into a tracker without needing root privileges. Depending on the operating system, one of the two attack techniques is employed. (1) On Linux, a *public* address is used for advertising and can be accessed without root privileges. Our observation is that the public address of a BLE chip has 24 bits of "Company ID" (also known as OUI) assigned by the IEEE, and these registry records are publicly accessible. Leveraging the public information, we can precompute rainbow tables that store the keys for various public addresses. As a result, given a public address, the matching public/private key pair can be retrieved instantly. (2) On Android and Windows, advertising typically uses a random *resolvable* private address and a *non-resolvable* private address, respectively, and the address cannot be accessed without root privileges. We consider an attack scenario, for example, where a popular app is installed on multiple computers at a location, such as a home, mall or office building. The app then can use one computer to sniff the advertisements from another and obtain the advertising address. Subsequently, the attacker can use rainbow table lookup or online key search to find a matching public/private key pair. We design a custom database to significantly reduce the storage space required for the rainbow table. To accelerate the key search, we implement online search to leverage multiple GPUs.

We make the following contributions.

- While the Find My specification mandates the use of random static addresses for advertising lost messages, computers typically do not use such addresses for BLE advertisements. Our study, however, reveals that the Find My service actually allows all types of BLE addresses. This has been overlooked by both prior work and Apple, but is leveraged in our attack.
- We present the first attack method, `nRootTag`, that turns a computer into an "AirTag" tracker without root privilege escalation.[1] We devise attack methods that can be applied to Linux, Windows, and Android systems.
- Our implementation significantly saves the space needed by rainbow tables and accelerates online key search through GPUs. The online key search success rate is over 90% in 3 minutes at a cost of just a few US dollars. Interestingly, the cost does *not* increase as the number of tracked computers grows.
- The evaluation results show that various computers, such as desktops, laptops, smartphones, and IoT devices, can be turned into trackers without needing root privileges, and that the attack is effective on Linux, Windows, and Android. We also discuss how the attack can be mitigated and extended to other systems, such as Apple.

2 Background

2.1 Bluetooth Low Energy Addresses

As illustrated in Figure 3, the BLE protocol defines two main address categories: Public Address and Random Address, which are distinguished by the TxAdd bit in the header.

Public Address (TxAdd = 0). The public address is designed to be globally unique and remains constant throughout the device's lifespan, serving as a permanent identifier. It comprises two parts: the most significant 24 bits, assigned by the IEEE and known as the Organizationally Unique Identifier (OUI), and the least significant 24 bits, assigned by the device manufacturer. As of December 2024, IEEE has published 47,054 OUIs [46]. Studies have shown that the use of public addresses poses privacy and security risks, allowing applications to identify and track users [35, 70].

[1] https://nroottag.github.io

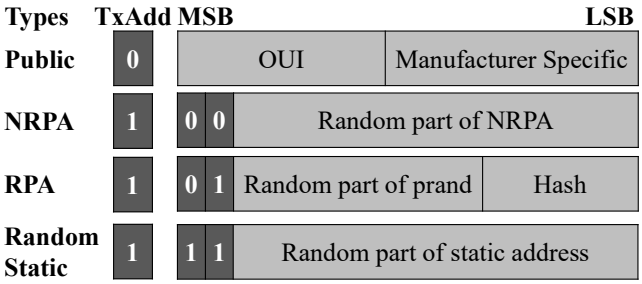


Figure 3: Categories of BLE addresses

Random Address (TxAdd = 1). Unlike public addresses, random addresses do not include vendor information. As shown in Figure 3, random addresses are classified into three types, distinguished by the two most significant bits: *0b00* for non-resolvable private address (NRPA), *0b01* for resolvable private address (RPA), and *0b11* for random static address. Random addresses are designed to enhance privacy by changing periodically, referred to as *address rotation*. According to the specification [15], a random static address is generated at power-up and remains unchanged until the next reboot, and an RPA or NRPA typically changes at regular intervals to further protect user privacy.

2.2 Find My Service

In 2019, Apple introduced its Find My service for iOS 13, macOS 10.15, and watchOS 6, enabling users to locate their Apple devices via BLE without relying on Internet access. It was further enhanced in 2021 with the introduction of AirTag, a standalone tracker designed for personal belongings such as backpacks and wallets.

Apple underscores the privacy and anonymity aspects of the Find My service [8], which is confirmed in recent studies [31,53]. First, lost messages are not authenticated by finder devices, nor are the devices sending them. Second, end-to-end encryption is applied to location reports. Specifically, each location report is encrypted using the advertised public key before being uploaded to the Apple Cloud, ensuring that it can only be decrypted with the corresponding private key. Third, a location report, along with the SHA256 hashed public key, is uploaded to the cloud. Anyone can query and download location reports using a hashed public key.

2.3 Details of Lost Messages

As illustrated in Table 1, an advertisement for a lost message comprises device-specific and constant fields.

Device-specific fields. The lost message requires 37 bytes [5], including a 28-byte public key. However, since a BLE advertisement can accommodate only up to 31 bytes [15], this results in a 6-byte deficit. To address this, Apple splits the public key into three parts and stores them as follows:

Table 1: Dissection of a BLE advertisement for a lost message. "pub" stands for the public key.

Type	Field	Value
Device specific	Adv. Address (PubKey part 1)	pub[0:6]
Constant	Adv. Payload Length	0x1E
Constant	Adv. Type	0xFF
Constant	Company ID	0x004C
Constant	Apple Payload Type	0x12
Constant	Apple Payload Length	0x19
Device specific	Status	Example: 0x00
Device specific	PubKey part 3	pub[6:28]
Device specific	PubKey part 2	pub[0] » 6
Device specific	Hint	Example: 0x00

- *Part 1*: The first six bytes of the public key are placed in the advertising address. However, Apple overwrites the two most significant bits of the address with *0b11*, indicating a random static address (Section 2.1).
- *Part 2*: The overwritten two bits of the public key are relocated here.
- *Part 3*: The remaining 22 bytes of the public key are stored in this part.

The *Status* byte, defined by the Find My specification, varies based on the device type (e.g., iPhone, Mac, AirTag) and battery level. Section 7.2 will examine how specific values (e.g., *0x00*) can be leveraged to evade detection of tracking.

Constant fields. Unlike device-specific fields, several fields remain constant, including the Advertisement Payload Length, Advertisement Type, Company ID, Apple Payload Type, and Apple Payload Length.

It is worth noting that the lost message does not include any fields for authenticating whether the message is sent by an Apple-certified device. As a result, non-Apple devices can also send lost messages and be tracked successfully, as demonstrated in prior work [31].

3 Limitations of Prior Methods

Prior State of the Art. `OpenHayStack` conducts a study of the Find My service through reverse engineering and interpretation of the Find My specification. Notably, it open-sources the attack that turns a computer into a tracker. However, several limitations of `OpenHayStack` remain unaddressed, constraining its applicability.

Address type. `OpenHayStack` concludes that the advertising address should be a random static address, as specified in the Find My specification [5]. Subsequent studies [27,43,53,69] share this consensus. However, our study reveals that finders accept a wide range of address types,

including public address, resolvable private address, and non-resolvable private address. This is crucial to the success of our attacks.

Hardware dependence. Through source code examination, we discovered that the implementation of `OpenHayStack` relies heavily on "hcitool" and Broadcom-specific commands, which are not standardized BLE operations and are specific to certain hardware and platforms. This hardware dependency prevents `OpenHayStack` from being deployed across a wide range of BLE devices.

Privilege dependence. `OpenHayStack` first generates a public/private key pair and then modifies the BLE advertising address to match the public key. However, modifying the advertising address requires root privileges or a device without privilege control mechanisms (e.g., ESP32), posing significant constraints on attack scenarios.

Other Approaches. Another approach to tracking a device involves determining its location based on its IP address. An ISP (Internet Service Provider) assigns the IP address to a specific customer account, and they maintain records that map the assigned IP address to the customer's billing information, including their home address. However, this detailed information is not publicly accessible and requires lawful means, such as a subpoena or court order, to be accessed.

Some companies maintain specialized geolocation databases that map IP addresses to approximate physical locations [22, 34, 42]. However, the accuracy of these databases is generally limited, typically providing location estimates at the city or neighborhood level [49]. Furthermore, many Internet Service Providers (ISPs) assign dynamic IP addresses, which can change over time, reducing the reliability of location tracking. This limitation is further exacerbated by the widespread use of Network Address Translation (NAT), a technique that allows multiple devices to share a single public IP address. NAT is commonly employed in academic institutions and corporate environments, where large groups of users egress their traffic from a shared IP pool. Richter et al. [50] note that the depletion of the IPv4 address pool has led approximately 40% of ISPs to deploy Carrier-Grade NAT, with user-to-IP ratios as high as 20:1. As a result, multiple users may share the same public IP address, making it impractical to precisely determine a target's location using IP-based methods.

Additionally, the proliferation of Virtual Private Networks (VPNs) further complicates IP-based location tracking. They are commonly used by international businesses and organizations for various purposes, such as circumventing censorship, reducing latency, and enhancing Quality of Service (QoS). In this technological landscape, a location inferred from an IP address may not accurately reflect the device's actual physical location. For example, a device physically located in Singapore may appear to have a Hong Kong IP address.

Two alternative methods for tracking users are worth consideration: (1) obtaining the MAC addresses of nearby Wi-Fi access points (APs), and leveraging a MAC-location database to infer the device's approximate location, and (2) employing BLE advertising to broadcast a unique message, which is detected by BLE receivers at specific locations. While these methods are technically feasible, they face notable limitations in practice.

MAC-based tracking relies on the accuracy and coverage of the database, which may be incomplete or outdated [32,37,39]. Additionally, many modern devices use MAC address randomization [3,24,37,60], which significantly limits the effectiveness of this approach. BLE-based tracking, while precise in controlled environments, requires the deployment of multiple BLE receivers in specific locations, which is logistically challenging and costly.

In comparison, our proposed attack leverages Apple's Find My network, which benefits from global coverage through over a billion active Apple devices acting as finders. This approach bypasses the challenges associated with MAC address randomization, database accuracy, and the need for additional infrastructure deployment, providing a robust, cost-effective, scalable and precise solution for tracking.

4 System Overview and Threat Model

4.1 System Overview

Our design for `nRootTag` aims to achieve the following goals.

- **Generalizability.** In contrast to prior work, the attack method should be applicable to a wide range of devices and attack scenarios without being constrained to specific chip architectures or requiring root privileges.
- **Time Efficiency.** An infected computer can be located rapidly.
- **Cost Efficiency.** The attack should be affordable even for individual attackers. Ideally, the computational cost remains largely unaffected by an increase in the number of tracked computers.
- **Stealth.** The attack should avoid raising immediate suspicion from the owner of the tracked computer.

Insights. Although the Find My specification and prior research specify that a random static address is required for advertising, our experiments reveal that the Find My implementation accepts a wide range of advertising addresses, including public addresses, NRPAs, RPAs, and random static addresses (Figure 3). Actually, public addresses are widely used for advertising by Linux systems, and NRPAs and RPAs are used by Windows and Android, respectively. Rather than modifying the advertising address, we aim to explore the feasibility of searching for a public/private key pair that matches the address, thereby eliminating the need for root privileges.

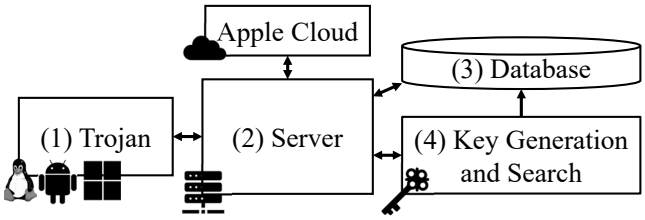


Figure 4: Architecture of `nRootTag`.

Approaches. We propose two attack approaches tailored to different operating systems. The operating system is identified on the target device, enabling the automatic selection of the appropriate attack approach. ***Attack-I:*** On operating systems, such as Linux, a *public* address is used for advertising and can be accessed without root privileges. The public address of a BLE chip includes a 24-bit OUI (Organizationally Unique Identifier) assigned by the IEEE, and these registry records are publicly accessible. By leveraging this public information, we can precompute rainbow tables that store keys for various public addresses. As a result, given a public address, the corresponding public/private key pair can be retrieved instantly. ***Attack-II:*** On other systems, a random *resolvable* or *non-resolvable* address is typically used for advertising, and the address cannot be accessed without root privileges. We consider an attack scenario, for example, where a popular app is installed on multiple computers at a location, such as a home, mall, or office building. Thus, the app, aiming to track users, can use one computer to sniff advertisements from another and obtain the advertising address. Subsequently, the app server can use a rainbow table lookup or online key search to find a matching public/private key pair. To accelerate the key search, we have implemented a search system that takes advantage of multiple GPUs.

Architecture. Figure 4 illustrates the architecture of our design. (1) The **Trojan** code runs on the computer to be tracked. It retrieves the advertising address, acquires the matching public key from our server, and then advertises lost messages. (2) The **Server** processes requests for acquiring public keys through rainbow table lookup or online key search. (3) The **Database** system, which contains a rainbow table of key information, provides the corresponding public/private key pair for a given advertising address. (4) The **Key Generation and Search** serves two purposes: it is used to precompute the rainbow table, and invoked to search for a matching public/private key pair on the fly. Finally, given a public key, the server uses its hash value to query the Apple Cloud for location reports, then decrypts the reports using the private key.

Attack Procedure. (i) Once the Trojan is deployed on a computer, it contacts the Server to receive a unique ID. (ii) The Trojan retrieves the BLE advertising address and sends a query to the Server, including its unique ID and the advertising address. (iii) The Server searches for a matching public/private key pair and returns the public key to the Trojan. The Server also updates its records to associate the Trojan's ID with the key pair. (iv) The Trojan advertises lost messages containing the public key. Any finder receiving these messages uploads an encrypted location report, along with the hash of the public key, to Apple Cloud. (v) The Server uses the hash of the public key to query Apple Cloud for location reports and decrypts them using the corresponding private key.

4.2 Threat Model

This work introduces a novel attack that allows an attacker to turn a computer into an "AirTag" tracker without the need for root privileges. The threat model considers various potential adversaries and their capabilities, as well as the assumptions made regarding the computers to be tracked.

The following adversaries may be interested in exploiting this attack.

- **Apps** for shopping, streaming, social media, and other personalized services could exploit this attack to track users' locations, enabling them to deliver tailored recommendations or targeted advertisements. These apps typically seek to enhance user experience or boost ad revenue by monitoring users' preferences, behaviors, and locations. Applications that rely on Bluetooth can easily legitimize their use of Bluetooth while concealing malicious objectives. Notable examples include payment applications that use Bluetooth for seamless transactions, as well as music applications that employ Bluetooth to connect with earphones.

- **Intelligence departments** may utilize this attack for espionage and surveillance. They can use social engineering (such as phishing and fake software updates), vulnerability exploitation (e.g., drive-by downloads), and software supply chain attacks (e.g., adding the Trojan code to a library) to deploy our attack.

- **Individuals** or organizations with malicious intent, such as botnet owners, can use this attack to track users. They can exploit social engineering, software vulnerabilities, or malicious ads to distribute our Trojan logic.

We assume the computer to be tracked is equipped with a functional BLE module and is running our Trojan code, which can communicate with our server via the Internet and can both send and sniff BLE advertisements.

While a single Trojan instance suffices to launch Attack-I, Attack-II requires at least one Trojan instance to be within the BLE range of another to obtain the advertising address. In other words, two Trojan instances within BLE range are sufficient to execute Attack-II. The range of BLE 5.0 is up to 400m [55].

Potentially harmful mobile apps are known to collect user data [19, 36, 40, 71] and may exploit our attack to track users, enabling personalized recommendations or increasing ad revenue. In practice, it is common for users to install the same app on multiple devices, such as smartphones, desktops, or laptops at home or in the office. For example, if a person is carrying a smartphone and wearing a smartwatch, Attack-II can be used to track them, provided both devices are running our Trojan code. Similarly, a potentially harmful mobile app may be installed on multiple nearby devices in shared spaces like hospitals, or shopping malls, providing an avenue to launch Attack-II.

As another example, large botnets have infected tens of millions of devices [4, 11]. It is not uncommon for multiple IoT devices in a home, office, or factory to be infected by the same botnet. Botnet operators could exploit our attack to locate these bots and execute targeted phishing, blackmail, or extortion campaigns.

On Android, user permission is required to perform Bluetooth operations. However, given the widespread use of Bluetooth peripherals and applications, such as file sharing, gaming, and smart homes, Bluetooth permissions are frequently granted to apps [54]. Bluetooth permissions are widely considered less sensitive than GPS location permissions [64].

Root privileges or a GPS module on the computer are *not* required for our attack. Additionally, the computer being tracked does not need to be registered on Apple Cloud, as finders do not verify whether a lost message originates from an Apple device or is registered on Apple Cloud [31]. Furthermore, the computer does not need to be in close proximity to the Server during preparation, as it receives the public key over the Internet.

5 System Design

To make the discussion concrete, we use Linux, Windows, and Android as examples: Attack-I applies to Linux, while Attack-II applies to Windows and Android. Regarding the attack scenario, as an example, we consider a popular app that intends to track its users.

5.1 Trojan

The Trojan enables Bluetooth (if not already enabled), obtains the BLE advertising address, queries the server for the matching public key, and then sends lost messages.

Enabling Bluetooth. Bluetooth is often already enabled on computers due to the widespread use of Bluetooth peripherals (e.g., keyboards, mice, earbuds, speakers), allowing this step to be skipped. If not, the Trojan must first enable Bluetooth. ***Attack-I:*** On Linux, prior work [68] observed that Gnome-based systems might implement special rules granting any program access to `/dev/rfkill`, an interface used to communicate with the Linux kernel to control the adapter's power state. However, among the 12 distributions we tested, write permissions for `/dev/rfkill` are generally restricted to root. Notably, we discovered a rule file (`61-gnome-settings-daemon-rfkill.rules`) that creates an exception, allowing non-root users to write to `/dev/rfkill`. This enables the Trojan to enable Bluetooth adapters without requiring user interaction or root privileges. ***Attack-II:*** Bluetooth on Windows and Android can be enabled using the `SetStateAsync` and `BluetoothAdapter.enable` APIs, respectively. However, starting with Android 13, explicit user permission is required to enable Bluetooth.

Retrieving Advertising Address. Our approach is to search for the public key that matches the advertising address, eliminating the need for root privileges. Thus, it is critical to get the advertising address. ***Attack-I:*** On Linux, retrieving the address is straightforward and can be done through the standard `bluez` interface. ***Attack-II:*** Android uses RPAs for advertising, while Windows uses NRPAs. Both systems change the advertising address periodically (e.g., every 10 minutes) and prevent non-root programs from reading the address. We propose leveraging nearby devices to retrieve the address. In practice, it is common for a user to install an app on multiple devices, such as a smartphone, desktop, and laptop at home. Similarly, the app may be installed on devices located in a shared space, like a building, shopping mall, or on a train or bus. In this scenario, Trojan *A* periodically (once per minute) advertises an *address-query* message containing its 64-bit ID assigned by the Server and a 32-bit sequence number *N*. Trojan *B* on a nearby device that sniffs the message can extract *A*'s advertising address from the message and respond with an *address-response* message that includes *A*'s ID, address, and the sequence number *N*. Similarly, Trojan *B* retrieves its own advertising address from Trojan *A*. The address-response message is sent with a random delay (ranging from 0 to 2000 ms). If an address-query message has already triggered a response, nearby Trojans refrain from responding to avoid redundant replies.

Obtaining Public Key. Using the obtained advertising address, the Trojan queries the Server for the matching public key. ***Attack-I***: Since a device's public address remains constant, the key query is a one-time operation. Once the Server responds with the matching public key, it creates a record mapping the Trojan's ID to the public/private key pair. ***Attack-II***: If the Trojan obtains its advertising address for the first time or detects a change in the address, it queries the Server for the public key. The Server then updates the record mapping the Trojan's ID to the public/private key pair.

Sending Lost Message. Once the matching public key is obtained, the Trojan crafts and sends the lost message (Section 2.3). On Android, we observed that the advertising address changes when the advertising payload is modified by default. However, by using the `AdvertisingSet` API available since Android 8, we ensure that the advertising ad-

dress remains unchanged despite payload modifications (from address-request to lost message).

5.2 Server

When a Trojan contacts the Server for the first time, the Server assigns a unique ID to the Trojan. Upon receiving a public-key query from a Trojan, the Server either searches the rainbow table or performs a real-time GPU-based key search, as detailed in Sections 5.4.1 and 5.4.2.

Once a key is found, the Server updates the record mapping the Trojan ID to the corresponding public/private key pair. To locate a Trojan, the Server uses the SHA256 hash of the public key to query the Apple Cloud for location reports and decrypts them using the corresponding private key. Notably, as demonstrated by the community project [14], attackers can create virtual MacBooks to download location reports. This approach enables attackers to access location reports without relying on a physical MacBook, which could be traced via its serial number, thereby maintaining anonymity. Technical details are elaborated in Appendix A.1.

5.3 Database

The database stores the rainbow table. A straightforward design is to use a regular database system, where the advertising address (6 bytes) serves as the *primary key* for data indexing, and the other table columns include the matching public key (28 bytes) and private key (28 bytes). With this design, each record requires 62 bytes.

Taking Attack-I as an example (with similar optimizations for Attack-II), we use a custom database system with several optimizations, as illustrated in Figure 5a. (1) **Public key elimination**: The public key can be derived from the private key [44, 65], making it unnecessary to store the public key. (2) **OUI-based organization**: Advertising addresses belonging to the same OUI result in repetitive OUI storage in the records. Instead of storing all the records together, we store the records for each OUI in a separate file, using the OUI as the file name. This eliminates the need to store the 24-bit OUI in the records. (Similarly, in Attack-II, the first 22 bits of the critical part of the address serve as the file name.) (3) **Implicit indexing**: In addition to the OUI, a public address contains 24 manufacturer-specific bits (Figure 3). Consequently, each file corresponding to an OUI contains 2^{24} records. These records are organized as elements in an array, where the manufacturer-specific bits of a public address are used as the array index for storing or retrieving the private key. This design eliminates the need to store the remaining 24 bits of the address, reducing storage requirements to just 28 bytes per record.

For Attack-I, we maintain a record in the rainbow table for each unique public address of an OUI. Thus, each file for an OUI contains 2^{24} records, requiring $2^{24} \times 28$ bytes, or 448 MB. According to IEEE registry records, there are 47,054 OUIs [46]. Consequently, the rainbow table for Attack-I requires approximately 20.10 TB of storage. A 24 TB hard drive costs around 479.99 USD [2], making this storage requirement affordable even for individual attackers.

For Attack-II, a slightly different design is used to save storage. Since only the least significant 46 bits of an address are used for the key match, we maintain 2^{46} records in the rainbow table for Attack-II, one for each unique value of the 46 bits, as shown in Figure 5b. If two devices randomly generate addresses differing only in the most significant two bits, the server returns the same public key for the two devices. However, given the $N = 2^{46}$ value space, this collision is highly unlikely, depending on the number of tracked computers,[2] and the concern is further mitigated because non-resolvable or resolvable random addresses change periodically. Thus, the rainbow table for Attack-II requires approximately $2^{46} \times 28$ bytes, or 1.75 PB of storage. While this is prohibitively expensive for individual attackers (the cost is analyzed in Section 6.2.2), it may be affordable for app owners or a country's intelligence department. A rainbow table is a more cost-effective choice for attackers aiming to track users continuously over long periods. However, for short-term or intermittent tracking, online key search may be a viable alternative (Section 5.4.2).

5.4 Key Generation and Search

We design the public/private key pair generation module to utilize multiple GPUs, enabling accelerated construction of the rainbow tables and key search.

5.4.1 Building Rainbow Table

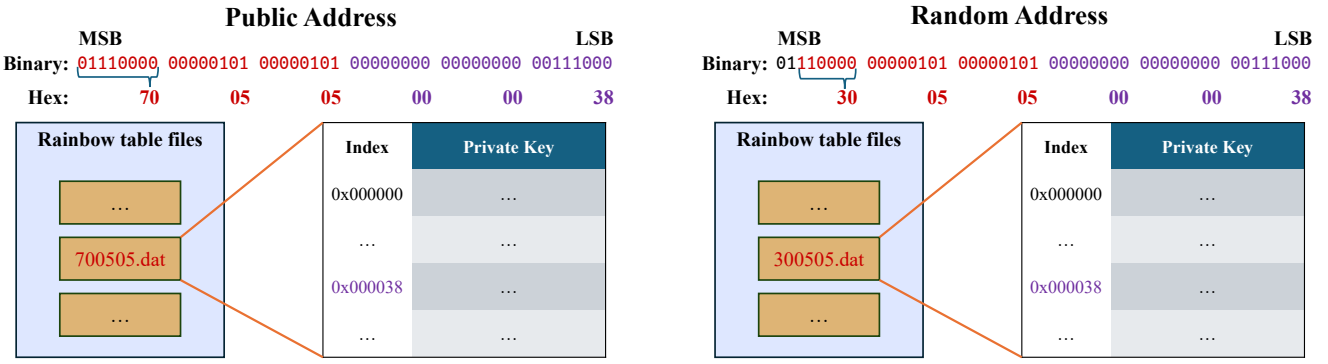
To build the rainbow table for Attack-I, we need to find a matching public/private key pair for each unique public address of a registered OUI. However, the private-to-public key derivation is a one-way process, making the generated public keys essentially random. In other words, the computation cannot be directed to generate keys exclusively for the OUIs published by IEEE. Given that a public address consists of 48 bits, the random search space is effectively $N = 2^{48}$. Each generated public/private key pair matches one of the 2^{48} possible addresses. This scenario can be modeled as the classic *coupon collector's problem*: given N coupons, how many coupons are expected to be drawn with replacement before having drawn each coupon at least once [23].

The expected number of trials required to collect all N distinct coupons is:

$$E[T] = N \cdot H_N$$

where H_N is the N-th harmonic number:

[2]The collision probability follows the birthday paradox. For instance, the probability of at least one collision is only 10% when the number of tracked computers exceeds 2.7 million.



(a) Rainbow table for Attack-I. **(b) Rainbow table for Attack-II.**

Figure 5: Storage of rainbow tables. In Attack-I, for instance, a rainbow table file is named based on an OUI, with the remaining bits of the public address used as an index to store and retrieve the private key, while the public key can be derived from a private key.

$$H_N = \sum_{k=1}^{N} \frac{1}{k}$$

For large N, an approximation is:

$$H_N \approx \ln(N) + \gamma + \frac{1}{2N}$$

Thus, the expected number of trials is approximately:

$$E[T] \approx N \cdot \left(\ln(N) + \gamma + \frac{1}{2N} \right)$$

where $\gamma \approx 0.577$ is the Euler-Mascheroni constant. For Attack-I, for example, we substitute $N = 2^{48}$ into the equation. First, calculate:

$$\ln(2^{48}) = 48 \cdot \ln(2) \approx 48 \cdot 0.693 \approx 33.264$$

Now, using the approximation:

$$E[T] \approx 2^{48} \cdot \left(33.264 + 0.577 + \frac{1}{2 \cdot 2^{48}} \right)$$

Since $\frac{1}{2 \cdot 2^{48}}$ is extremely small, it can be neglected for practical purposes. Thus, the expected number of trials is approximately:

$$E[T] \approx 2^{48} \cdot (33.841)$$

In short, to build the rainbow table for ***Attack-I***, where $N = 2^{48}$, the expected number of trials is approximately $E[T] \approx 2^{48} \cdot (33.841)$. With the GPU RTX 3080, for example, our code can generate 4.90 billion public/private key pairs per second. Thus, using one RTX 3080 GPU, the expected time for the key generation is approximately 539.99 hours. When 200 such GPUs are used, the time is reduced to 2.70 hours.

To build the rainbow table for ***Attack-II***, where $N = 2^{46}$, the expected number of trials is approximately $E[T] \approx 2^{46} \cdot (32.455)$. Using one RTX 3080 GPU, the expected time for the key generation is approximately 129.47 hours.When 200 such GPUs are used, the time is reduced to 0.65 hour.

It is important to clarify that when on-cloud GPUs are rented for key generation, the computation is billed based on the total active usage time. Therefore, the cost does not increase with the number of rented GPUs [33, 51, 66]. The cost is discussed in Section 6.2.

5.4.2 Online Key Search

Given a BLE address, online key search continues generating keys until a public key matches it. As described in Section 5.3, for short-term or intermittent tracking with Attack-II, online key search may be a more cost-effective option.

Given a BLE address and the search space $N = 2^{46}$ for Attack-II, the probability of match success (i.e., the generated key matches the address) in a single key generation is $p = \frac{1}{N}$. Therefore, with k keys generated, the probability q of at least one match success is:

$$q = 1 - (1-p)^k$$

Given q, we can solve for k as follows:

$$k = \frac{\ln(1-q)}{\ln(1-p)}$$

The number of keys generated per second is denoted as KPS, and the time spent on key generation measured in seconds is t. Since $k = KPS \times t$, we can solve for t:

$$t = \frac{\ln(1-q)}{KPS \cdot \ln(1-p)} \quad (1)$$

According to Equation 1, for example, when 200 RTX 3080 GPUs are rented for online key search, given a BLE address and a targeted match success rate $q = 0.9$, we can calculate $t =$

165.34 seconds (2.76 minutes), which costs 2.20 USD. When tracking multiple m computers, we organize their addresses in a hash table based on the value of each address's critical part. For each public key generated, its critical part is used to search the hash table. Note the number of generated keys k required to attain the probability q remains the same for each of the m addresses, as the success probability for each address is independent of others. Therefore, the cost remains constant regardless of the number of tracked computers. Further cost details are provided in Section 6.2.

We have implemented key search functionality for the secp224r1 curve, utilized by Apple's Find My network. A notable optimization involves creating a generator table that precomputes numerous intermediate scalar multiplication results. Subsequent computations perform scalar additions with these cached results, significantly reducing the computational time associated with expensive scalar multiplications. Crucially, this approach leverages GPU acceleration, yielding performance improvements of several orders of magnitude, thereby rendering our proposed method feasible.

The performance of the key search process is influenced by several factors. To identify the optimal settings, we conduct comprehensive experiments across various GPUs and configurations, as detailed in Section 6.1.

6 Evaluation

We first study the impact of GPUs and configurations on key generation speed (Section 6.1), and then investigate the cost efficiency (Section 6.2). The latency and accuracy of the attack are detailed in Section 6.3, with its applicability discussed in Section 6.4.

6.1 GPUs and Configurations

We begin by examining the impact of various GPUs and configurations on key generation speed. Our study includes several consumer-grade GPUs, such as the RTX 3070, RTX 3080, and RTX 4090, as well as data center-grade GPUs, specifically the A100 (80GB) and H100 (80GB).

A Streaming Multiprocessor (SM) is a core computational unit in NVIDIA GPUs, equipped with CUDA cores, warp schedulers, and shared memory to handle thousands of threads concurrently. Proper selection of X, the number of blocks per SM, and Y, the number of threads per block, is crucial for performance. Excessively large X can cause scheduling overhead, while very high Y may exhaust shared memory and registers, reducing active warps and GPU occupancy. Conversely, overly small X or Y leads to underutilization of the resources. Balancing these parameters ensures efficient operation and maximizes performance for key generation tasks.

We conducted experiments to examine the impact of these parameters on key generation speed, measured in Keys per Second (KPS). As shown in Figure 6, the configuration $X = 32$ and $Y = 512$ achieves optimal performance across GPU models. Increasing these parameter values beyond this point does not yield further improvements in key generation speed.

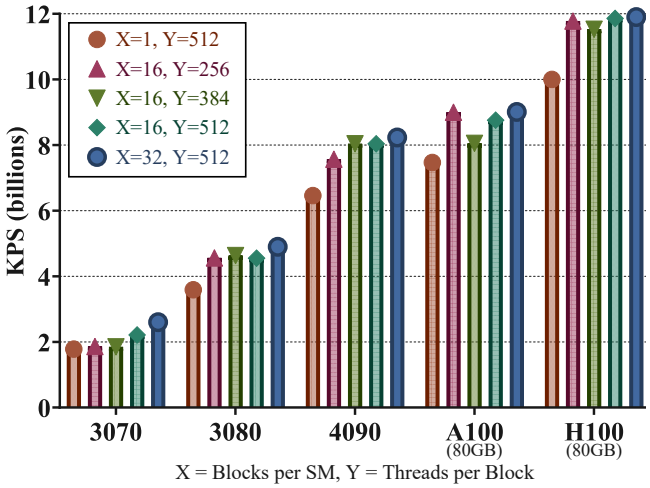


Figure 6: The impact of GPUs and configurations on key generation speed.

Table 2: GPU selection. The optimal configuration $X = 32$ and $Y = 512$ is used for each GPU.

Model	KPS (billion)	Price (USD)	Price-to-KPS Ratio	Rent (USD/h)	Rent-to-KPS Ratio
RTX3070	2.64	499	189	0.13	0.050
RTX3080	4.90	699	**143**	0.22	**0.045**
RTX4090	8.22	1,599	195	0.40	0.049
A100 80GB	9.07	13,224	1458	1.60	0.178
H100 80GB	11.91	48,200	4047	2.14	0.180

While the H100 achieves the highest key generation speed, its cost is prohibitively high. Therefore, we use the price-to-KPS ratio (lower is better) to guide GPU selection. For scenarios where cloud GPUs are rented, we consider the rent-to-KPS ratio (lower is better). Many cloud GPU platforms, such as `vast.ai`, charge based on active usage time; to make the following discussion concrete, we use its listed rental rates. As shown in Table 2, the RTX 3080 offers both the lowest price-to-KPS ratio and the lowest rent-to-KPS ratio. Thus, we recommend the RTX 3080 for the key generation.

6.2 Cost Efficiency

As discussed in Section 4.1, the rainbow table for Attack-I requires only 20.10 TB of storage, making it a practical choice for this attack. However, for Attack-II, the rainbow table demands 1.75 PB of storage, which is prohibitively expensive. Therefore, for Attack-II, we compare the costs of the two key search methods: rainbow table-based search and online key search.

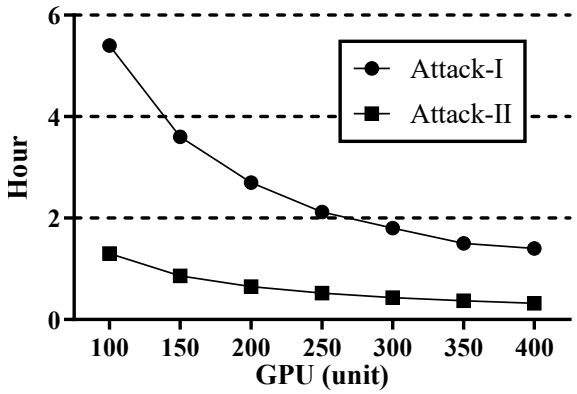


Figure 7: The time needed for building the rainbow table decreases as the number of used GPUs increases.

6.2.1 Attack-I

Building the rainbow table is a one-time effort, making it cost-effective to rent cloud GPUs for this purpose. As discussed in Section 5.4.1, the expected number of key generations is $E[T] \approx 2^{48} \cdot (33.841)$. One RTX 3080 generates 4.90 billion keys per second (Table 2). Notably, on cloud GPU platforms that charge based on active usage time, the cost remains constant regardless of the number of GPUs rented, as long as the total usage time is unchanged. Therefore, we propose renting multiple GPUs to expedite the process. Figure 7 shows that the required time decreases as the number of rented GPUs increases. For instance, using 200 RTX 3080 GPUs, the process takes 2.70 hours. At a rate of 0.22 USD per GPU per hour, the total cost to create a rainbow table for Attack-I is $118.80 (= 0.22 \times 200 \times 2.7)$ USD.

As discussed in Section 5.3, the rainbow table for Attack-I requires a storage of 20.10 TB, which can be satisfied with a standard 24 TB hard drive, priced at 479.99 USD [1]. These resources are affordable even for individual attackers.

6.2.2 Attack-II

Rainbow table. As discussed in Section 5.4.1, building the rainbow table for Attack-II requires an expected number of key generations of $E[T] \approx 2^{46} \cdot (32.455)$. Accordingly, we can plot Figure 7, which illustrates how the required time varies with the number of rented GPUs. For example, using 200 RTX 3080 GPUs in parallel completes the task in 0.65 hours at a total cost of $28.60 (= 0.22 \times 200 \times 0.65)$ USD.

As presented in Section 5.3, the rainbow table for Attack-II demands a substantial storage of 1.75 PB (or 1,792.0 TB). As illustrated in Table 3, for on-premises storage, a dedicated server with 75 drives (24 TB each) necessitates an initial investment of 65,601.85 USD, coupled with monthly operational costs of 1,484.52 USD.

Cloud storage offers an alternative solution that eliminates hardware purchases but incurs a monthly cost of 18,350.00 USD, based on current market rates of 10.24 USD/TB [9, 25, 52]. Despite the significant upfront capital required for the dedicated server, our analysis demonstrates that the on-premise storage becomes more economical than cloud storage after 3.89 months.

Table 3: Storage system setup and monthly operation costs for maintaining the Attack-II rainbow table.

Category	Details	Cost (USD)
One Time Setup Expense		
Rainbow Table	Rent 200 RTX 3080 for 0.65h	28.60
Hardware	Supermicro Storage Server	29,574.00
Hardware	479.99 USD Per Drive × 75	35,999.25
	Total One Time	**65,601.85**
Operation Expense (Monthly)		
Electricity	1.3 kW × 30 days × 0.177 USD/kWh	165.67
Cooling	0.2 kW × 30 days × 0.177 USD/kWh	25.49
Depreciation	Five year depreciation model	1,093.36
Maintenance	Estimated Repairs and personnel	100.00
Network	Estimated for small business	100.00
	Total Monthly	**1,484.52**

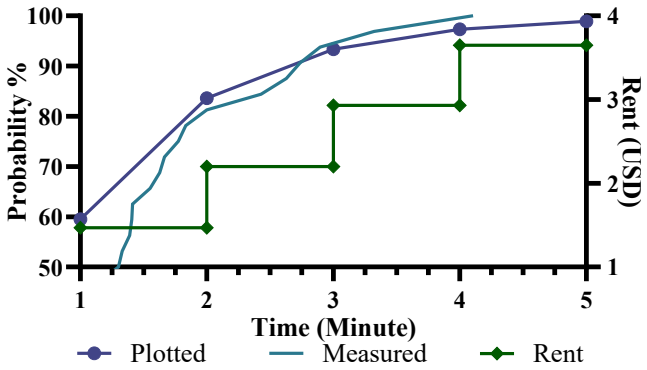


Figure 8: The success rate and cost of online key search change over time.

In summary, for rainbow table-based key searches, on-premise storage is a more cost-effective option compared to renting cloud storage. However, it may still be prohibitively expensive for most individual attackers.

Online key search. For short-term or intermittent tracking (e.g., once per hour), we find that online key search is more cost-effective. According to Equation 1, given 200 RTX 3080 GPUs rented, we can generate Figure 8, which illustrates how the probability of key search success and the associated cost increase with search time. For instance, achieving a 90% success probability requires a search time of 2.76 minutes and incurs a cost of $2.20 (= 200 \times 3 \div 60 \times 0.22)$ USD. Note that for 2.76 minutes, the cloud GPU platform charges for 3 minutes, as it rounds up to the nearest whole minute. If

Table 4: Confidence levels correlate with deviations.

Confidence	Deviation
3	27m
2	56m
1	264m

users are tracked once per hour, the daily cost amounts to 52.8($= 2.2 \times 24$) USD. If attackers limit tracking to once per hour during daytime on weekends (e.g., 7AM to 6PM), the *weekly* cost is 52.8($= 2.2 \times 12 \times 2$) USD.

By renting 200 RTX 3080 GPUs, we also conducted measurements with a group of random addresses to observe how the key search success rate evolves over search time. The results show that the measured curve closely aligns with the theoretical curve plotted.

In summary, online key search is affordable even for individual attackers, and we recommend online key search for launching Attack-II. As explained in Section 5.4.2, the online key search cost does increase as the number of tracked computers grows.

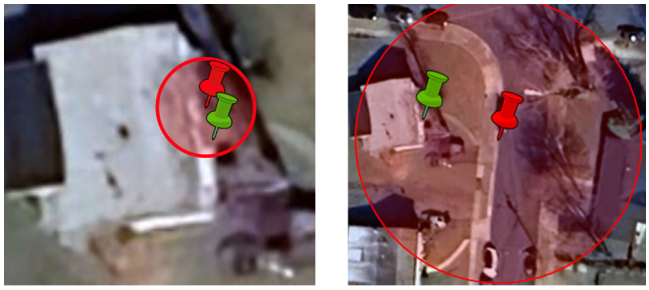
6.3 Latency and Accuracy

We conducted tracking in stationary, riding, and flight scenarios. For the stationary scenarios, we used a desktop computer running our Trojan. In the e-bike riding, a Raspberry Pi running our Trojan was carried, and the ground truth trajectory data was collected using the iOS app "GPS Tracks." In the flight scenario, we carried a Steam Deck, a popular portable gaming console, and the ground truth for the flight trajectory was collected from "Flight Radar 24." In the flight scenario, other passengers (i.e., Finders) may have paid to use in-cabin Wi-Fi and caused submitting location reports to iCloud.

The location report contains an attribute *Confidence*. While Apple has not disclosed the meaning of this attribute, our measurements show that it correlates with the deviation between the actual location of the Finder and its reported location, as illustrated in Table 4. Reports with confidence levels 2 and 3 demonstrate low deviation, while level 1 reports show significant deviation and were excluded from our analysis.

6.3.1 Latency Measurement

We measured the latency between the first advertisement and the first available location report in two environments: office and residence. For the office environment, measurements were conducted within a university building for 24 hours. We collected 637 samples, yielding an average wait time of 6.75 minutes, with the 90th percentile at 10.34 minutes. The residential environment was a single-family house in a suburban area, where 526 samples were collected in 24 hours, showing a slightly longer average wait time of 8.09 minutes, with the 90th percentile reaching 15.92 minutes. These statistics indicate the expected latency for receiving the first available location report after a lost message is sent.



(a) Eps = 3m **(b) Eps = 25m**

Figure 9: Tracking in a residential environment. The ground-truth location (green pin), estimated position (red pin), and epsilon boundary region (semi-transparent red area) are shown.

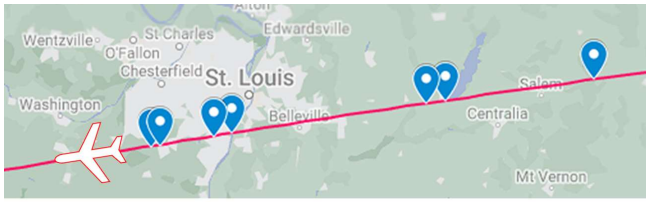


Figure 10: Flight trajectory and Find My location reports.

6.3.2 Accuracy Measurement

Stationary. We first conducted accuracy measurements at the aforementioned single-family house. Given the location reports collected during 24 hours, we performed a location estimation using DBSCAN with ball tree clustering. We observed an interesting phenomenon: the clustering produces accurate estimates when using a small epsilon, as shown in Figure 9(a). Epsilon, in the context of DBSCAN, defines the maximum distance between two points for them to be considered part of the same cluster. While in our experiment the estimates were highly accurate (the green pin and the red pin are within 3m), it is worth clarifying that these represent the actual Finder locations, rather than the locations of the target device. As illustrated in Figure 9(b), a larger epsilon value results in location reports being aggregated over a broader area, leading to a shift in the estimated position.

Riding. The e-bike riding speed was approximately 15 km per hour, and the ride lasted for 57 minutes. As the rider quickly passed potential adjacent Finders, they might not have been able to receive the complete lost message. As a result, only a small number of location reports were collected. Totally, nine location reports were collected, with seven reports aligning

with the actual track and two reports showing deviations of 45.88 m and 58.92 m.

Flight. The flight lasted a total of 91 minutes, with an average speed of 863 km per hour. Figure 10 illustrates the flight trajectory and the locations reported by the 17 location reports. The average distance between the reported locations and the ground truth was 70.03 meters. GPS location on a flight can be less accurate due to signal blockage from the aircraft's structure, high altitude and speed affecting satellite geometry, and limited visibility of satellites. Interestingly, by chaining the reported locations, we were able to reconstruct the flight path. By incorporating real-time flight trackers, such as "Flight Radar 24," we were able to deduce the flight number.

6.4 Attacks on Various Devices and OSes

Our attack was carried out on various devices and operating systems, categorized under Attack-I and Attack-II.

Attack-I: Attack-I is applicable to Linux, which uses a public address for advertising. As detailed in Table 5, we tested the attack on 12 different Linux distributions and achieved success on all of them.

1. **IoT Devices**: In addition to a Gigabyte desktop computer, the attack was also validated on two IoT devices: a Steam Deck, a well-known portable gaming console, and a Raspberry Pi, a popular IoT platform.

2. **Bluetooth State and Stealth Enabling**: On three of the 12 distributions, Bluetooth is disabled by default. However, Bluetooth can be programmatically enabled on all distributions without triggering any alerts. Despite these successes, unique challenges were encountered on openSUSE and Debian for enabling Bluetooth: openSUSE requires root privileges for executing `rfkill`, and Debian lacks the executable. We overcame these obstacles by directly writing the bytestream *0x0000000002030000*[3] to `/dev/rfkill`, successfully enabling Bluetooth on both systems.

Attack-II: Attack-II is applicable to Android and Windows, which use RPAs and NRPAs for advertising, respectively.

As detailed in Table 6, the experiment involved seven Android phones (versions 11 through 14), a Sony smart TV (Bravia A80J), and a Meta VR headset (Quest 2), and all the attacks succeeded. A Raspberry Pi device running our Trojan was placed around 20 meters away from the tested computer, and they were in different rooms. Interestingly, the smart TV and headset prevented users from disabling Bluetooth, as they rely on it for control. On Android 11 and 12, Bluetooth can be enabled programmatically. However, starting with Android 13, if Bluetooth is disabled by the user, an app requires the user's permission to enable it. Despite this, Bluetooth is typically enabled on most smartphones due to the widespread use of Bluetooth accessories, such as earpods, and applications, such as smart homes and health. Thus, this restriction does not pose a significant barrier to the attack. The address rotation period is large enough for launching Attack-II (Section 6.2.2).

Regarding Windows, the experiment involved one Dell laptop and one Gigabyte desktop, both running the latest version of Windows 11, and Attack-II succeeded on both. Windows supports two application frameworks: Win32 (traditional .exe applications) and Universal Windows Platform (UWP) [45].[4] We validated Attack-II on both frameworks, and our investigation shows that the advertising address rotation behavior is independent of the application framework.

7 Discussion

7.1 Applicability to Apple Devices

Apple's products are known for offering restricted access to APIs, permitting only two advertisement types (i.e., LocalName and ServiceUUID). As a result, a Trojan without root privileges cannot advertise the data required for a lost message. To integrate Trojan with macOS, we adapted `nRootTag` to effectively work with Internal Blue [41,62], a methodology and accompanying software designed to exploit the diagnostic interface inherent in certain Broadcom Bluetooth chips. This adaptation facilitated the manipulation of Bluetooth adapters on macOS systems without the need for root-level access privileges. Our findings indicated that this integration operated successfully on macOS versions 10.15 and 11, allowing for the successful tracking of a MacBook, although Apple has mitigated this issue in subsequent macOS versions.

However, a Trojan on an Apple device can advertise its unique ID assigned by our server, while a Trojan on a nearby device (e.g., an Android phone) that detects the ID-advertising message can relay this ID to the server. The detection of the ID-advertising message indicates that the devices are within Bluetooth transmission range. Therefore, by locating the nearby device, the server can infer the location of the Apple device. Alternatively, the nearby Linux/Windows/Android device can advertise its own ID. In this case, an Apple device that receives the ID can query the server to obtain the location associated with that ID.

[3] The source code [38] reveals that the first four bytes represent the device index, where zeros indicate broadcast to all devices, followed by bytes indicating the device type, operation, software blocking state, and hardware blocking state, respectively.

[4] UWP introduces a modern application model with features such as sandboxing, managed distribution through the Microsoft Store, and cross-platform compatibility across Windows devices, including Xbox.

Table 5: Attack-I succeeded on all the 12 Linux distributions we tested. On three of these, Bluetooth is disabled by default. However, Bluetooth can be programmatically enabled on all the distributions without triggering any alerts.

Device Type	Operating System	Version	Default Bluetooth State	Stealth Enabling
Desktop	openSUSE	Leap 15.6	Disabled	Yes
Desktop	Manjaro Linux	24.0.3	Disabled	Yes
Desktop	Kali Linux	2024.2	Disabled	Yes
Desktop	Linux Mint	21.3	Enabled	Yes
Desktop	MX Linux	23.3	Enabled	Yes
Desktop	Debian	12.5.0	Enabled	Yes
Desktop	Fedora	Workstation 40	Enabled	Yes
Desktop	Ubuntu	22.04	Enabled	Yes
Desktop	Garuda Linux	Dr460nized	Enabled	Yes
Desktop	Pop!_OS	22.04	Enabled	Yes
IoT	SteamOS Holo	3.5.19	Enabled	Yes
IoT	Raspberry Pi OS	2024-07-04	Enabled	Yes

Table 6: Attack-II succeeded on all the nine Android devices and two Windows devices.

Device Type	Device Model	Operating System	Version	Address Type	Rotation Period	Default Bluetooth State	Stealth Enabling
Phones	Google Pixel 3a	Android	11	RPA	809s	Enabled	Yes
	Google Pixel 4	Android	11	RPA	459s	Enabled	Yes
	Google Pixel 5	Android	13	RPA	652s	Enabled	No
	Google Pixel 6	Android	14	RPA	684s	Enabled	No
	Sony Xperia 5 II	Android	12	RPA	378s	Enabled	Yes
	Galaxy S23 Ultra	Android	14	RPA	373s	Enabled	No
	Galaxy Xcover Pro	Android	13	RPA	No Rotation	Enabled	No
Smart TV	Sony Bravia A80J	Android TV	10	RPA	378s	Always Enabled	Not Required
VR Headset	Meta Quest 2	Android	12	RPA	3593s	Always Enabled	Not Required
Laptop	Dell Latitude E5470	Windows	11	NRPA	707s	Enabled	Yes
Desktop	Realtek RTL8761B	Windows	11	NRPA	243s	Enabled	Yes

7.2 Stealth Tracking

Apple has implemented unwanted tracking detection to warn individuals about the potential misuse of tracking devices. Previous research [27, 43, 53] examined this mechanism and found that an alert may be triggered after a tracker travels with an individual for a distance ranging from half a mile [27] to a mile [53].

Stationary devices under our attack, such as desktop computers and household IoT devices, do not move and therefore do not trigger such alerts. In contrast, portable devices (e.g., laptops, gaming consoles) that continuously advertise lost messages can travel with individuals and, without evasion techniques, may activate these alerts.

Previous work by Mayberry et al. [43] describes multiple evasion techniques to avoid such alerts. For instance, setting the *Status* field of a lost message to *0x00* can prevent tracking alerts, as such messages are typically sent by MacBooks, which Apple deems "irrelevant" for tracking purposes. We validated this evasion technique on the latest version of iOS for iPhone, confirming that no alerts were triggered.

7.3 Mitigation

The Find My network specification mandates the use of random static addresses for advertising lost messages. However, our research reveals that public addresses, resolvable private addresses, and non-resolvable private addresses can also serve this purpose without any issues. This implementation vulnerability is exploited by our attack to track Linux, Android, and Windows systems.

To mitigate the attack, Apple could patch finder devices to only process lost messages sent from random static addresses. However, given the vast number of finder devices (e.g., iPhones and Apple Watches), it may take considerable time for the patch to be widely installed.

Even after the patch is broadly adopted, if an attacker identifies devices using random static addresses for advertising, such as some smart home and health devices [16, 35], our attack remains feasible on these devices, leveraging over a billion active iPhones and other Apple devices as spies for tracking. Our future work will investigate this further to assess the broader implications of the proposed attack.

8 Related Work

A series of work has studied the security and privacy implications of Apple's wireless ecosystem. For instance, prior work [10] demonstrated methods to spoof an AirDrop receiver's identity, enabling unauthorized access to personal data. Stute et al. [58, 59] illustrated the potential for user tracking via Apple Wireless Direct Link (AWDL). Celosia and Cunche [17] presented novel techniques for tracking Bluetooth Low Energy (BLE) devices, such as Apple AirPods, and extracting user email addresses and phone numbers from BLE advertisements associated with Apple's Wi-Fi Password Sharing (PWS). Further research revealed design flaws in AirDrop that exposed user phone numbers and email addresses [28]. Additionally, Stute et al. [57] analyzed the protocols involved in Apple's Handoff feature, identifying vulnerabilities that enable device tracking through Handoff advertisements.

Among the works on exploiting BLE for tracking [12, 31, 35, 70], `OpenHayStack` [31] is the most closely related to our work. `OpenHayStack` has reported that one can turn a computer into a tracker leveraging Apple's Find My network. However, to launch the attack, `OpenHayStack` requires root privileges or a device that does not distinguish root and non-root processes at all. Our work can turn a Linux, Android or Windows system into a tracker without root privileges. It exploits a vulnerability of the Find My network we discovered. We also developed key search techniques to make the attack cost-effective and fast.

Recent work [13, 26, 63] has examined the potential utilization of Find My network for transmitting arbitrary data, exploring applications such as crowd-sourced sensing and data-muling.

In a broader context, Weller et al. [67] investigated the security and privacy of commercial Bluetooth tags from various vendors. Their study uncovered multiple design and implementation issues, including unauthorized access to location reports and user data leakage.

Stephenson et al. [56] explored the abuse of IoT devices for surveillance and its impact on personal safety. By conducting interviews with victims of the abuse, this work shows the ineffectiveness of amateur attempts to locate hidden trackers.

Prior work [69] has demonstrated that Samsung SmartTags can be exploited for malicious tracking. However, this approach requires replaying advertisements from an actual SmartTag device. Moreover, SmartTag's lost message includes a liveness attribute protected by a message authentication code (MAC), requiring periodic lost message synchronization with Samsung's servers. Additionally, SmartTag does not implement end-to-end location encryption but enforces online decryption. Combined with the mandatory activation and pairing process for SmartTag usage, such attacks require much effort to launch and are easy to identify. In contrast, our attack does not involve Apple Cloud for lost message generation, key generation, or location report decryption. Moreover, our attack exploits Apple's Find My network, the largest tracking network, significantly increasing the likelihood of locating a targeted device compared to trackers on other networks.

9 Conclusion

Our work uncovered a vulnerability in the Find My service that permitted all types of BLE addresses for advertising. Exploiting this vulnerability, we proposed a novel attack, `nRootTag`, which transforms a Bluetooth device into an "AirTag" tracker without requiring root privilege escalation. By utilizing over a billion active Apple devices as finders, the attack is able to accurately track user devices. Through rainbow table-based offline key search or GPU-accelerated online key search, an infected computer can be quickly turned into a tracker. Notably, the online key search cost does not increase as the number of tracked devices grows. The evaluation shows that the attack is effective across various devices, including desktops, laptops, smartphones, and IoT devices, and worked on Linux, Windows, and Android platforms. We also discussed how the attack could be extended to track Apple devices.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-2304720, CNS-2310322, CNS-2309550, and CNS-2309477. It was also supported in part by the Commonwealth Cyber Initiative (CCI). The authors extend their gratitude to the anonymous reviewers and shepherd for their invaluable feedback and suggestions. We also acknowledge Google Earth for the map tiles used in our figures.

Ethics Considerations

All the attacks described in the evaluation were conducted using our own devices. We reported the discovered vulnerability and the attack to Apple and assisted them in reproducing the issue. Throughout the process, we provided all the details and artifacts requested by Apple. Apple has confirmed the vulnerability and the attack.

Open Science

We have released the artifact on Zenodo [18] to comply with the Open Science Policy. The artifact comprises a detailed README document and instructional videos facilitating reproduction; codes pertaining to all components necessary for the search and advertisement of lost messages; development guidelines for online key search; and three precomputed rainbow tables.

References

[1] Amazon. Seagate Exos X24 24TB Enterprise Internal Hard Drive HDD. https://www.amazon.com/Seagate-Exos-Enterprise-Internal-Drive/dp/B0CM293XCL.

[2] Amazon. Seagate IronWolf Pro 24TB Enterprise NAS Internal HDD Hard Drive. https://www.amazon.com/dp/B0CSPCFKR9.

[3] Android. Mac randomization behavior. https://source.android.com/docs/core/connect/wifi-mac-randomization-behavior/.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *USENIX security symposium (USENIX Security)*, 2017.

[5] Apple Developer. Find my network accessory specification, June 2020.

[6] Apple Inc. Apple's find my network now offers new third-party finding experiences, April 2021.

[7] Apple Inc. Airtag, 2024. https://www.apple.com/airtag/.

[8] Apple Legal. Find my & privacy, September 2023. https://www.apple.com/legal/privacy/data/en/find-my/.

[9] Microsoft Azure. Microsoft Azure: Cloud Computing Services. https://azure.microsoft.com/en-us/.

[10] Xiaolong Bai, Luyi Xing, Nan Zhang, XiaoFeng Wang, Xiaojing Liao, Tongxin Li, and Shi-Min Hu. Staying secure and unprepared: Understanding and mitigating the security risks of apple zeroconf. In *Proc. IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2016.

[11] David Balaban. The 8 biggest botnets of all time. https://cybernews.com/security/the-8-biggest-botnets-of-all-time/.

[12] Johannes K Becker, David Li, and David Starobinski. Tracking anonymized bluetooth devices. *Proc. Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[13] Alex Bellon, Alex Yen, and Pat Pannuto. Tagalong: Free, wide-area data-muling and services. In *Proc. International Workshop on Mobile Computing Systems and Applications*, 2023.

[14] wangwillian0 biemster, olivluca. FindMy. https://github.com/biemster/FindMy.

[15] Bluetooth Special Interest Group. Bluetooth core specification 4.0, June 2010. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737.

[16] Guillaume Celosia and Mathieu Cunche. Saving private addresses: An analysis of privacy issues in the bluetooth-low-energy advertising mechanism. In *Proc. ACM EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2019.

[17] Guillaume Celosia and Mathieu Cunche. Discontinued privacy: Personal data leaks in Apple bluetooth-low-energy continuity protocols. In *Proc. Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[18] Junming Chen, Xiaoyue Ma, Lannan Luo, and Qiang Zeng. Tracking You from a Thousand Miles Away! Turning a Bluetooth Device into an Apple AirTag Without Root Privileges (USENIX Security 2025 Artifacts), January 2025. https://doi.org/10.5281/zenodo.14728530.

[19] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2016.

[20] Dadoum. Provision. https://github.com/Dadoum/Provision.

[21] rkreutz Dadoum, Macleykun. anisette-v3-server. https://github.com/Dadoum/anisette-v3-server.

[22] digitalelement. NetAcuity. https://www.digitalelement.com/netacuity/.

[23] Marco Ferrante and Monica Saltalamacchia. The coupon collector's problem. *Materials matematics*, pages 0001–35, 2014.

[24] GL-iNet. Preventive Actions to Safeguard GL.iNet Users from BSSID-Based Location Tracking. Accessed: 2024-01-10.

[25] Google. Google Cloud: Cloud Computing Services. https://cloud.google.com/.

[26] Max Granzow, Alexander Heinrich, Matthias Hollick, and Marco Zimmerling. Poster: Leveraging Apple's find my network for large-scale distributed sensing. In *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2024.

[27] Alexander Heinrich, Niklas Bittner, and Matthias Hollick. Airguard-protecting android users from stalking attacks by apple find my devices. In *Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2022.

[28] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. Privatedrop: Practical privacy-preserving authentication for Apple airdrop. In *USENIX Security Symposium (USENIX Security)*, 2021.

[29] Alexander Heinrich, Milan Stute, and Matthias Hollick. OpenHaystack HCI Script for Linux. https://github.com/seemoo-lab/openhaystack/tree/main/Firmware/Linux_HCI.

[30] Alexander Heinrich, Milan Stute, and Matthias Hollick. OpenHaystack, 2023. https://github.com/seemoo-lab/openhaystack.

[31] Alexander Heinrich, Milan Stute, Tim Kornhuber, and Matthias Hollick. Who can find my devices? security and privacy of Apple's crowd-sourced bluetooth location tracking system. *arXiv preprint arXiv:2103.02282*, 2021.

[32] Apple Inc. Crowd-sourced Wi-Fi and cellular Location Services. https://support.apple.com/en-us/102515.

[33] io.net. https://io.net/.

[34] IP2Location. IP Geolocation. https://www.ip2location.com.

[35] Taher Issoufaly and Pierre Ugo Tournoux. Bleb: Bluetooth low energy botnet for large scale individual tracking. In *Proc. International Conference on Next Generation Computing Applications (NextComp)*, 2017.

[36] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. Why are they collecting my data? inferring the purposes of network traffic in mobile apps. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–27, 2018.

[37] Dave Levin. Surveilling the masses with wi-fi-based positioning systems. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2024.

[38] Linux. Linux kernel. https://github.com/torvalds/linux/blob/master/include/uapi/linux/rfkill.h.

[39] Google LLC. Control access point inclusion in Google's Location services. https://support.google.com/maps/answer/1725632.

[40] Lannan Luo, Qiang Zeng, Chen Cao, Kai Chen, Jian Liu, Limin Liu, Neng Gao, Min Yang, Xinyu Xing, and Peng Liu. System service call-oriented symbolic execution of android framework with applications to vulnerability discovery and exploit generation. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 225–238, 2017.

[41] Dennis Mantz. Internalblue–a bluetooth experimentation framework based on mobile device reverse engineering, 2018. https://github.com/seemoo-lab/internalblue.

[42] MaxMind. GeoIP. https://www.maxmind.com/.

[43] Travis Mayberry, Ellis Fenske, Dane Brown, Jeremy Martin, Christine Fossaceca, Erik C Rye, Sam Teplov, and Lucas Foppe. Who tracks the trackers? circumventing Apple's anti-tracking alerts in the find my network. In *Proc. ACM Conference on Computer and Communications Security Workshop on Privacy in the Electronic Society*, 2021.

[44] David McGrew, K Igoe, and M Salter. Rfc 6090: Fundamental elliptic curve cryptography algorithms, 2011.

[45] Microsoft. What's a Universal Windows Platform (UWP) app? https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

[46] Institute of Electrical and Electronics Engineers. IEEE Registration Authority: Assignments, 2024. https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries.

[47] paolostefanucci. GitHub Discussion - OSX-KVM Virtual Machine. https://github.com/Chapoly1305/FindMy/discussions/12.

[48] paolostefanucci. OSX-KVM. https://github.com/kholia/OSX-KVM.

[49] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. IP geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[50] Philipp Richter, Florian Wohlfart, Narseo Vallina-Rodriguez, Mark Allman, Randy Bush, Anja Feldmann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. A multi-perspective analysis of carrier-grade nat deployment. In *Proc. Internet Measurement Conference*, 2016.

[51] RunPod. https://www.runpod.io/.

[52] Amazon Web Services. Cloud Computing Services - Amazon Web Services (AWS). https://aws.amazon.com/.

[53] Narmeen Shafqat, Nicole Gerzon, Maggie Van_Nortwick, Victor Sun, Alan Mislove, and Aanjhan Ranganathan. Track you: A deep dive into safety alerts for Apple airtags. In *Proc. Privacy Enhancing Technologies Symposium (PETS)*, 2023.

[54] Pallavi Sivakumaran and Jorge Blasco. A study of the feasibility of co-located app attacks against BLE and a Large-Scale analysis of the current Application-Layer security landscape. In *USENIX Security Symposium (USENIX Security)*, 2019.

[55] Jon Gunnar Sponås. Things You Should Know About Bluetooth Range. https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range.

[56] Sophie Stephenson, Majed Almansoori, Pardis Emami-Naeini, and Rahul Chatterjee. It's the equivalent of feeling like you're in Jail: Lessons from firsthand and secondhand accounts of IoT-Enabled intimate partner abuse. In *USENIX Security Symposium (USENIX Security)*, 2023.

[57] Milan Stute, Alexander Heinrich, Jannik Lorenz, and Matthias Hollick. Disrupting continuity of Apple's wireless ecosystem security: New tracking, DoS, and MitM attacks on iOS and macOS through bluetooth low energy, AWDL, and Wi-Fi. In *USENIX Security Symposium (USENIX Security)*, 2021.

[58] Milan Stute, David Kreitschmann, and Matthias Hollick. One billion apples' secret sauce: Recipe for the apple wireless direct link ad hoc protocol. In *Proc. International Conference on Mobile Computing and Networking*, 2018.

[59] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. A billion open interfaces for eve and mallory: MitM, DoS, and tracking attacks on iOS and macOS through Apple wireless direct link. In *USENIX Security Symposium (USENIX Security)*, 2019.

[60] SuperNetworks. BSSID Randomization. https://www.supernetworks.org/pages/blog/bssid-randomization.

[61] Tile, Inc. How does tile work?, 2024. https://www.tile.com/en-us/how-it-works.

[62] Davide Toldo, Jiska Classen, and Matthias Hollick. Attaching internalblue to the proprietary macos iobluetooth framework. In *Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020.

[63] Leonardo Tonetto, Andrea Carrara, Aaron Yi Ding, and Jörg Ott. Where is my tag? unveiling alternative uses of the Apple findmy service. In *Proc. IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2022.

[64] Emiliano Tramontana and Gabriella Verga. Mitigating privacy-related risks for android users. In *Proc. IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2019.

[65] Sean Turner, D Brown, K Yiu, R Housley, and T Polk. Rfc 5480: Elliptic curve cryptography subject public key information, 2009.

[66] Vast.ai. https://vast.ai/.

[67] Mira Weller, Jiska Classen, Fabian Ullrich, Denis Waßmann, and Erik Tews. Lost and found: stopping bluetooth finders from leaking private information. In *Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020.

[68] Zhice Yang, Qianyi Huang, and Qian Zhang. Backscatter as a covert channel in mobile devices. *GetMobile: Mobile Computing and Communications*, 22(1):31–34, 2018.

[69] Tingfeng Yu, James Henderson, Alwen Tiu, and Thomas Haines. Security and privacy analysis of Samsung's Crowd-Sourced bluetooth location tracking system. In *USENIX Security Symposium (USENIX Security)*, 2024.

[70] Yue Zhang and Zhiqiang Lin. When good becomes evil: Tracking bluetooth low energy devices via allowlist-based side channel and its countermeasure. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[71] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W Freeh. Taming information-stealing smartphone applications (on Android). In *Proc. Trust and Trustworthy Computing*. Springer, 2011.

A Appendix

A.1 Apple Location Report Retrieval

Listing 1: HTTP request for downloading location reports.

```
POST /acsnservice/fetch HTTP/1.1
Host: gateway.icloud.com
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
X-Apple-I-MD: ...
X-Apple-I-MD-M: ...
X-Apple-I-Client-Time: ...
X-Apple-I-TimeZone: UTC
loc: en_US
X-Apple-Locale: en_US
X-Apple-I-MD-RINFO: ...
X-Apple-I-MD-LU: ...
X-Mme-Device-Id: ...
X-Apple-I-SRL-NO: 0
Content-Length: 121
Content-Type: application/json
Authorization: Basic ...

{
    "search": [{
    "startDate": 1737842550000,
    "endDate": 1737846150000,
    "ids": ["..."]}]
}
```

Genuine AirTag users utilize the Find My application to access AirTag locations. The Find My application lacks the capability for programmatic export of location reports and does not support integration with third-party applications. Previous studies [14,30,31] have identified APIs for accessing Find My reports. These Apple APIs are protected by multi-factor authentication, which utilizes periodically changing credentials based on an undisclosed protocol. To dispatch requests with these credentials, `OpenHaystack` employs a plug-in crafted for the Apple Mail application, specifically associated with macOS. However, with the termination of plug-in support in macOS 14, `OpenHaystack` is now obsolete. A provisional solution employed OSX-KVM virtual machines [47,48] to circumvent OS limitations, but this method is resource-intensive due to the requirement for running macOS.

We exploited the methodology outlined in *biemster/FindMy* [14], which employs a more efficient technique through the integration of specialized libraries acquired through reverse engineering [20,21]. This technique invokes the requisite cryptographic operations for authentication from a dynamic library extracted from Apple Music for Android. The library generates the necessary authentication headers linked with metadata. Significantly, this method facilitates the retrieval of reports using solely an Apple account, thereby eliminating the necessity for either physical Apple hardware or resource-intensive virtual machines. This approach dramatically lowers the barriers to accessing the Find My network, and, importantly, the minimal resource demand enables flexible integration and automation. While *biemster/FindMy* is shell interactive, we used a publicly accessible fork, *Chapoly1305/FindMy*,[1] which provides API features. An example HTTP request for requesting location reports is shown in Listing 1.

Within our threat model, we postulate that an adversary might target a large population for the purposes of profiling and surveillance. Such an attack requires the retrieval of location reports for a multitude of devices. Notably, discussions in 2021[2] and our preliminary tests conducted in 2023 suggest that Apple does not strictly enforce a throttling policy concerning the number of tags in a single request or the size of a single report file. In our experiment, we successfully queried 1,552 tags in a single request and obtained a report file amounting to 1,576,864 bytes. Our Apple account, employed for the experimentation, carried out periodic queries over a period exceeding 13 months without encountering bans or restrictions. Thus, it is plausible to exploit the Find My network to observe thousands of devices.

[1] https://github.com/Chapoly1305/FindMy

[2] https://github.com/seemoo-lab/openhaystack/issues/41