Beware of Geeks Bearing Gifts: Attacking Smart Homes

Ashraf Yassin

Beware of Geeks Bearing Gifts: Attacking Smart Homes

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Ashraf Yassin

Submitted to the Senate
of the Technion — Israel Institute of Technology
Shevat 5782 Haifa January 2022

This research was carried out under the supervision of Prof. Eli Biham and Amichai Shulman, in the Faculty of Computer Science.

Acknowledgements

I would like to thank my amazing advisors, Prof. Eli Biham and Amichai Shulman, for their professional guidance and their incredible support during the work on this thesis. I would also like to thank my beloved family for their unconditional love and support.

The generous financial help of the Technion, the Hiroshi Fujiwara cyber security research center and the Israel cyber directorate is gratefully acknowledged.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Smart homes are becoming more and more popular all over the world. They allow the homeowner to control and access his home accessories (IoT devices) from a smartphone. The smart home platform supports a variety of accessories, for example, thermostats, lights, locks, security cameras, and many others. Apple is one of the major players in the smart home industry. Apple's platform uses the proprietary HomeKit Accessory Protocol (HAP) to allow iOS devices to connect accessories to the user's smart home and control them securely. As part of this platform, most accessories are certified by Apple and follow various security specifications.

This thesis shows that the HAP protocol does not adequately authenticate certified accessories. It introduces a new attack on Apple software authentication, exploiting the ability to steal authentication data from certified accessories and the fact that the authentication data is not cryptographically protected.

In our attack, to which we call the "Gift Attack", an attacker gives a rogue accessory that pretends to be certified to a victim. Once the rogue accessory is installed in the victim's home, the attacker gains complete control over the accessory without the homeowner's knowledge. Our attack imposes a significant threat to Apple's smart home security since it can be used with any smart home accessory that the user trusts to secure his home. For example, using this attack on a door lock gives the attacker control over who comes in or out of the home, lets intruders or accomplices into the home, or locks residents out of the home. Alternatively, using the attack on security systems allows the attacker to disarm the security system and break into the home. We emphasize that the attack uses a weakness of the protocol itself rather than depending on the existence of any implementation weakness or the ability to break into the accessory.

The Gift attack can be leveraged to perform additional attacks on the HAP protocol. These attacks include hijacking newly installed accessories in the victim's home at installation time, hijacking Siri commands intended for other accessories, disclosing encrypted commands and the accessory status, or launching a DoS attack on the iOS device to prevent the user from adding new accessories.

We also present another independent attack on Apple's smart home platform. Apple provides users with the ability to control their smart home remotely when they are not at home. This attack allows impersonating users to Apple, and thus allows an attacker to receive and send forged commands without even being near the home.

Chapter 1

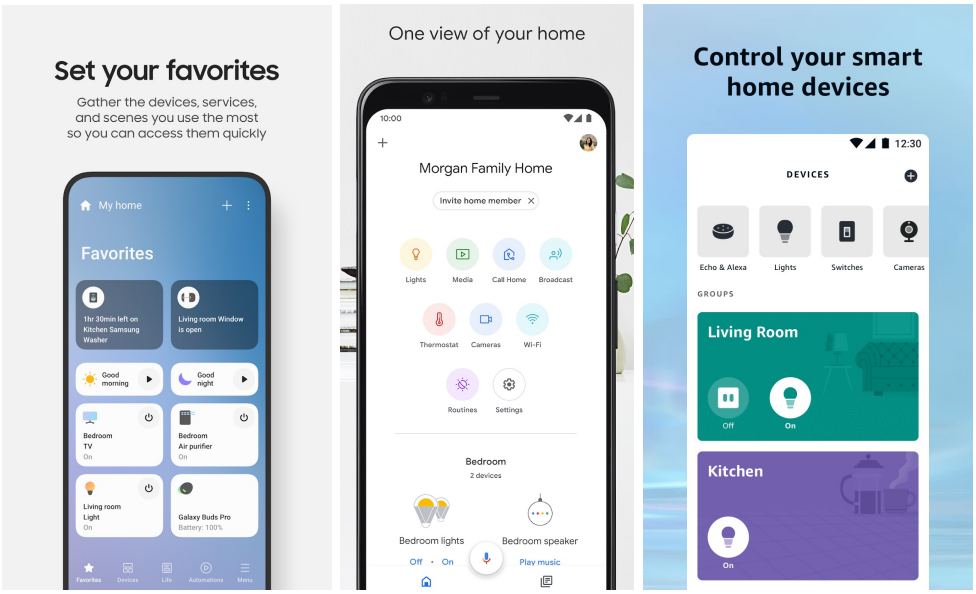
Introduction

A smart home is a home that is equipped with smart devices (IoT devices). Smart devices are home appliances connected via the Internet. The users control and manage the smart home devices mainly using their smartphone or using a voice assistant or other networked device. Examples of smart home devices that users can control include lighting systems that can be turned on and off, thermostats that can be scheduled and monitored, smart locks that can be used to grant or deny home access, security cameras to check in on the home, and even coffee brewer to brew coffee automatically.

The smart home industry is revolutionizing the home experience, and it is rapidly growing, reaching more and more homes around the world. The growth is driven by various factors, such as the increasing number of Internet users, the growing adoption of smartphone devices, and the rising disposable income of people in developing economies. According to a new research report from Martin Bäckman, an IoT analyst for the firm Berg Insight [8], the number of smart homes in Europe and North America has reached 102.6 million in 2020 and will increase significantly to 179 million by 2024. Most smart homes are mainly managed by the Internet's giant companies. Each of these companies provides an easy and secure platform for controlling the smart home. For example, the Home app by Apple (shown in Figure 1.1) , SmartThings by Samsung (shown in Figure 1.2a), Home by Google (shown in Figure 1.2b), and Alexa by Amazon (shown in Figure 1.2c). Each app works using its own proprietary protocol. For example, Apple uses the HomeKit [5] protocol, and Google uses the OpenWeave [10] protocol.

This thesis discusses the security of the smart home platform provided by Apple and its underlying protocol. The protocol is called the Apple HomeKit Accessory Protocol (HAP). It is considered one of the most secure protocols in the smart home industry. Apple says it has more than one billion active iPhones. Assuming that most Apple users will eventually use the smart home platform, the HAP protocol will have a considerable share of those smart homes and will significantly impact the daily lives of many users around the world. Thus, the HAP protocol must ensure complete privacy and security for its users.

Figure 1.1: Apple Home App



(a) Samsung App (b) Google App (c) Amazon App

Figure 1.2: Competitive Applications

1.1 Apple HAP Protocol

HAP is Apple's proprietary protocol that enables third-party accessories in the home (e.g., cameras, garage doors, and locks) to communicate with Apple products (e.g., iPhones, iPads, watches, and MacBooks). The protocol ensures that communications between Apple products and the accessories are secured and that third-party accessories are certified by Apple. The certification is made by Apple through the MFi (Made for iPhone/iPad/iPod) program [14]. The certification includes both compatibility and

security tests.

The Home app is the users' frontend for controlling the devices. It has many features:

- It supplies an easy and quick control interface for any accessory in the home.
- It allows users to control their accessories through Scenes and Automation.
- It allows users to access the home accessories from anywhere outside the home.
- It allows the homeowner to invite other people to control accessories in his home.

Apple also provides users the ability to control their home accessories by voice commands using Siri (Apple's voice assistant). Voice commands let the users control any accessory in the home without clicking any button in the Home app. For example, the user can ask Siri to close the door by saying "Hey Siri, close the door". Then, Siri closes the door and informs the user that the door was closed.

1.2 Previous Work

The Apple HAP protocol was introduced in 2014. Since its introduction, only four vulnerabilities where reported. Three of them are assigned with a CVE. The vulnerabilities are as follow:

1. **CVE-2020-9978** [3] an attacker in a privileged network position may be able to unexpectedly alter the application state.
2. **CVE-2017-2434** [2] an attacker can have an unspecified impact by leveraging the presence of Home Control on the Control Center.
3. **CVE-2017-13903** [1] a remote attacker can modify the application state by leveraging incorrect message handling, as demonstrated by the use of an Apple Watch to obtain an encryption key and unlock a door.
4. **Sneaky Element Attack** [6] an attacker with a compromised certified accessory can steal network passwords from Homekit users.

There are not enough details on the reported CVEs. So we cannot discuss them in detail in this thesis. However, we can conclude that the protocol had vulnerabilities and that each vulnerability had a huge impact on home security. On the other hand, the "Sneaky Element Attack" is published, and the publication contains all the details. We discuss it in Section 2.13.

1.3 Our Contributions

In this thesis, we show that the HAP protocol is vulnerable to multiple attacks, which compromise the security of smart homes and the privacy of their users.

Our main contribution presents a new attack on the HAP protocol, which we call the Gift Attack. The Gift Attack allows an attacker to introduce a rogue accessory into the victim's home. Though the rogue accessory pretends to be certified by Apple, in practice it is controlled by the attacker. A variant of attack enables the attacker to act as a Man-in-the-Middle (MitM) between a (real) certified accessory and an iOS device without being detected as uncertified by the iOS device or the victim.

We emphasize that every smart home that uses HAP accessories is vulnerable to the Gift Attack. Moreover, any rogue accessory installed in a smart home can compromise the rest of the home's accessories by impersonating existing accessories in the home or by hijacking a newly installed accessory using multiple vulnerabilities in the protocol.

Our Gift Attack uses a novel method to exploit certified accessories with software authentication. This attack method combines exploiting the accessories' improper authentication and the ability to reveal the accessory's authentication token. This combination enables the rogue accessory to connect with the user's iOS device without triggering a security warning to the user.

Additionally, we present four different attacks on the HAP protocol that are achievable once the Gift Attack is performed. The first attack allows an attacker to hijack Siri commands intended for other accessories. The second attack hijacks newly installed accessories that use NFC (e.g., hijack a newly installed surveillance camera to spy on the home residents). The third attack allows an attacker to disclose the status of the home (e.g., is the home's door open or closed). The fourth allows the attacker to launch a DOS attack on the victim's iOS device to prevent the victim from adding new accessories.

Finally, we introduce a new attack on the Apple Push Notification Service (APNS). In this attack, an attacker leverages an improper authentication in the APNS protocol to gain access to the user's private account in the APNS. Accessing the APNS allows the attacker to read and change the user's private messages, including messages from iMessages, HomeKit remote commands, and other Apps. The attacker can use this attack to compromise the user's smart home security. He can send remote commands to the user's home or invite other people to control accessories in the user's home without the user's consent.

1.4 Structure of the Thesis

Chapter 2 describes the Apple HAP Protocol and its security mechanisms and discusses one of the previous attacks on the protocol. Chapter 3 outlines our newly proposed "Gift Attack." Chapter 4 describes follow-up attacks to the gift attack and introduces advanced attacks on the HAP protocol. Chapter 5 introduces an attack on Apple remote commands and how an attacker can leverage it to compromise a home's security. Chapter 6 suggests mitigations to secure against the previously mentioned attacks. Finally, Chapter 7 concludes the thesis.

Chapter 2

The HAP protocol and Prior Attacks

The HAP protocol allows users to control HAP accessories easily and securely from their iOS devices. The protocol lets users control and monitor the state of the accessories. It also supports adding new accessories, configuring them, and selecting the users who can access them and their permission. For example, a user can lock and unlock the home door, turn the home lights on and off, invite other users to control his home lights, and so on.

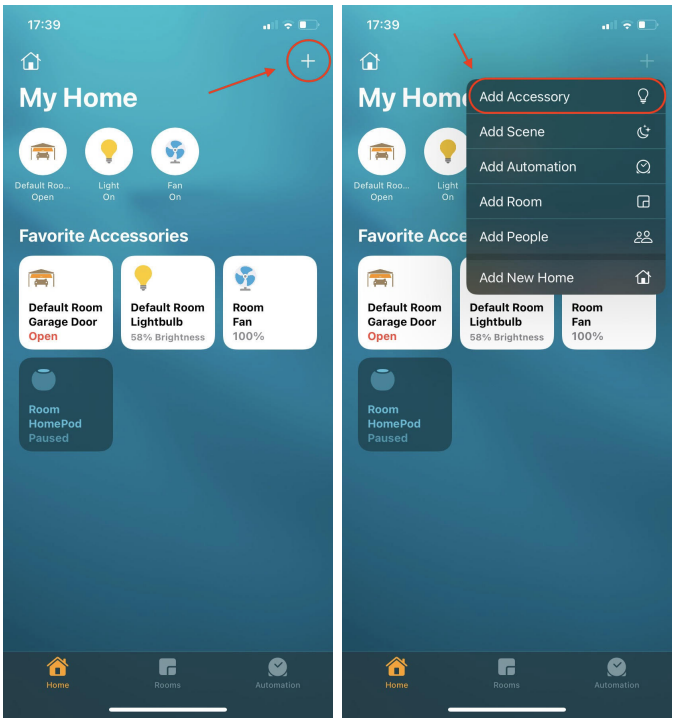
The HAP protocol provides multiple security mechanisms. The secure pairing mechanism ensures that the user can securely connect his iOS device to the correct accessory. The secure communication mechanism prevents an attacker from eavesdropping or changing commands to the home accessories. The protocol also provides a mechanism for authenticating certified accessories, which prevents attackers from impersonating certified accessories. These mechanisms cooperate by sharing a cryptographic key that ensures that secure communication is performed with the paired and authenticated accessories.

2.1 User Interaction

The user interacts with the accessory using an iOS device. The interaction includes sending functionality commands (e.g., turn on or off the home light, lock or unlock the door), as well as supporting interactions like setting the accessory in the home and inviting other users to control the accessory (e.g., inviting a family member to send commands to the home light).

Controlling the accessory is very easy and can be done with one click through the Home app. When the user opens the Home app, it presents to him all the accessories in the home (shown in Figure 2.4b), and to send a command to the accessory, the user clicks on the accessory's icon. For example, the user clicks on the garage door icon to close the garage door. To open it, the user clicks again on the icon.

Figure 2.1: The Accessory Setup Code

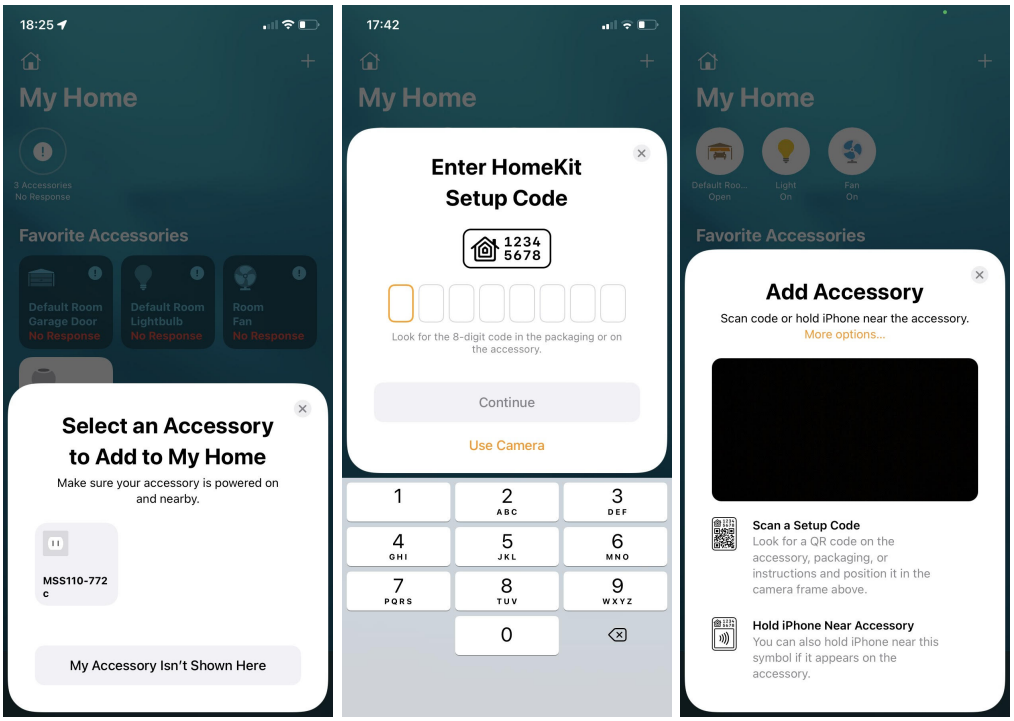


(a) Plus Button (b) Add Accessory Button

Figure 2.2: Starting a New Window to Add a New Accessory

To add an accessory to the home, the user needs to have the accessory setup code. The setup code is an 8-digit code, typically printed on the accessory itself or its package, either as a number, as a QR code (as seen in Figure 2.1), or can be digitally included in the accessory NFC tag.

When the user wants to add a new accessory to his home, he powers the accessory on and then opens the Home app in his iOS device. Using the Home app, the user clicks on the "+" button (marked in Figure 2.2a), and then clicks on the "add accessory" button (marked in Figure 2.2b). The Home app then opens a new window for adding the new accessory. The user must enter the accessory setup code to instruct the iOS device to start pairing with the accessory. The Home app provides three different methods to enter the accessory's setup code, a manual one that is always available to the user, and two that are optional and may not be supported in every accessory. The three methods



(a) Nearby Accessories Search Window

(b) Manually Entering the Setup Code

(c) Scan QR Code or tap NFC Tag

Figure 2.3: iOS Interaction for Connecting to an Accessory

are as follow:

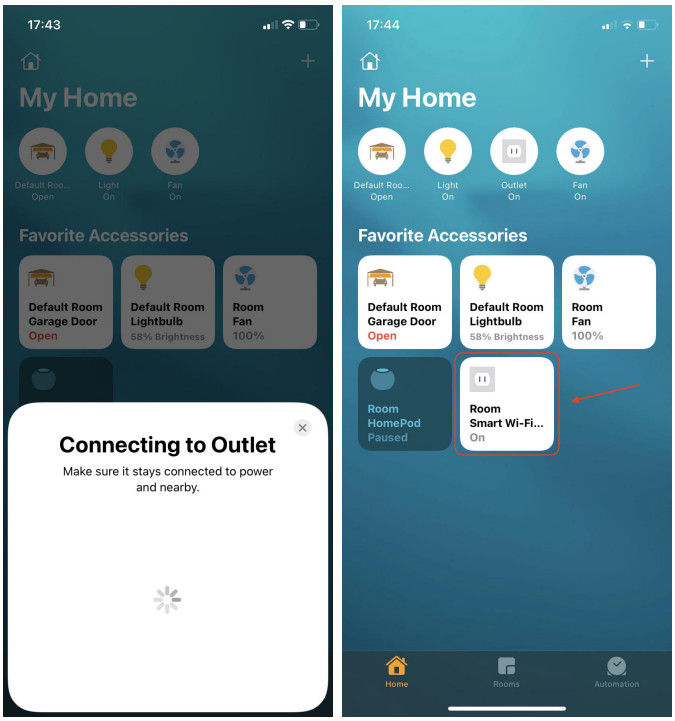
1. Manually search for nearby accessories in the search window (shown in Figure 2.3a). Then, select the wanted accessory and enter its setup code (shown in Figure 2.3b).

2. Use the iOS device's camera to scan the accessory QR code, if available(shown in Figure 2.3c).

3. Tap the accessory's NFC tag, if available (shown in Figure 2.3c).

Once the iOS device identifies the device and gets the accessory setup code, it begins setting up the new accessory and adding it to the home (shown in Figure 2.4a). On successful addition, the new accessory appears in the Home app (shown in Figure 2.4b), and the user can start to control it.

In addition to controlling the accessory itself, the user can invite other users to control his home accessory. Adding new users is done from the Home app, using the "+" button, and then clicking on the "add people" button (shown in Figure 2.2b). The Home app opens a new window in which the homeowner can enter the new user's iCloud email and send him an invite. The homeowner can invite people as either admin users or non-admin users. Both types of users can control the functionality of the accessories, e.g., turn the light on and off. Admin users can also add or remove accessories to/from the home and invite other users.



(a) During Add (b) After Add

Figure 2.4: Adding a New Accessory

2.2 Accessory States and Interaction

HAP accessories can be in one of three different states through their life cycle: Unconfigured, Unpaired, and Paired. Figure 2.5 outlines the three states:

1. **Unconfigured**: Every new accessory starts in this state, where it is not yet configured to connect to the home network. In this state, the accessory starts an access point that enables the iOS device to identify and communicate with it. The iOS device connects to the accessory's AP and configures the accessory to connect to the home network by sending the home network WiFi credentials over the AP network. Once the accessory receives the WiFi password, it turns the AP off, connects to the home network, saves the password in its long-term memory, and switches to the Unpaired state.

2. **Unpaired**: In this state, the accessory is connected to the home network but not yet assigned to any user. The user's iOS device pairs with the accessory and sets the user as the accessory admin. Once the accessory admin is set, the accessory switches to the Paired state. This state is typically transparent to the user, as the iOS device performs the pairing immediately following the connection to the home network.

3. **Paired**: This is the final state where the accessory is assigned to a user, and it can receive commands from the user.

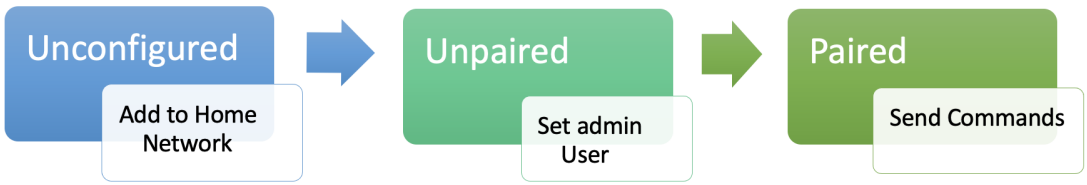


Figure 2.5: Accessory States

An accessory can downgrade its state in two cases:

- From the Paired state to the Unpaired state when all the users are removed from the accessory.
- From any state to the Unconfigured state when a user clicks on a dedicated physical reset button.

2.3 HAP Protocol Structure

From a user's point of view, the HAP protocol supports two situations: Installing a new accessory in his home and using it afterward. The former is supported by a setup mechanism that identifies the accessory, configures it in the home network, and pairs it with the user's iOS device. The latter supports sending commands to the accessory. These mechanisms are designed with security in mind and use cryptographic tools to protect themselves. They are complemented with mechanisms to add additional users and give them permissions to access and control accessories.

The accessory setup mechanism includes two phases, and the accessory usage includes one phase. The first phase in the accessory setup configures the accessory to connect to the home network and switches the accessory state from the unconfigured to the unpaired state. The second phase pairs the accessory with the user's iOS device and switches its states to the paired state. The third phase lets the user control the accessory and send commands in a secure channel.

The three phases use multiple building blocks, which we list and describe briefly. Some of the building blocks are used by a single phase, while others are used by several phases.

- **WAC**: WAC provides the identification of an accessory in the unconfigured state to the iOS device and the initial connection. Since the accessory and the iOS device are not initially on the same network, the accessory starts a WAC device that starts an access point (AP). The iOS connects to the AP and communicates over it with the accessory. The WAC is used only in the initial connection as the iOS device later communicates the accessory over the home network. When the accessory connects to the user's home network, it stops the WAC and shuts the AP down.

- **mDNS**: This protocol supports the advertisement of accessory parameters to the iOS device. An accessory may have different IP addresses throughout its life, and the iOS device needs to know its address each time it wants to connect to it. As a solution, the accessory uses the mDNS protocol and repeatedly sends mDNS advertisement messages over the WiFi network. The mDNS message includes the accessory's unique ID, IP address, TCP port, and other info. The iOS device sends an mDNS query to ask all HAP accessories in the network to send their information, listens for their replies, and uses the accessory's unique ID to select the wanted accessory. Once selected, the iOS device extracts the address and port of the accessory and connects to it.

- **Exchanging Ephemeral Keys**: This protocol provides an ephemeral key for a short-term secure channel between the accessory and the iOS device after performing mutual verification between the accessory and the iOS device. The resulting secure channel is used to secure the communication between the accessory and the iOS device during the accessory setup phases.

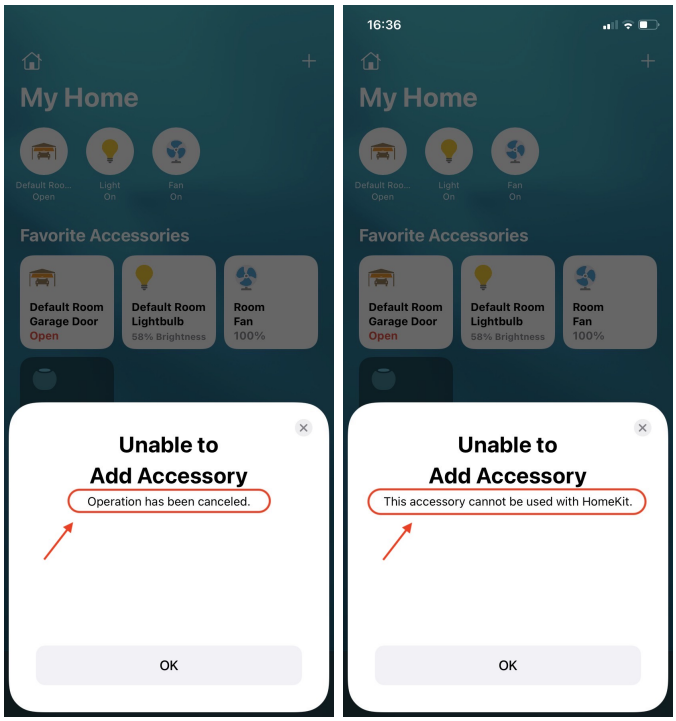
- **Certification Validation**: It is crucial to check if an accessory is certified by Apple or otherwise it is fake. The iOS device performs a Certification Validation to the accessory. Uncertified accessories do not go through the Certification Validation, and instead, they trigger a security warning to the user. A Certification Activation is also performed during the pairing phase to activate the certified accessory.

- **Certification Activation**: After validating the certification of an accessory the iOS device "activates" it. The activation includes updating Apple that the accessory is now paired and updating the accessory with data from Apple. We are not sure about the purpose of activation since the accessory works fine without it.

- **Exchanging Long-term Keys**: At the end of the accessory setup, the iOS device and the accessory exchange long-term public keys. These keys are used for authentication, where the accessory ensures that only paired users can send commands to the accessory, and the iOS device ensures that only the paired accessory is getting the user command.

- **Pair Verify**: After the accessory setup is complete, the accessory and the iOS device use the Pair Verify procedure to create a secure channel between them. The Pair Verify uses the long-term keys to perform mutual authentication and performs an ephemeral key exchange to create a key to secure the channel.

These building blocks are used by the three phases of the communication with the accessory. The structure of these phases is as follows.



(a) User disapproval (b) Certification Failure

Figure 2.6: Error Messages While Adding New Accessory

2.3.1 Phase 1: The Configuring Phase

To use a HAP accessory in the home, the user must first configure the accessory to connect to his home network. To do that, the unconfigured accessory starts an access point (by a WAC device) and starts an mDNS service in the AP network. When the user asks the iOS device to search for a new accessory, the iOS device looks for a WiFi network (with the required properties) and connects to it. Once connected to the accessory's AP network, the iOS device sends an mDNS query to ask all HAP accessories in the network to send their information. It then listens for mDNS messages, fetches the accessory service parameters from the mDNS message, and connects to the accessory service as instructed by the mDNS message.

Once connected to the accessory's service over the AP WiFi network, the iOS device and the accessory create a secure channel to secure the rest of the configuration process. The iOS device and the accessory exchange ephemeral ECDH keys to create the secure channel. The exchange is authenticated using the accessory setup code and results with a shared key, which is then used to encrypt and authenticate the rest of the session.

The iOS device expects for a certified accessory. If the accessory is not certified by Apple, the iOS device triggers a warning to the user and asks for his confirmation to add the accessory. If the user does not confirm it, the configuration phase ends with an error message "Operation has been canceled" (shown in Figure 2.6a). If the accessory is certified, the iOS device validates the certification. On failure, the configuration phase

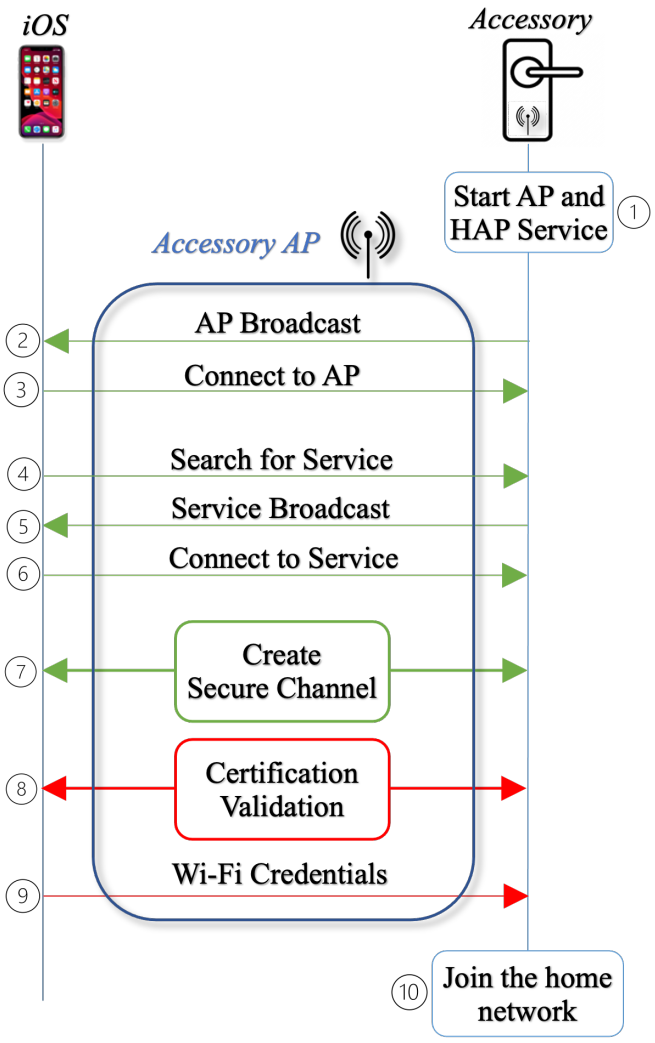


Figure 2.7: Phase 1: Configuring the Accessory to connect to the Network

ends with the error message "This accessory cannot be used with HomeKit" (shown in Figure 2.6b). On successful validation or user approval, the iOS device transmits the WiFi credentials to the accessory. Once the accessory gets the WiFi credentials, it shuts its access point down and then connects to the user's home network.

The configuration phase is summarized by the following steps, and is illustrated in Figure 2.7.

1. The accessory starts a WAC device by starting an AP and an mDNS service.
2. The accessory sends its AP broadcasts.
3. The iOS device recognizes the new accessory WiFi network. Upon user request, the iOS device connects to the accessory's WiFi network.
4. The iOS device searches for HAP services in the network by sending an mDNS query.

5. The accessory sends its service information in an mDNS response.

6. The iOS device receives an mDNS message and fetches the service information from it and connects to the accessory service.

7. The iOS device and the accessory authenticate each other by the accessory setup code and create a shared key that secures the rest of the phase (secure messages in the figure are marked in red).

8. The iOS device checks the accessory certification or ask for the user approval:

   (a) If the accessory is not certified, the iOS device requests the user's approval to add it. If the user does not approve it, the configuration phase ends with a failure.

   (b) If the accessory is certified, it proves that to the iOS device. If it fails, the configuration phase ends with a failure.

9. On success, the iOS device sends the WiFi credentials to the accessory.

10. Once the WiFi credentials are received, the accessory shuts its access point down and connects to the user's home network.

2.3.2 Phase 2: The Pairing Phase

Once the accessory connects to the home network, the iOS device can start pairing the accessory. The pairing process assigns the user as the accessory's owner and connects the accessory as part of the user's smart home.

At the beginning of the phase, both the accessory and the iOS device are connected to the home network, and the accessory has its mDNS running. To start the pairing process, the iOS device sends an mDNS query to ask all HAP accessories in the network to send their information. It then listens for mDNS messages, fetches the accessory service parameters from the mDNS message, and connects to the accessory service as instructed by the mDNS message.

Once connected to the accessory's service over the home network, the iOS device and the accessory create a secure channel to secure the rest of the pairing process. The secure channel is created the same way as in the configuration phase (ECDH key exchange that is authenticated using the accessory's setup code).

The iOS device then validates that the accessory is certified by Apple or requests the user approval. The certification is done in the same way as in the configuration phase. On successful validation, the iOS device activates the accessory. The activation includes updating Apple about adding the accessory and updating the accessory of special data. After the activation, the iOS device and the accessory exchange long-term public keys. These public keys are used to establish mutual authentication in future sessions.

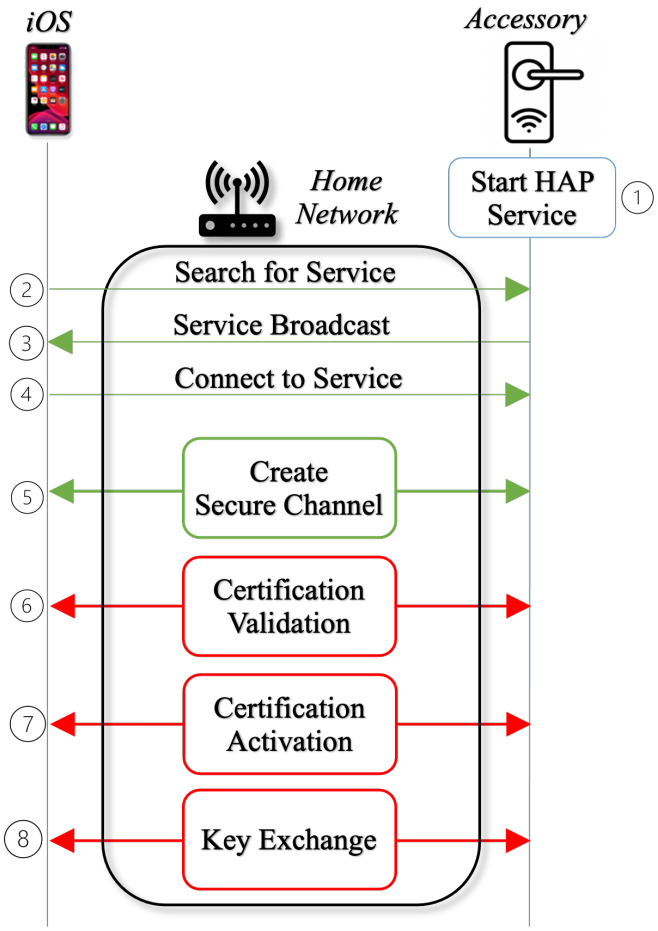


Figure 2.8: Phase 2: Pairing the Accessory with the iOS device

The pairing process is summarized by the following steps, and is illustrated in Figure 2.8.

1. The accessory starts an mDNS service in the home network.
2. The iOS device searches for the HAP service by sending an mDNS query in the network.
3. The accessory sends its service information in an mDNS response.
4. The iOS device receives the mDNS response, fetches the service information from it and connects to the accessory service.
5. The iOS device and the accessory authenticate each other by the accessory setup code and create a shared key that secures the rest of the phase (secure messages in the figure are marked in red).
6. The iOS device checks the accessory certification or for the user approval:
   (a) If the accessory is not certified, the iOS device validates that the user approved it in the configuration phase. If the user did not approve it, the iOS device

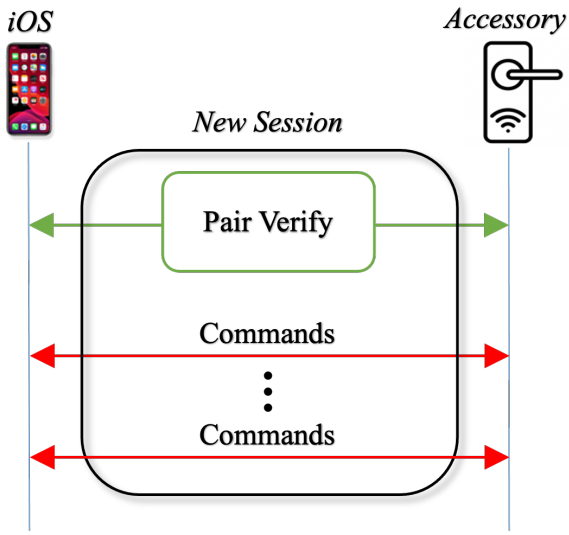


Figure 2.9: Phase 3: Sending Commands Securely

asks for his approval. If the user does not approve it, the pairing phase ends with a failure.

(b) If the accessory is certified, it proves that to the iOS device. If it fails, the pairing phase ends with a failure.

7. On success, the iOS device activates the certified accessory.

8. The iOS device and the accessory exchange long-term public keys.

2.3.3 Phase 3: The Controlling Phase

Once the accessory's setup is complete, the user can use it and send commands to control it. The HAP protocol supports multiple types of commands. There are functional commands (e.g., turn the light on or off), commands to add new users to control the accessory, commands to remove users, and commands to list the added users. To ensure these commands' security and prevent attackers from eavesdropping or changing the commands, the user's iOS device maintains a secure channel with the accessory. The secure channel is created using the Pair Verify procedure at the start of each session. The Pair Verify procedure uses the Station-to-Station protocol to perform bidirectional authentication, resulting in a mutually authenticated shared secret for future session security. Figure 2.9 illustrates the creation of a new session and sending commands that are secured using the Pair Verify procedure.

2.4 WAC Devices

The Wireless Accessory Configuration (WAC) feature is an Apple licensed technology introduced in 2013 [13]. HAP accessories use the WAC feature to be configured to

Type	Length	Value
0	3	Flags (Configured status, WiFi Channel, Authentication method)
1	N	Name (presented to the user in the app)
2	N	Manufacturer
3	N	Model
7	6	Device ID
8	2	Category
9	4	Setup hash

Table 2.1: Vendor Specific Data

connect to the user's home network. When using the WAC feature, they advertise themselves as a WAC device to inform their existence to nearby iOS devices. Users can then see the WAC device in the iOS device, and they can then choose to configure it to connect to the home network. Setting up a WAC device is a user-friendly process. It does not require the user to type in the network name and password, the user selects the accessory, and the iOS device automatically configures the accessory to connect to the user's network.

To explain how WAC devices work, consider a HAP accessory that wants to connect to the user network. It starts a WAC device by starting a software access point (AP) that supplies an unencrypted Wi-Fi network. The AP advertisement messages that publish the SSID of the AP indicate it as a WAC device. To configure the accessory, the iOS device connects to its AP and securely sends the local network credentials, including the Wi-Fi SSID and password, down to the accessory. Then the accessory disconnects its AP and connects to the user's network.

The AP broadcasts a beacon that includes its name (SSID) and other information about the network. Not every AP is considered a WAC device. The accessory has to include an Apple-assigned number to its beacon advertisement that identifies it as a WAC device.

As the details are not officially published, we reverse-engineered it by using Wireshark to capture the WAC beacon and using the iOS device logs. Table 2.1 describes this Apple-specific data that we identified. Notice that each element in this protocol is sent in a Type-Length-Value (TLV) format.

2.5 mDNS Services

The iOS device needs to search and find any HAP accessory in its network, including paired and unpaired accessories in the user's home network or unpaired and unconfigured accessories that are not in the user's home network.

To help the iOS device find the HAP accessories, they utilize the multicast DNS (mDNS) protocol. The mDNS protocol is a zero-configuration and multi-platform service designed to resolve hostnames to IP addresses within small networks. Using mDNS,

HAP accessories send mDNS messages that include their information to advertise their presence on the network. These messages are received by the devices connected to the network.

iOS devices search for HAP accessories and can display the found accessories to the user or connect to them if the user wishes to. To search for HAP accessories in the network, the iOS device sends an mDNS query requesting HAP accessories to send their information. Once the accessory receives the query, it sends an mDNS response that includes its information. The accessory also sends its information once it is powered on or once its information has changed.

The accessory mDNS messages include its IP and port address and its information in the TXT record. The iOS device uses the IP and port address to connect to the accessory and uses the TXT records to find other information about the accessory state. The TXT record includes the following values:

1. Model name.
2. Device ID.
3. Category type.
4. Feature flag.
5. Pairing status.
6. Protocol version.
7. State number.
8. Configuration number.
9. Optional setup hash.

The model name is the name that is presented to the user. The device ID is used by the iOS device to identify different accessories. The category type is the accessory type (such as lock, light bulb, fan). The feature flag includes the accessory-supported authentication methods (i.e., software and hardware authentication). The pairing status contains the accessory status (i.e., Unconfigured, Unpaired, Paired). The Protocol version includes the version string (e.g., "1.0", "1.1"). The state number is always 1. The configuration number is used to track changes in the accessory information (increments by one on each change). The setup hash is used by the enhanced setup feature, where it helps the iOS device to identify the accessory.

2.6 Exchanging Ephemeral Keys

The Ephemeral Keys exchange is used to create a secure channel between the accessory and the iOS device during the accessory setup. The secure session protects from attackers that do not have the accessory's setup code.

The secure channel is created using the entrance control protocol called Secure Remote Password protocol (SRP) [21]. SRP is a secure password-based authentication and key-exchange protocol. It exchanges a cryptographically-strong secret as a byproduct of successful authentication, enabling the two parties to communicate securely. The used SRP protocol is the SRP-6a version of the protocol, and it uses the accessory's setup code as the SRP password. The technical details of the protocol are not crucial for our thesis, and you can learn more about it in the HomeKit Accessory Development Kit [16].

2.7 Accessories Certification

To be trusted by the users, HAP accessories must be verified for authenticity and security level. Apple uses the MFi program to certify a wide range of accessories (e.g., smart home accessories, power and cables, headsets and headphones, and more).

Accessory vendors enroll in the MFi program to certify their accessories and be able to sell their accessories to Apple users. The MFi program ensures that these accessories work reliably with HAP protocol and that they meet minimum security requirements, as well as checking that the vendor is legitimate and that the vendor indeed manufactured the accessory.

The MFi program supports two kinds of authentications: hardware authentication and software authentication. Hardware authentication is the older standard, and it requires a security chip. In 2017 it was replaced by a newer software authentication that does not need a specialized hardware chip and works in a slightly different protocol.

2.7.1 The MFi Program

The MFi program is intended for hardware accessories that connect electronically to iPhone, iPad, iPod, and Apple Watch. The MFi Program offers a broad range of wireless and wired technologies that can be used in accessories that a vendor plans to develop or manufacture [14].

A major focus of the MFi program is to set a collection of security specifications and hardware requirements that third-party accessories must meet. An accessories vendor can enroll in this program to certify his accessories.

Certified HomeKit accessories come with a HomeKit badge sticker that is placed on their package, and a dedicated secret placed in their chip. Certified accessories are assured to work with HomeKit or with any other Apple software or devices [12]. On the other hand, accessories that are not certified by Apple may or may not reliably work with Apple devices, and thus they trigger a security warning to the user when they are connected. Such a warning informs the user of the risks and requests his approval to add the uncertified accessory. Such a warning is shown in Figure 2.10.

Apple uses a variety of criteria for certifying accessories, such as compliance to Apple

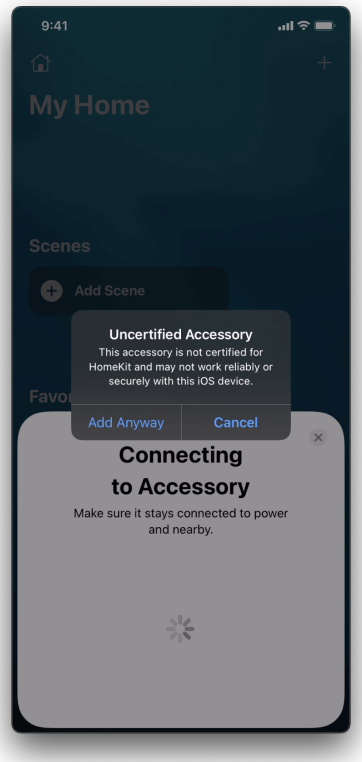


Figure 2.10: The iOS Device Warning to the User

standards and protocols and various safety and security requirements. The certification is only given after the accessories are tested by Apple's authorized testing agent.

In order to let iOS devices know whether an accessory is certified, Apple provides two authentication methods, namely hardware authentication, and software authentication. A certified accessory with hardware authentication includes a certificate signed by Apple. The certificate is installed within a co-processor (MFi chip) that provides secure encryption and signing, and the chip is installed in the accessory. On the other hand, certified accessories with software authentication have only software tokens, which Apple supplies.

2.7.2 Software Authentication

The software authentication method was introduced in 2017 as an alternative to the hardware authentication method, and we focus on this method since nowadays, most of the accessories use it. The iOS device uses this method to validate that the accessory is indeed certified by Apple. Since this method is only available to vendors enrolled in the MFi program, we had to reverse-engineer it. We split this method into two phases. The first one validates that the accessory is certified, and the second is responsible for activating the certified accessory.

A certified accessory that supports software authentication uses a software token,

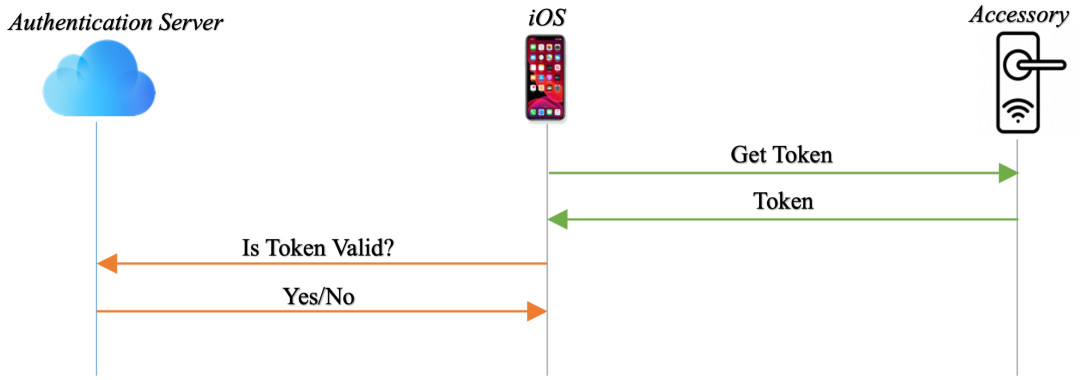


Figure 2.11: The Certification Validation Process

and each accessory is given a token generated and validated by Apple.

The accessory's certification validation is performed at two different times of the accessory's life cycle:

1. Before it connects to the home network: the iOS device authenticates the accessory before configuring it to connect to the user's home network, and for that purpose, it uses only the Certification Validation process.

2. Before it pairs with the user: the iOS device authenticates the accessory before pairing with it and adding it to the Home app, and for that purpose, it uses both the Certification Validation and Certification Activation processes.

Both times, the accessory sends its certificate and a signature on the common DH keys to prove its identity.

2.7.2.1 Certification Validation

To Validate that an accessory is certified by Apple, the iOS device asks the accessory for its token and then asks Apple to validate it. On successful validation, the iOS device continues to the next step in setting up the accessory. On failure, it stops setting up the accessory and presents an error to the user stating that the accessory is not certified by Apple as it claims. The certification validation process is illustrated in Figure 2.11, and is summarized with the following steps:

1. The iOS device asks the accessory to provide its token.

2. The accessory sends its token back to the iOS device.

3. The iOS device sends the token to the Apple authentication server.

4. Apple authentication server responds with the status of the token.

The iOS device communication with the Apple authentication server is secure under HTTPS sessions.

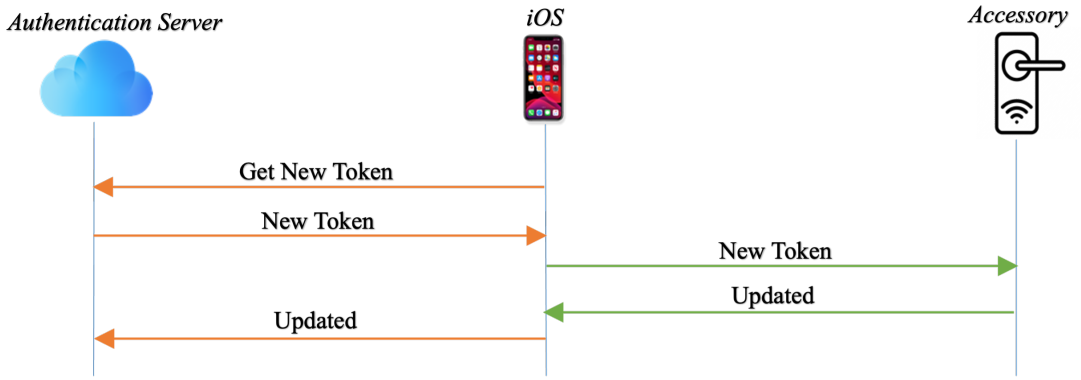


Figure 2.12: The Certification Activation Process

2.7.2.2 Certification Activation

The Certification Activation process is performed in the pairing process after the certification validation. The certification activation is used to update the accessory with its new token. We are not sure why there is a need to update the accessory token. We know that the token includes a timestamp of its creation, and one possibility is that the certification activation process is used to track accessories and their time of activation.

The technical details of the process are as follows: The iOS device requests a new token from the Apple authentication server and updates the accessory with the new token. The accessory updates the stored token with the new one and confirms the successful update to the iOS device. The iOS device confirms updating the token to the Apple server, which invalidates the old token and activates the new token. The certification activation process is illustrated in Figure 2.12, and is summarized with the following steps:

1. The iOS device requests a new token from the Apple authentication server.
2. Apple authentication server responds with a new token.
3. The iOS device sends the new token to the accessory.
4. The accessory updates the token in its persistent memory and responds with an updated message.
5. The iOS device sends the updated response to the Apple authentication server.

The iOS device communication with the Apple authentication server is secure under HTTPS sessions.

2.8 Long-Term Keys Exchange

The goal of the pairing is to exchange Ed25519 long-term public keys between the iOS device and the accessory. These keys are stored in persistent storage and used

to authenticate each side later during the Pair Verify procedure. The key exchange between the iOS device and the accessory is also performed securely under the previously established secure channel (using the previously mentioned SRP session). We do not explain the key exchange scheme in detail since it is not essential to our discussions. Detailed information about it is available in the HAP specification [15, Section 5.6.5.2].

2.9 Pair Verify

The Pair Verify procedure uses the Station-to-Station protocol to perform bidirectional authentication, resulting in a mutually authenticated shared secret for future session security. Once the Pair Verify procedure is performed, the user's iOS device can securely send commands to the accessory. The following describes the Pair Verify procedure in high-level as it is not necessary for our discussions: The iOS device and the accessory exchange ECDH keys to create a shared key to secure the session. The key exchange is authenticated using the long-term public keys. Recall, the accessory and the iOS device's long-term public keys are exchanged in the pairing process. Full details on the Pair Verify procedure can be found in the HAP specification [15, Section 5.7].

2.10 Commands

There are two types of commands, functional commands and configuration commands. The configuration commands are identical in all the accessories. They include adding a new user to control the accessory, removing a user, and listing all the users who have control over it.

In addition to configuration commands, each accessory supports a variety of functional commands that control the accessory's state. For example, a door lock supports lock and unlock commands, and a light bulb supports turn on and turn off commands. For example, a light bulb with a dimming feature also exposes a command that controls the dimming percentages. At the end of the pairing process, the iOS device fetches the list of supported commands from the accessory and presents them to the user in the Home app. On/Off switches are presented by a switch button in the Home app, and a dimmer is presented by a slider. Each accessory may provide services for several sub accessories. Their list is sent to the iOS device as a part of the list of supported commands.

The commands are transmitted to the accessories as JSON objects that have three main key-value pairs. The first key ("aid") specifies the sub-accessory instance ID, the second key ("iid") specifies the accessory characteristic instance ID (e.g., on, lock), and the third key ("value") specifies the requested value of the characteristic. The value is typically in the range 0–100, wherein a binary feature is represented by 0=false and 1=true. The accessory sets both the accessory instance ID and the characteristic

```
{
  "characteristics": [
    {
      "aid": 2,
      "iid": 8,
      "value": false
    }
  ]
}
```


Figure 2.13: Turn Off Command

instance ID, and the iOS device fetches them after adding the accessory to the home and whenever they change (e.g., in case of a firmware update).

Figure 2.13 illustrates the JSON instance for turning off a light bulb that has accessory number 2 and characteristic number 8. Notice that true and false are represented by true and false in the JSON object, while their internal representation is 1 and 0, respectively.

2.11 Hardware Authentication

When the Homekit was first introduced, HAP accessories had to prove they are certified by Apple using the hardware authentication method. Although this method is not in wide use any more, we describe it in order to introduce previous attacks on it.

In the hardware authentication method, each accessory uses a certificate issued by Apple, and the accessory had to include an MFi Chip. The MFi chip is used to secure the certificate and the private key. The accessory's certification validation is performed at two different times of the accessory's life cycle:

1. Before it connects to the home network: the iOS device authenticates the accessory before configuring it to connect to the user's home network, and for that purpose, it uses a protocol called MFiSAP.

2. Before it pairs with the user: the iOS device authenticates the accessory before pairing with it and adding it to the Home app.

Both times, the accessory sends its certificate and a signature on the common DH keys to prove its identity.

Apple provides details on the hardware authentication methods only for accessory vendors enrolled in the MFi program. So we had to reverse-engineer the hardware authentication method. In the next subsection we describe the MFiSAP protocol used in this method.

2.11.1 MFi Security Association Protocol (MFiSAP)

We previously mentioned that when configuring an accessory to connect to the network, the iOS device performs a certification validation on the accessory and validates that it is certified by Apple. We discussed the protocol that validates accessories that have software authentication support. This subsection describes the protocol that validates certified accessories with hardware authentication support.

Certified accessories with hardware authentication are authenticated using the MFi Security Association Protocol (MFiSAP). The MFiSAP protocol validates the accessory certificate to verify that it is certified by Apple. The protocol also provides an encrypted channel using a shared key. The following provides more technical details of the protocol.

At the start of the protocol, the iOS device and the accessory exchange Curve25519 public keys. The iOS device sends its ephemeral Curve25519 public key A. The accessory responds with its ephemeral Curve25519 public key B and with two additional values, the $Certificate$ and a $Signature$. The signature is created using the MFi chip, and it is over the hash of the two public keys.

$$Signature \quad = \quad \text{RSA-SIG}(\text{SHA-1}(A, B)).$$

The signature is calculated with the private key that corresponds to the certificate's public key. The signature is encrypted using AES_CTR algorithm under the key K and initial value IV, both derived from:

$$\begin{aligned} S &= \text{ECDH shared key}, \\ K &= \text{SHA-1}(salt1, S), \\ IV &= \text{SHA-1}(salt2, S). \end{aligned}$$

When the iOS device receives the accessory certificate and signature, it validates the certificate and verifies the signature. On successful validations, the iOS device sends the network credentials and receives the accessory info in response. The MFiSAP protocol is outlined in Figure 2.14 and is summarized with the following steps:

1. The iOS device sends its public key A.
2. The accessory sends its public key B, its certificate, and an encrypted signature.
    - The iOS device validates the certificate and verifies the signature.
3. On success, the iOS device sends the Wi-Fi credentials.
4. Finally, the accessory sends back its info.

Once the MFiSAP protocol is successfully completed, the accessory uses the received Wi-Fi credentials to connect to the user's home network.

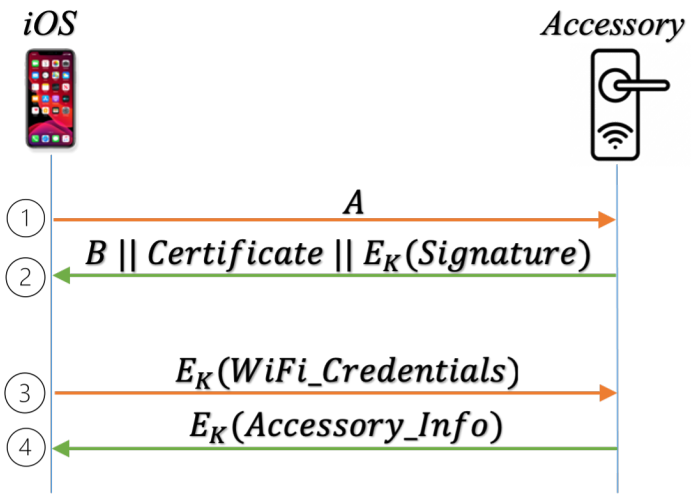


Figure 2.14: MFiSAP – the Security Association Protocol

2.12 Security Requirements

Unfortunately, Apple never published the threat model of their smart home platform nor the security properties provided by their mechanisms. This fact makes it difficult to analyze the security of this platform. We thus compile here a partial list of required security properties and mechanisms. This list contains properties that Apple provides in their mechanisms (based on reverse engineering of their protocols) and our interpretation of the width of the scope that these properties should cover. This partial list includes:

- **Certified Accessories** – Accessories should be verified for their authenticity. Apple provides two types of certifications for authentication: one is certified accessories that come with a digital certificate or authentication token generated by Apple, while the other is uncertified accessories where the user testifies for their authenticity. In the former case, the accessory is automatically detected and approved by the iOS devices, while in the latter case Apple warns the user, and the user manually approves or rejects the accessory. Clearly, it is expected that an accessory cannot impersonate a legitimate certified accessory.

- **Secure (Local) Commands** – The iOS device should be able to securely communicate with accessories and send commands to the accessories when both are directly connected to the same local network. Apple protects the communication between the user and his home accessories with encryption and authentication. Clearly, an attacker should not be able to guess the content of the commands or the responses nor to modify them or forge them.

- **Secure Pairing** – The iOS device and the accessory should be able to authenticate each other and create a secure session for their further communications. Apple provides a secure pairing mechanism for this purpose. Clearly, such a mechanism

should ensure that an attacker cannot hijack the accessory during the installation time or at any other time during the session.

- **Definite Interpretation of Voice Commands** – When using Siri, the voice commands should have a definite interpretation, and if it has more than one it should ask the user for a more specific command or alternatively it should cancel the command and report that to the user. Apple uses Siri to perform the user's voice commands in the home. Siri is programmed as a fail-closed system, meaning that if it fails to interpret the command, it stops the command's execution and reports that to the user. Clearly, a failure in interpretation due to multiple meanings should not result in the execution of any command and should not result in a report of the success of such a command.

- **Safe Protocol** – The use of the protocol (even by an attacker) should not affect the user experience and cause a major failure on the iOS device (e.g., shutdown, denial of service, UI freeze, etc.). Apple's smart home platform uses protocols that are designed to secure against such attacks. Clearly, an attacker should not be able to utilize the protocols to harm the user's iOS device.

- **Secure Remote Commands** – The commands from a remote user should be secured, and only authorized users should be able to send remote commands to the user's home. Apple uses the APNS protocol to secure home remote commands. The APNS protocol ensures that messages are sent from an authorized user and that the commands are end-to-end encrypted. Clearly, such a protocol should ensure that an attacker cannot bypass the authentication and send commands to the victim's home.

2.13 Prior Attacks

The Apple HomeKit protocol is considered very secure, and indeed many users trust Apple with their smart home security. As we mentioned earlier, since HomeKit's introduction in 2014, four vulnerabilities have been published on it. Only one of the vulnerabilities is published with its details. This published vulnerability leads to an interesting attack called "The Sneaky Element Attack," which exploits WAC devices to steal users' Wi-Fi passwords. During the attack, the attacker can be located in the "parking lot", i.e., outside the user's home, and still listen in to packets and send packets to the user. Let us describe the vulnerability and the attack.

In 2017 Don Bailey [6] uncovered a flaw in the HAP protocol by utilizing WAC devices to steal users' Wi-Fi passwords. This attack is based on the fact that iOS devices trust WAC devices with an MFi chip. Using this trust, an attacker can make the user's iOS device transfer the Wi-Fi credentials without validating the accessory setup code. As he illustrated, an attacker with a compromised certified accessory that

supports hardware authentication can perform a parking lot attack and impersonate a certified accessory. He then tricks the iOS device into handing the attacker the Wi-Fi password.

The attack is based on insufficient authentication in the MFiSAP protocol, where the iOS device configures the accessory to connect to the user's network. The MFiSAP protocol validates the MFi chip in the accessory. However, oddly, it does not validate the accessory's setup code to validate the identity of the accessory before handing the accessory the Wi-Fi password.

The technical details of the attack are as follows. The attack assumes that the attacker has already compromised a certified accessory and that the attacker is located in a victim's parking lot and waits for the user to pair a new accessory. Once a user powers on the new accessory and tries to pair with it, the attacker starts a fake accessory that pretends to be the user's accessory, i.e., the fake accessory uses the exact details as the victim's new accessory. Now the user sees two identical accessories in the Home app. There is a 50% chance that the user chooses the attacker accessory. If so, the user chooses the attacker's accessory, and the iOS device starts the pairing process with it. The iOS device starts configuring the accessory during the pairing process to connect to the user's network using the MFiSAP protocol. During the MFiSAP protocol, the iOS device requests the accessory certificate and a signature over the exchanged public ECDH key. The attacker, who has no legitimate keys, uses the compromised accessory MFi chip to generate a valid signature and sends it to the iOS device. The signature passes the iOS device validation since the attacker responded with info created by a valid MFi chip. The iOS device then sends the user's Wi-Fi password to the attacker's accessory without realizing it is fake. Unfortunately for the attacker, the pairing process fails later since the attacker does not have the accessory's setup code. However, the attacker successfully acquires the user's Wi-Fi password before the failure.

Although an attacker can gain the user's password remotely by using this attack, it has two drawbacks and restrictions:

1. It is hard to compromise a certified accessory: the attack requires the attacker to compromise a certified accessory and inject his own code to control it. Compromising a certified accessory demands significant effort because certified accessories get tested thoroughly by Apple, and the chance of finding a code execution vulnerability is not high.

2. It is hard to scale the attack: an attack on multiple networks using the compromised accessory requires the attacker to travel to each user's home and wait for each user to install a new accessory. Attacking many networks using this approach is time and energy-consuming and thus hard to implement.

Chapter 3

The Gift Attack

This chapter introduces the Gift Attack that lets an attacker control accessories in the victim's home. In this attack, the attacker donates a rogue accessory as a gift to the unsuspecting victim, and the victim installs it in his home network. Though it is clearly not certified, the rogue accessory in the gift attack can perform the pairing process with the iOS device without triggering a warning to the victim. Thus, the victim has all reasons to believe it is a certified accessory. Once admitted into the network, the attacker monitors and controls the rouge accessory while it pretends to be controlled by the victim. For example, using the Gift Attack with a rogue door lock accessory gives the attacker complete control over who comes into or out of the home. He can then remotely let intruders or accomplices into the victim's home or lockout the actual residents.

Our attack exploits the now popular software authentication method for certified accessories. The attacker extracts a valid authentication token from a certified accessory and uses it for authenticating the rogue accessory. This attack is made possible because there is no cryptographic binding between the token and the current pairing session or the accessory.

The MFi certification process provides security guarantees and certifies only legitimate accessories. Hence it is difficult to get past the certification process with an accessory that includes a malicious code. It should also be difficult to introduce a malicious code into an existing certified accessory. So the MFi program seems to ensure that accessories that pass the certification checks cannot run malicious code. With this in mind, we would like to create a rogue accessory that impersonates an already certified HAP accessory, thus bypassing the MFi certifications assurance.

Contrary to the Sneaky Element attack [6], our attack can impersonate certified accessories without the need to break into them. Moreover, it is easier to execute our attack than the Sneaky Element attack.

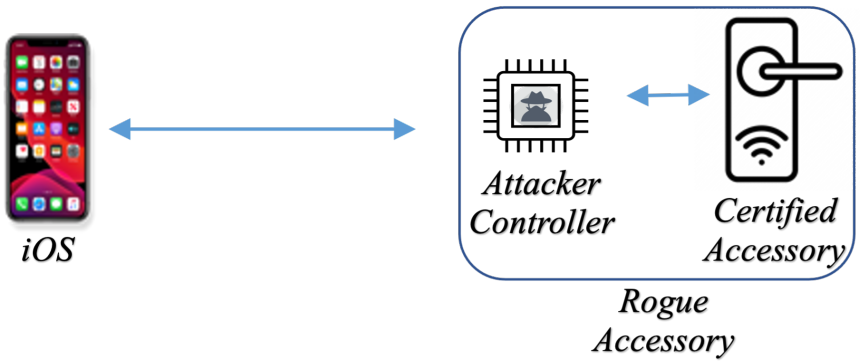


Figure 3.1: The Rogue Accessory Setup

3.1 The Six Phases of the Attack

In the Gift Attack, the attacker introduces a rogue HAP accessory into a home network. This rogue accessory behaves (from a protocol perspective) like a certified accessory. To create the rogue accessory, the attacker creates a lightweight component that performs the necessary steps required to introduce the rogue device into the network and behaves as a MitM to relay all accessory functionality to an original accessory. The lightweight component is typically placed inside the original accessory in a hidden place. Figure 3.1 outlines this setup.

The Gift Attack is comprised of the following phases:

1. The devices acquisition phase: the attacker buys a legal accessory and a controller.
2. Physical setup phase: the attacker prepares the rogue accessory.
3. Gifting phase: the attacker gives the rogue accessory to the victim.
4. Installation phase: the rogue accessory connects with the victim's iOS device. The iOS device does not detect that the accessory is uncertified.
5. Normal operations phase: the attacker allows the victim to use the gift accessory as expected from an original accessory.
6. Performing attacks phase: the attacker performs malicious operations using the rogue accessory (e.g., locking and unlocking the home's door).

3.1.1 The Devices Acquisition Phase

In the acquisition phase, the attacker buys the following devices:

1. A Homekit accessory, which is certified and supports software authentication.
2. A MitM controller which the attacker programs to:
   (a) Advertise its existence to the victim.

(b) Act as the MitM between the victim and the certified accessory.

(c) Receive his remote commands by connecting to his command and control (C&C) server.

The MitM controller should be small enough to be well-hidden inside the accessory chassis. The controller should also have a Wi-Fi MCU to be able to communicate. An example of such a controller is the ESP chip [20] that consists of a low-cost, low-power system-on-a-chip micro-controller with an integrated Wi-Fi MCU.

3.1.2 Physical Setup Phase

In this phase, the attacker prepares a rogue accessory (which we call the gift or the gift accessory) to be sent to the victim.

The attacker programs the controller to act as a MitM between the original accessory and the victim. The victim and his iOS device see only one accessory, which is the MitM controller.

The MitM controller implements a WAC device to communicate with the victim's iOS device and uses an internal wireless network to communicate with the original accessory. To keep the stealth of the attack, the attacker sets the internal network to be hidden (by not broadcasting its SSID), so the victim cannot notice it. The pairing process of the original accessory and the MitM controller is performed during this phase. This pairing ensures that the original accessory is not advertising itself and thus cannot be noticed by the victim. To ensure the continuation of the attack even after a factory reset to the accessory by the users, the attacker programs the MitM controller to pair with the accessory every time it gets reset, and to then advertise itself as the original accessory.

The attacker also sets up the MitM controller and the original accessory as a single physical device. Figure 3.2 illustrates this setup, where the attacker first unpacks the original accessory package and then connects his MitM controller to the original accessory. Once the MitM controller pairs with the accessory, it becomes an admin for the original accessory. The attacker then hides his controller inside the original accessory, where it cannot be spotted. He then puts the gift accessory back in the accessory's original package and wraps it as a gift.

In the remainder of this chapter, we refer to the constructed accessory as the gift accessory or the rogue accessory interchangeably and to its two sub-devices as the original accessory and the MitM controller.

3.1.3 The Gifting Phase

In the gifting phase, the attacker gives the gift accessory to the victim. The gift receiver sees an original HomeKit device package and has no reason to suspect an attack or a forgery.

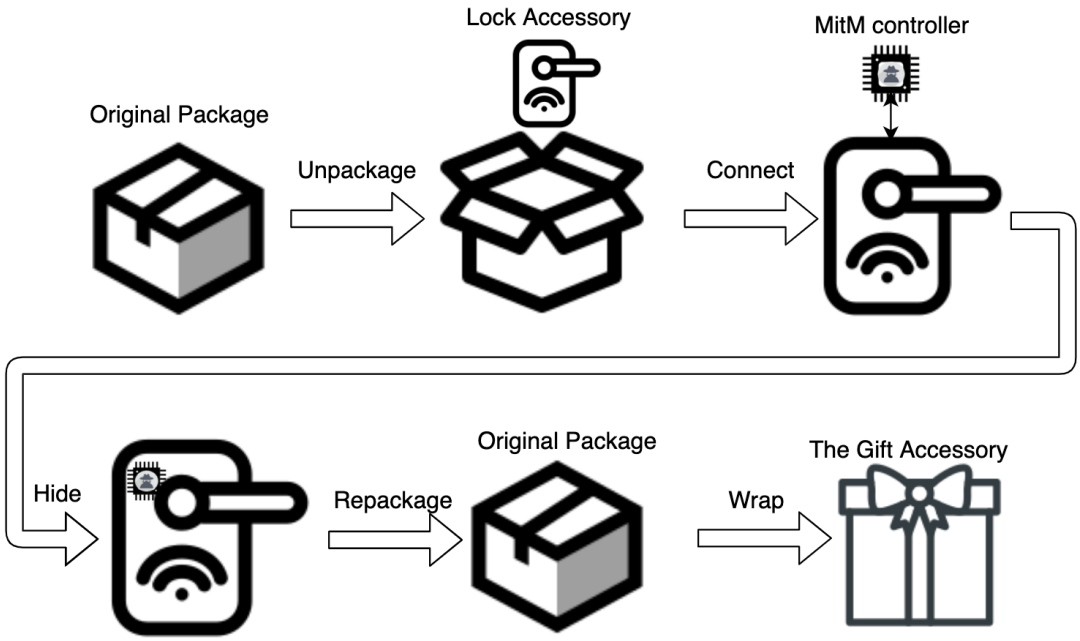


Figure 3.2: The Physical Setup Phase

3.1.4 Installation Phase

Once the victim opens the gift accessory and powers it on, he uses the Home app on his iOS device to add it to his home. When the victim instructs his iOS device to add the accessory, his iOS device configures the accessory to connect the home network (discussed in Section 2.3.1) and then pairs with it (discussed in Section 2.3.2). In the configuration process and the pairing process, the iOS device creates a secure channel with the gift accessory. The MitM controller was given the setup code of the original accessory by its operator at the setup phase, and it can use it to create the secure channel with the iOS device.

After creating the secure channel, the iOS device performs a certification validation and requests the token from the accessory. The MitM controller establishes a connection with the original accessory, requests its token, and relays it to the iOS device. Since the token is a certified accessory's token, the rogue accessory passes the certification check and continues to get the home Wi-Fi credentials. The certification validation step in the gift attack is summarized using the following steps and is illustrated in Figure 3.3.

1. The iOS device requests the accessory's token from the gift accessory.
2. The attacker redirects the request to the original accessory.
3. The original accessory responds with its token.
4. The attacker redirects the token response to the iOS device.
5. The iOS device sends the token to Apple for validation.

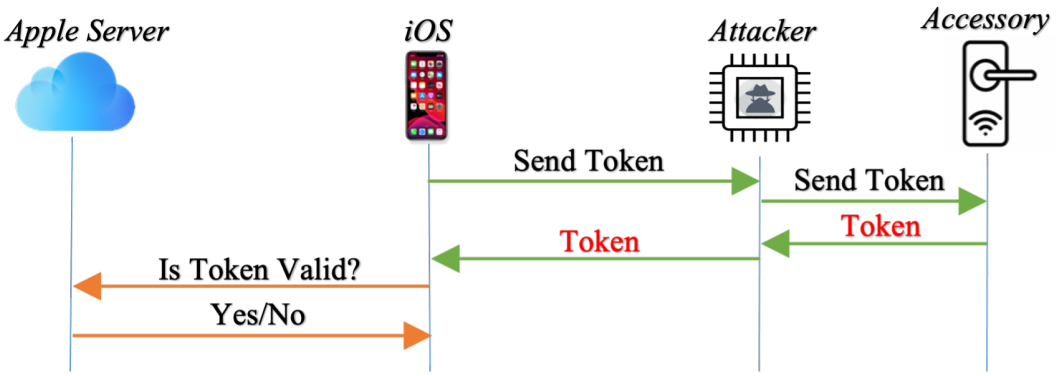


Figure 3.3: The Installation Phase

6. Apple responds with the token status.

Once the certification validation is successfully completed, the iOS device sends the home Wi-Fi credentials to the gift accessory, and the attacker uses them to connect to the victim's home network.

Once the gift accessory is connected to the home network, the iOS device starts pairing with it. During the pairing process, the iOS device updates the gift accessory with a new token, where the rogue accessory saves it in its local memory for future use (e.g., when the victim re-installs the accessory and the new token is needed).

Once the entire pairing process is complete, the MitM becomes part of the home network and is recognized as an Apple Home accessory. The MitM has an internal connection with the original accessory (using an internal wireless network) and delivers all commands from the iOS device to the original accessory and all responses from the original accessory to the iOS device.

At the end of this phase, the victim has connected the rogue (gift) accessory to his home. The victim thinks that it is a certified accessory from Apple, though, in reality, it is controlled by the attacker.

3.1.5 The Normal Operations Phase

After connecting the rogue accessory to the victim's home, the victim should be able to control the accessory and use its functionality as expected. Thus, the rogue accessory needs to guarantee the command's execution. The attacker needs to relay the communication between the victim and the original accessory. For this purpose, the MitM controller uses two secure channels. The first one is established with the victim's iOS device, and the second is established with the original accessory. Each session is secured using the Pair Verify procedure as appropriate to that device.

Figure 3.4 illustrates the commands relay, where the iOS device and the MitM controller establish a secure channel using the Pair Verify procedure to agree on an ECDH shared key K_1. The MitM controller and the original accessory also establish a

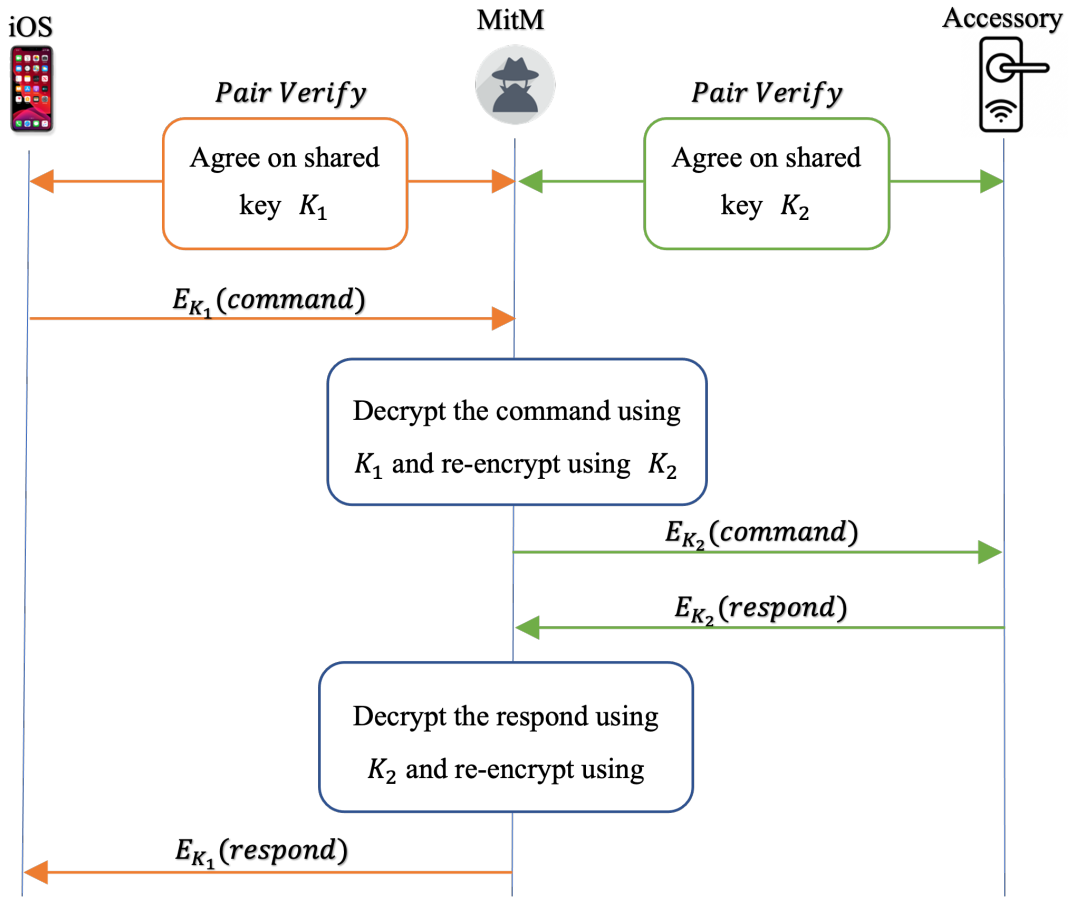


Figure 3.4: The Relay Part of the Normal Operations Phase

secure connection using the Pair Verify procedure to agree on a different ECDH shared key K_2. Then, whenever the victim sends a command to the accessory, the command reaches the MitM controller in the first channel. The MitM controller decrypts it under K_1, re-encrypts it under K_2, and then sends it to the accessory. The accessory performs the commands and sends a response to the MitM controller. The MitM controller decrypts it under K_2, re-encrypts it under K_1, and sends it to the iOS device.

3.1.6 Performing Attacks Phase

The attacker can monitor and control the gift accessory remotely while letting the victim think that he is the only admin of the accessory. For this purpose, the attacker uses a command and control (C&C) server and an Out-of-Band channel to send commands to the MitM controller. The MitM controller can hide its communication with the C&C from the victim's network, e.g., using a cellular network (in this case, the attacker inserts a SIM card in the MitM controller), or in case of a door lock, a microphone that waits for the phrase "open sesame". When the MitM controller receives such a command, it sends the command to the original accessory through the secure session established between them.

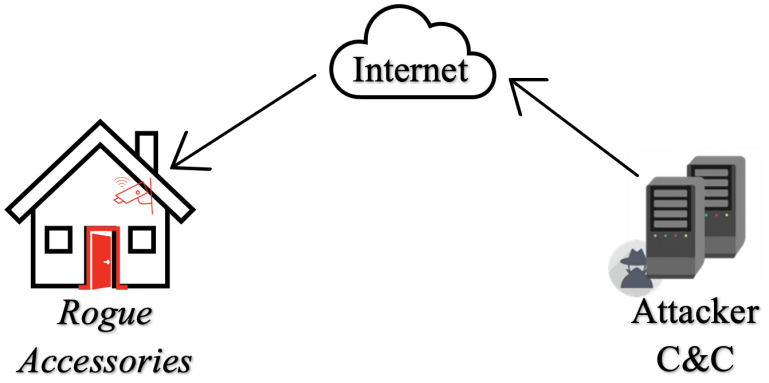


Figure 3.5: Further Attacks Phase Using a C&C

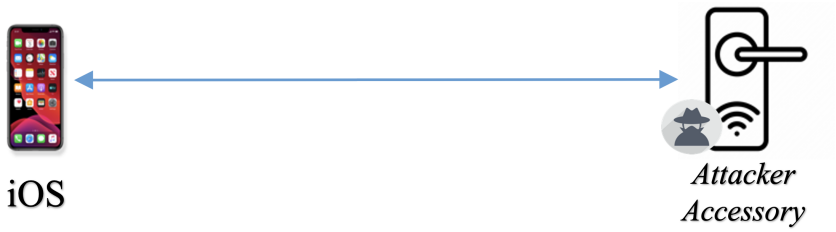


Figure 3.6: The Standalone Flavor

Figure 3.5 illustrates the connection between the rogue accessories and the attacker C&C to receive remote instructions. The rogue accessory can be of any type of accessory, e.g., a security camera or a door lock.

Consider, for example, a rogue security camera in the gift attack. In this example, the attacker can sneak accomplices into the victim's home without being detected by the victim, even if the victim is watching the security camera video feed. To do that, the attacker instructs the camera to alter its video feed and show old footage while the accomplices enter the victim's home.

3.2 An Alternative Construction of the Gift Attack

This section introduces an alternative flavor to perform the gift attack, to which we call the standalone flavor. In this flavor, the attacker creates an accessory that is then used as the gift, and he is not required to include the original accessory in the gift. We also use the standalone flavor to introduce a new attack model, to which we call "Token-Sharing Service", which acts as a certification authority. This service reduces the cost of attacks and makes the attacks scalable and more profitable.

3.2.1 Standalone Flavor

The standalone flavor is performed using one accessory without a MitM controller (shown in Figure 3.6), which the attacker can make. The standalone flavor uses the gift attack phases with few changes. The phases of this flavor are as follows:

1. The devices acquisition phase.
2. Token Acquisition Phase.
3. Setup phase.
4. Gifting phase.
5. Installation phase.
6. Normal Operation.
7. Performing attacks phase.

We describe the phases of the standalone flavor with emphasis on the differences from original gift attack.

3.2.1.1 Acquisition Phase

In this phase, the attacker acquires an impersonating accessory that he either builds on his own, or modifies an existing accessory from a vendor of his choice. He also buys a certified accessory.

3.2.1.2 Token Acquisition Phase

This new phase does not exist in the original gift attack. In the token acquisition phase, the attacker obtains a token that allows the standalone accessory to perform the initial pairing process while looking like a certified accessory.

To extract the authentication token from the certified accessory, the attacker first opens the accessory package and finds its 8-digit setup code (typically printed on an included paper or the accessory's back). Then, he uses the setup code to perform a normal configuration process. He connects to the accessory, creates a secure session, and performs a certification validation step. The attacker requests the accessory's token during the certification validation and receives the desired token. Once the token is received, the accessory is no longer needed.

Figure 3.7 illustrates the token acquisition phase.

3.2.1.3 Setup Phase

The attacker programs the accessory to perform two types of commands: The attacker includes support for HAP commands with the accessory's expected functionality, as well as support for additional malicious functionality for his own instructions. For example, if the gift accessory is a door lock, the attacker acquires an accessory that has the physical lock functionality and programs it to support HAP with the locking functionalities. He also adds his own additional functionality, including, for example, a

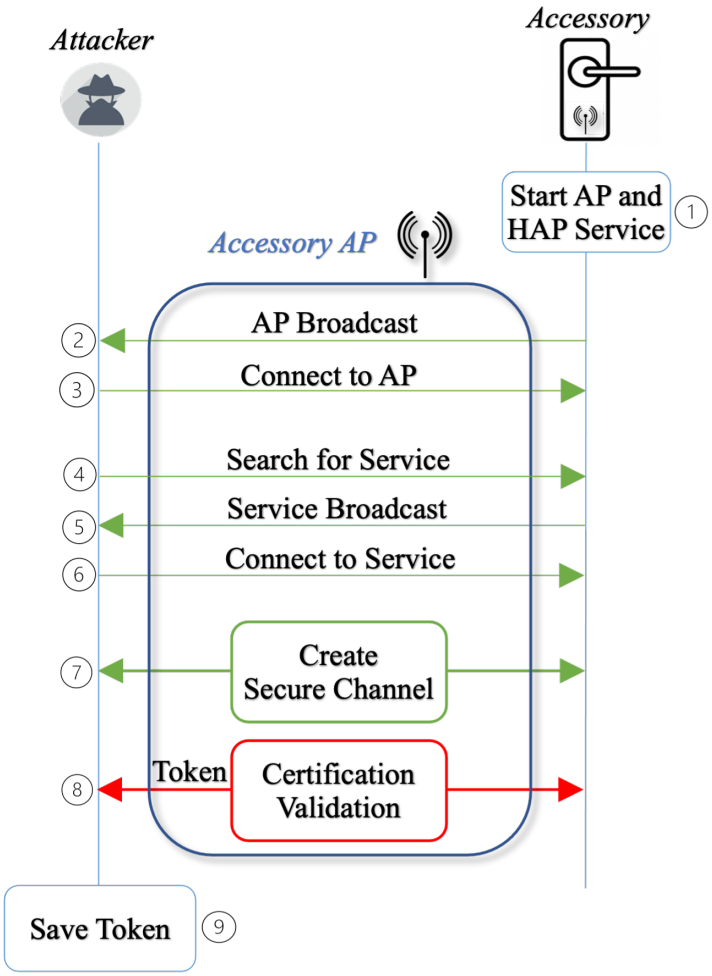


Figure 3.7: Token Acquisition Phase

cellular network for receiving remote instructions and code that receives instructions from the network and locks and unlocks accordingly.

The attacker uses the acquired token that he got in the previous phase, and saves it in the accessory. This token is later used by the accessory to pass the authentication checks when the victim installs it in his home.

3.2.1.4 Gifting phase

This phase behaves exactly like in the original gift attack.

3.2.1.5 Installation Phase

In the installation phase, the rogue accessory implements the WAC functionality. In this scenario, the setup code is chosen by the attacker and hence is known to the rogue accessory. The rogue accessory starts an access point with the required information that identifies it as a HAP accessory with software authentication. When the iOS device asks for the accessory's token, the rogue accessory sends back the token acquired at

the Token Acquisition Phase. The iOS device has no way of telling that the token was stolen from another accessory. Therefore, the pairing completes successfully, and the rogue accessory receives credentials for the home network, and connects to it. Assuming that it correctly implements the expected Apple Home functionality (that it pretends to have), there is no way for the network owner to realize that this is not a genuinely certified accessory.

3.2.1.6 Normal operations phase

In this phase, the accessory acts as a legitimate accessory and performs the victim's commands.

3.2.1.7 Performing attacks phase

This phase behaves exactly like in the original gift attack.

3.2.2 Token-Sharing Service

This section introduces a new profitable and scalable attack model for the Gift Attack. In this model, an attacker builds a token-sharing service that acts as a certification authority that certifies accessories on-demand and enables large-scale attacks. The token-sharing service certifies any uncertified or rogue accessories by providing them with a valid authentication token. Additionally, the service allows the attacker to participate in industrialized hacking by profiting from the token-sharing service.

This attack model is presented in several steps. We start by describing the initialization of the token-sharing service. Then, we explain how to use it in a large-scale attack. Then, we offer a new model where a third-party accessory can also use the attacker's service and get certified. Finally, we introduce a cost-efficient model to operate the token-sharing service.

3.2.2.1 Service Initialization

The token-sharing service includes a token pool that serves tokens to uncertified accessories on demand. The attacker creates the token pool using certified accessories previously acquired by the attacker. As illustrated in Figure 3.8, the attacker acquires several certified accessories, extracts their tokens, and saves them in the token pool. The attacker extracts the accessories' token using the Token Acquisition Phase (discussed in Section 3.2.1.2).

3.2.2.2 Producing Large-Scale Attack

Next, the attacker performs a mass production of rogue accessories that work in the standalone flavor, but without buying a new certified accessory for each rogue accessory and without performing the Token Acquisition Phase. We stress that this production

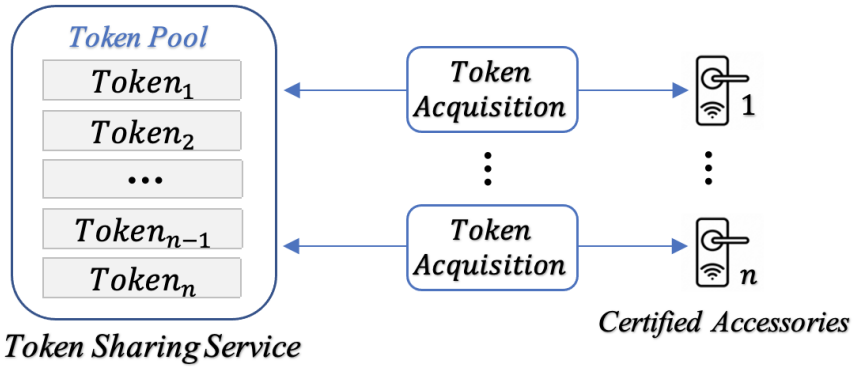


Figure 3.8: Initializing the Service with Valid Tokens

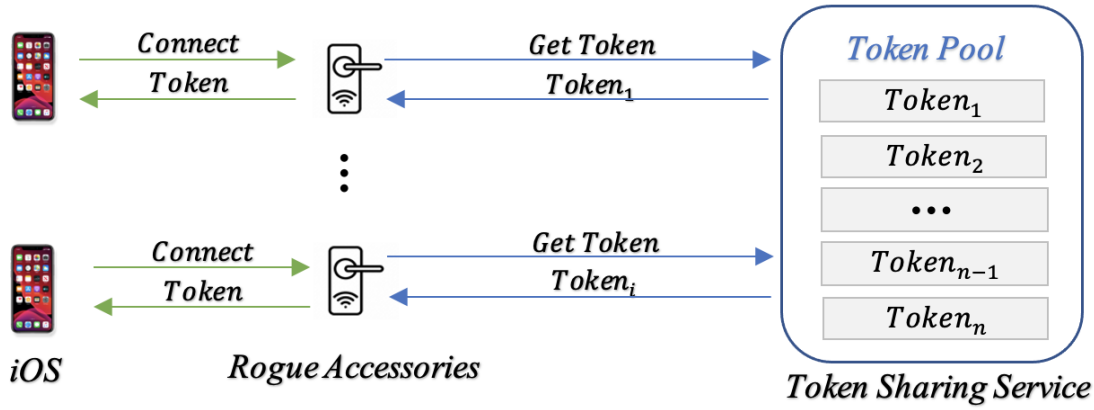


Figure 3.9: Token Distribution Path

does not require prior preparations, as it does not require buying certified accessories nor extracting their authentication tokens. These rogue accessories are then shipped to different users in different homes and countries.

When one of these accessories is activated by a user, it needs an authentication token to successfully connects to the iOS device. As it does not have any authentication token, it contacts the token-sharing service and requests an authentication token. The service extracts one token from the token pool and sends it to the rogue accessory. Then, the accessory presents the token to the user's iOS device. Since it is a valid token, the accessory passes the authentication checks, continues pairing with the user's iOS device, and eventually becomes part of his smart home. Figure 3.9 illustrates the token distribution path. It starts from the token pool where the attacker stores valid authentication tokens, then when the rogue accessory receives the token, it passes it to the victim's iOS device.

3.2.2.3 Certifying Third-Party Accessories

The attacker can use the token-sharing service to serve other malicious actors who wish to certify their rogue accessories. In this scenario, the attacker provides this service to

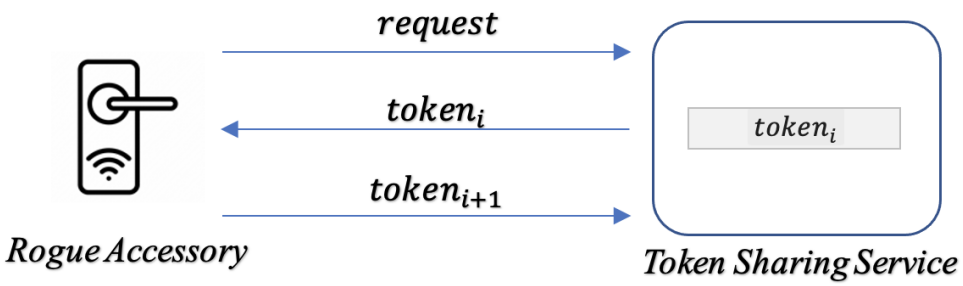


Figure 3.10: Certifying Accessories Using One Token

paying customers that manufacture forged accessories. He either charges a specific price for each requested token to profit from his service or charges for long-term subscriptions. Using such a service, any uncertified accessory can present itself as a certified accessory by simply purchasing an authentication token from the attacker's service.

3.2.2.4 Cost-Efficient Model

The attacker can dramatically lower the cost of this attack. Instead of buying a certified accessory for each rogue accessory, the attacker only buys one certified accessory and uses it to certify many rogue accessories. Recall that when the iOS device activates the rogue accessory, it sends a new valid token back to the accessory. In this model, each rogue accessory that receives a token from the service returns the new updated token that it got from the iOS device back to the service. The service uses the new token to certify another rogue accessory. Figure 3.10 illustrates the protocol between the token-sharing service and the rogue accessory, where each accessory gets a valid token and returns a new valid token to the service.

We illustrate the new efficient model using Algorithm 3.1. In this algorithm, the attacker first extracts an authentication token from a certified accessory, denoted by $token_0$. Then, when an uncertified accessory requests an authentication token, the attacker provides it with the extracted token ($token_0$), the accessory uses $token_0$ and sends a new token $token_1$ to the attacker service. When the next accessory requests a token, the attacker provides it with $token_1$ and receives $token_2$ from it. The attacker receives a new valid token from each accessory to which it sends a token since on every accessory authentication Apple provides the accessory with a new valid token for future authentication. Our attack uses this new token for authenticating other accessories. This way, the attacker's service provides tokens that support a large-scale network of accessories while acquiring only a single certified accessory.

Algorithm 3.1 Token-Sharing Service

Initializing:
$token_0 \leftarrow Token\_Acquisition(certified\_accessory)$
$i \leftarrow 0$
loop
  upon event $< token\_request >$ **do**
  $Send(token_i)$
  $token_{i+1} \leftarrow Receive(new\_token)$
  $i \leftarrow i + 1$
end loop

Chapter 4

Follow-up Attacks on the Apple HAP Protocol

Once the attacker performs the Gift Attack, he can control the installed accessory in the victim's home. The attacker can leverage this capability to launch follow-up attacks on the victim's smart home. These attacks impact smart home security, privacy, and availability.

This chapter introduces diverse and follow-up attacks on the HAP protocol that are launched once the gift attack is performed, i.e., once the rogue accessory is installed. These attacks exploit additional flaws that we discovered in the protocol's design and code. We present four such attacks.

1. Hijacking Siri commands.
2. Hijacking new accessories.
3. Disclosing accessories Status.
4. DoS attack on adding new accessories.

4.1 Hijacking Siri Commands

An attacker that can introduce a rogue Apple Home accessory into a home network can use this accessory to hijack Siri voice commands that are intended for other accessories on the home network. The attack uses specially crafted names for the rogue accessory that takes precedence over the names of the actual target accessories within Siri's parsing algorithm.

4.1.1 Using Siri Voice Commands in Apple Home

The Siri voice assistant is commonly used to control accessories in the Apple Home ecosystem. Siri's vocabulary includes a set of action terms for each type of device. For example, we consider a user's door and describe how Siri is used to close it.

Figure 4.1: Closing Door Lock Using Siri

1. The user activates Siri by saying: "Hey Siri."
2. The user then asks Siri to close the door using a command such as: "Close the door".
3. Siri parses the user's voice command.
4. Siri tries to find the door accessory that has a matching name and a closing capability.
5. Siri sends the found door accessory a "close" command using the HAP protocol.
6. The door receives the HAP command, performs it and responds with an OK message.
7. Siri receives the HAP OK message and informs the user of the command's success using a voice message, such as "Closing the door", "OK, the door is closing" or "Done".

Figure 4.1 illustrates this example.

4.1.2 Siri's Parsing and Searching Weakness

We issued home voice commands using Siri and analyzed its parsing algorithm. The commands structure we used consists of the two parts "$<$ *Action* $><$ *Accessory* $>$", where $<$ *Action* $>$ is a set of the words such as *close, open, lock, unlock, turn on, turn off*, and $<$ *Accessory* $>$ is the target accessory name or type. We analyzed how Siri finds the target accessory in the user home, and based on the results we observed the following behavior:

1. Siri tries to find accessories in the home with a similar name to the target $<$ *Accessory* $>$ name.
2. If multiple accessories match the target name, Siri selects the first accessory in a lexicographical order.
3. If no accessory with a similar name was found, Siri tries to find accessory of type $<$ *Accessory* $>$.

Voice Command	New Accessory Name	Select Reason
“Close the door”	the door	match by name
“Close my door”	my door	match by name
“Close door”	ddoor	match by name
“Close door 1”	door-1	match by name && first in lexicographical order

Table 4.1: Accessory Names to Trick Siri’s Parsing Algorithm

4. If Siri finds more than one accessory with the given type, it informs the user and asks him to choose one.

5. Finally, Siri sends the command to the selected accessory.

After we understood how Siri’s parsing works, we noticed that it could be utilized to trick Siri and force it to choose our accessory, which is different from the one the user intended. Table 4.1 lists several variants of voice commands that close the user’s door. That table presents a new accessory name for each command and why Siri selects the new accessory. The accessory name is a crafted name that manipulates Siri into selecting it instead of the user’s intended one. Select reason column specifies the reason for Siri’s parsing algorithm to choose the new name.

4.1.3 Changing the Accessory Information

According to the HAP specification [15, Section 6.6.2 R2], paired accessories can change their properties. For example, they can:

1. Change the type.

2. Change the name.

3. Register multiple new services.

4. Register multiple new accessories. (Accessories can be composed of multiple accessories, e.g., a surveillance camera that is equipped with a microphone and a doorbell).

As stated in the HAP specification [15, Section 2.3.2.4 R2], services can be hidden from the user. A hidden service contains characteristics that are all hidden. Though hidden services are not shown in the Home app to the user, Siri recognizes them.

When an accessory changes its properties, it informs the iOS device using the mDNS protocol. The accessory sends an mDNS message containing the accessory information, including its configuration number. The configuration number is used to track changes in the accessory properties. Once the iOS device receives a different configuration number from the previously saved value, it fetches the new properties from the accessory and updates the Home app accordingly.

This behavior allows a rogue accessory to change its properties whenever it wants and automatically change the Home database without the user's consent and knowledge. For example, our rogue accessory uses this behavior to register multiple hidden services with crafted names that help us subvert Siri's accessory search algorithm.

4.1.4 The Hijack Attack

To launch the Hijack attack, we use the Gift Attack to introduce a rogue accessory into the victim's home. We utilize two weaknesses: the ability to trick Siri using crafted names and register a new hidden service without the user's knowledge. We describe the Hijack Attack assuming that the user uses the voice command "close the door" (the first command in Table 4.1).

The attack flow is as follow:

1. The rogue accessory registers a new hidden service which has:
   - "the door" as its name.
   - Locking capability characterized as hidden.
2. The user initiates the voice command: "Hey Siri, close the door".
3. Siri parses the user's command and tries to find a target accessory that has a closing capability.
4. Siri's parsing algorithms selects the rogue accessory since it has a service with both "The door" name and the close capability.
5. Siri sends a close command to the rogue accessory.
6. The rogue accessory responds with an OK message.
7. Siri receives the OK message and sends a voice confirmation to the user.

Eventually, the attacker has hijacked a Siri command and left the victim's door open, while the user thinks Siri has closed it.

Figure 4.2 outlines the attack. In this Figure, the attacker has control over a victim's (rogue) light bulb and uses it to hijack a close command to the user's home door as follow:

1. The rogue light bulb registers a new hidden service that has "the door" name and a locking capability.
2. The user says: "Hey Siri, close the door".
3. Siri sends the close command to the light bulb instead of the real door.
4. The light bulb sends an OK message to the iOS device.

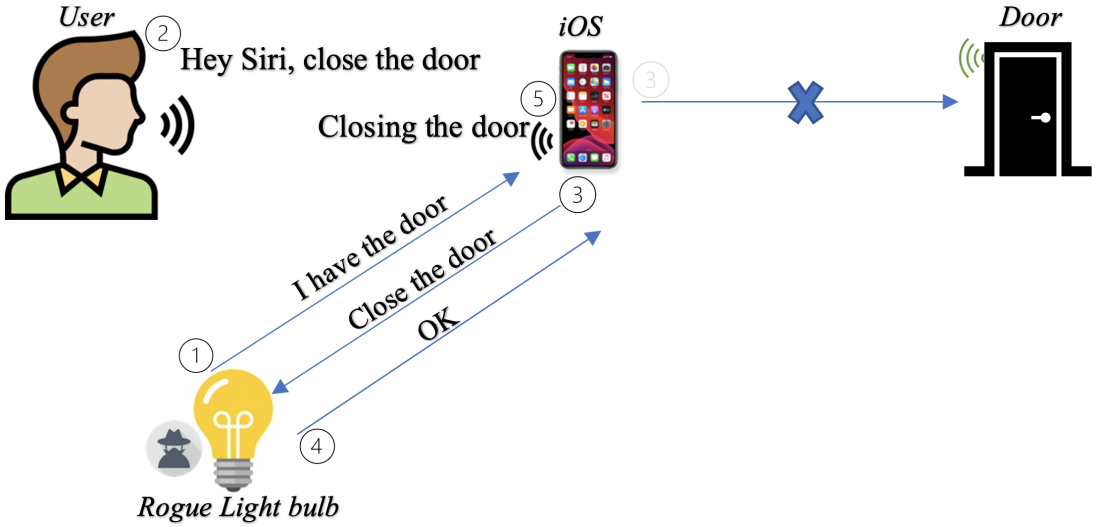


Figure 4.2: Hijack a Close Command

5. Finally, Siri responds with "Closing the door" to the user.

We want to emphasize that this attack becomes possible if the attacker controls a victim's accessory or has installed the accessory in the victim's home, e.g., using the Gift Attack.

Using this technique, we can hijack any Apple Home Siri command. Although user voice commands might be nontraditional and in many variations, the attacker can register many hidden services with different names to cover the various types of commands since there is no limitation to the number of registered services. The attacker can also utilize the gift accessory to spy on the user commands, where he uses a hidden microphone inside the gift accessory to collect voice commands and then craft the names more accurately.

4.2 Hijacking New Accessories

In this attack, the attacker hijacks a newly installed accessory in the victim's home. Hijacking must happen when the victim attempts to connect a new accessory using NFC technology. The stages of such an attack are as follow:

1. Intercept the pairing process of a new accessory and obtain its setup code.

2. Takeover the new accessory using the setup code.

3. Impersonate the original accessory.

4. Pair the attacker's accessory with the iOS device and become MitM.

4.2.1 The Enhanced Setup Code Feature

Before we describe the attack, we describe how the Enhanced Setup Code Feature works. Apple introduced this feature in 2017 to allow users to add new accessories without the need to enter the accessory's setup code manually. Accessories that support the enhanced setup code feature have a QR code or an NFC tag that includes a URL with the enhanced setup code in Base36,[1] and has the following format:

$$\text{X-HM://}\mathbf{Base36(}Ci\_Flags\_SetupCode\mathbf{)}, SetupID.$$

This URL includes two values, but first, notice that the X-HM prefix specifies that the data is intended for the Home App.

1. **Ci_Flags_SetupCode** contains the following data:
   (a) **Ci** is the accessory category identifier as defined in the HAP specification [15, Section 13 R2].
   (b) **Flags**:
      i. BLE and IP support.
      ii. Paired status.
   (c) **setup code** is an 8-digit code used as a password for the SRP protocol to connect the accessory.
2. **setup ID** is a 4-character string used to identify the accessory in the search process.

An attacker who obtains an accessory's enhanced setup code can take control over the accessory using the setup code.

4.2.2 NFC Tags Used in HAP Accessories

This section describes the NFC technology used in HAP accessories and its security risks.

The non-commercial HAP specification does not provide information on NFC tags in HAP accessories or their security requirements. However, Apple's open-source code HomeKitADK contains an implementation for an NFC tag emulator for HAP accessories and provides enough information on this feature.

The implemented emulator emulates a type 2 tag and stores its data in an NFC Data Exchange Format (NDEF). The tag data, in this case, is the enhanced setup code. We also used a HAP accessory [7] which includes an NFC tag and uses Mifare Ultralight C tag (type 2) with an NDEF data containing the enhanced setup code.

Although Type 2 is the most popular tag, it does not provide enough security as it sends its data in plaintext to the controller. This type of tags poses security risks to

[1] Base36() denotes a base36 encoding.

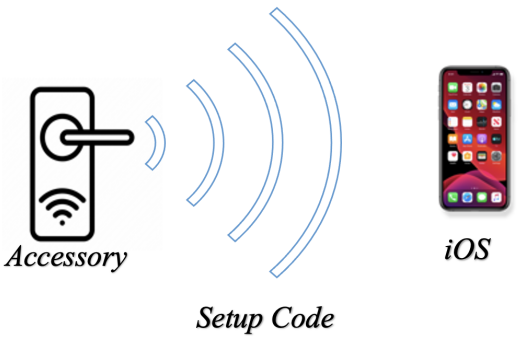


Figure 4.3: NFC Pairing

the home ecosystem since the sent data contains a sensitive setup code. The following section utilizes this vulnerability to launch a hijack attack on newly installed accessories.

4.2.3 NFC Pairing

Accessories with an NFC tag allow seamless connection with the user. From the user's perspective, he only needs to tap the accessory and select it in the Home app to add it. Behind this seamless pairing, an NFC tag transmits data to the iOS device. To understand what happens behind the scenes of this seamless pairing, we look deeper into the NFC communication and conclude the following behavior:

1. The user brings the iOS device near the accessory NFC tag.
2. The tag gets discovered by the iOS device.
3. The iOS device requests the NDEF data.
4. The tag sends its data.
5. The iOS device receives a URL.
6. The iOS device parses the URL, uses the prefix to find the intended application and uses the rest of the data to determine the accessory information.
7. The iOS device shows a pop-up for the user presenting him with the discovered accessory.
8. The user chooses to add the accessory to his home.
9. The iOS device decodes the enhanced setup code in the URL and uses the values to start the pairing process with the accessory.

As illustrated in Figure 4.3, the accessory transmits the enhanced setup code when the iOS device is near it.

4.2.4 The Hijack Attack

At the start, the attacker builds a good RF antenna aimed at eavesdropping on other accessories NFC traffic. The attacker increases the success rate of the eavesdropping by placing the RF antenna inside the gift accessory (introduced into the home network). After that, the attack is performed using the following three stages:

1. The attacker's rogue accessory constantly monitors its environment for NFC transmissions. When a user attempts to pair with a new accessory at home, the rogue accessory picks up the enhanced setup code data sent by the new accessory. This is made possible because the data is transmitted in cleartext. A rogue accessory with a good antenna can pick up NFC transmissions in a 3m radius. Once the rogue accessory captures the data, it quickly moves to the next stage of the attack.

2. Once the attacker obtains the setup code for a new accessory, the attacker can quickly takeover that accessory before it is paired with the user's iOS device. This is made possible because the UI for pairing a new accessory includes multiple steps that require human interaction, providing enough time for the rogue accessory to complete the pair-setup process with the new accessory first. Once the new accessory is paired with the rogue accessory, the user can no longer complete a successful pairing process with it.

3. Once the original accessory is paired with the rogue accessory, the rogue accessory publishes fake mDNS messages on the home network with the setup ID of the hijacked accessory (the rogue accessory is already a member of the home network). When the user attempts to pair with the new accessory, the pairing process is performed between the user's iOS device and the rogue accessory. Since the rogue accessory knows the setup code, it can complete the pair-setup protocol with the iOS device and impersonate the original accessory. The rogue accessory can relay all messages between the iOS device and the original accessory, manipulate some of the commands, drop some of the commands or even send arbitrary commands to the original accessory. The rogue accessory can, of course, manipulate the status information sent from the original accessory to the iOS device.

As illustrated in Figure 4.4 and Figure 4.5, an attacker antenna located 2–3 meters from the smart door is monitoring the environment for NFC transmissions. Once the user attempts to connect the smart door using NFC, the smart door sends the enhanced setup code in cleartext. Then, the attacker sniffs the setup code, uses it to pair the smart door, and then impersonates it. Eventually, this attack allows the attacker to hijack the newly installed smart door without being detected, as the victim thinks that he has installed the smart door as expected in his home. In reality, he is connected to the attacker.

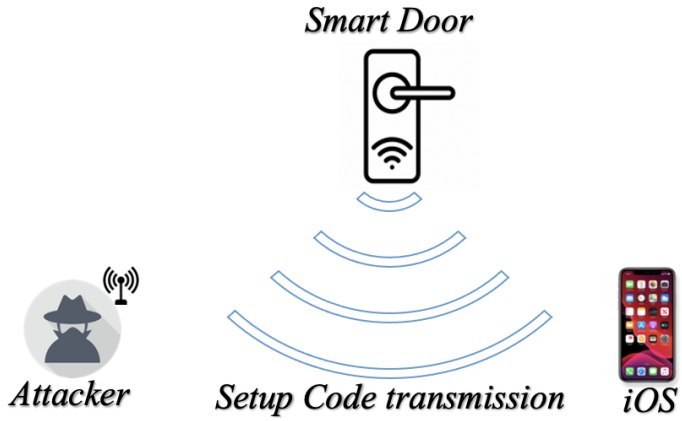


Figure 4.4: NFC Eavesdropping Attack

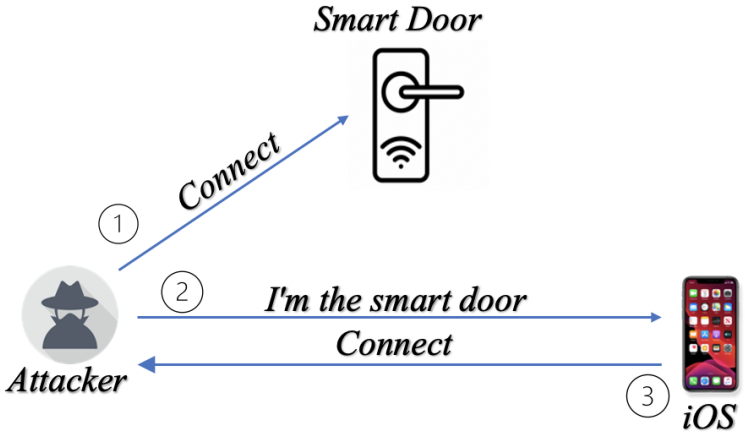


Figure 4.5: Hijack and Impersonate the New Accessory

To summarize the attack, the attacker exploits the ability to eavesdrop on an accessory NFC communication during the pairing to reveal its setup code, which allows him to pair the accessory and control it. The attack is based on the fact that the new enhanced setup code feature used in NFC does not include cryptographic protection against eavesdropping.

4.2.5 Limitations

Note that this attack requires eavesdropping on NFC communicating from a distance. The success of eavesdropping depends on the attacker's antenna distance and environmental conditions. To estimate the required distance to eavesdrop on this NFC communication, we found that in 2011 research [11] Gerhard was able to illustrate an eavesdropping attack using commercial RF equipment, which was able to eavesdrop on NFC communication from 2–3 meters distance. Our attack uses the gift accessory to introduce the RF antenna into the victim's home, allowing the attacker to initiate hijacking attacks on accessories in the same room as the gift accessory.

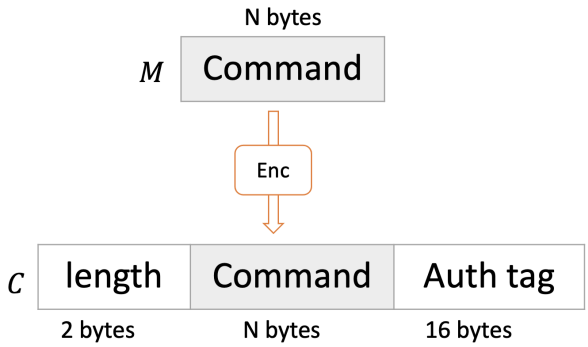


Figure 4.6: HAP Commands Encryption Structure

4.3 Disclosing Encrypted Commands

A rogue accessory introduced into the home network can position itself as MitM for any accessory. While the traffic between the accessories and the iOS device is encrypted, it is possible for a device in this position to understand encrypted commands and extract information about device status.

4.3.1 Commands Structure and Encryption Scheme

Before describing the attack, we describe the command encryption scheme and the command structure. Each new session between an iOS device and an accessory is secured via the pair-verify process, where it establishes a secure communication using the AEAD algorithm *ChaCha20-Poly1305*, that encrypts and authenticates commands between the iOS device and the accessory. *ChaCha20* [19] is a stream cipher, and as illustrated in Figure 4.6 it takes a variable-length message M as an input and produces an encrypted message C of the same length as the message M. The *Poly1305* adds a 16-byte authentication tag to the encrypted message C. According to HAP specification (R2 6.5.2), the encrypted frame has the following format:

- AAD: little-endian length of encrypted data (n) in bytes. (2 bytes).
- encrypted data: according to the AEAD algorithm. Up to 1024 bytes.
- authTag: according to AEAD algorithm. (16 bytes).

Looking at the encrypted frame's length, we can calculate the plaintext message length using the following formula:

$$Length(M) = Length(C) - 18.$$

As the HAP specification states, each command is sent in HTTP requests, and each request's body consists of a JSON object that defines the target accessory. As illustrated in Figure 4.7, each accessory has:

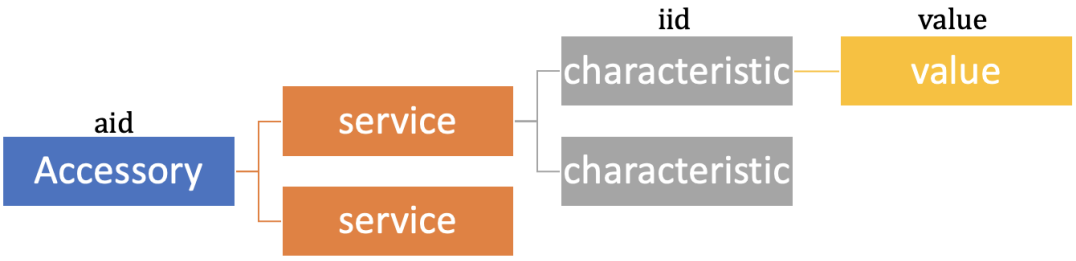


Figure 4.7: Accessory's Fields Structure

(a) Door Close Command (b) Closed Door

Figure 4.8: Closing the Door

- Accessory instance ID (aid).
- Characteristic instance ID (iid).
- Characteristic value.

The aid and the iid values are constants. The accessory sets them once and sends them to the user to respond to a user GET accessories request. Different commands to the same accessory's characteristic have the same payload except for the write value, which contains the accessory's target state. For example, a door accessory uses a *target position* characteristic. The characteristics value ranges from 0–100. To fully close the door, the iOS device sends a PUT request with a JSON body including the value "0", as seen in Figure 4.8 the aid and iid are known and set to 1 and 51, respectively. To fully open the door, the iOS device sends the value "100", as seen in Figure 4.9.

Door commands that partially open or close the door have a value ranging from 1–99. For simplicity of the next discussion, we consider the values 1–9 as almost close commands and 10–99 as a partially open close commands. Figure 4.10 illustrates the command values and their meaning.

If we assume that the length of the close and the almost fully close commands are x-byte long, then the open command is $(x+2)$-byte long, and the partially open/close command (partial) maybe $(x+1)$-byte long. Using the length equation stated previously,

(a) Door Open Command (b) Opened Door

Figure 4.9: Opening the Door

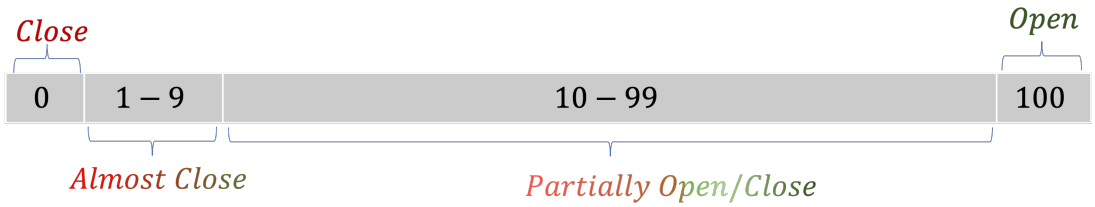


Figure 4.10: Door Command Values

we conclude that

$$length(open) = length(partial) + 1 = length(almost\_close) + 2 = length(close) + 2.$$

The discovered vulnerability can be used to disclose other parameters in the command. For example, if the *aid* length is changing, the attacker can disclose the target accessory, or if the *iid* length is changing, the attacker can disclose the command's functionality. Another type of command uses the same encryption scheme, such as adding new users or removing users, and they can also be vulnerable if they have a parameter with a changing length. The above is out of our attack scope, but it is worth mentioning for future research.

4.3.2 MitM Using mDNS

This section describes how the attacker can position himself as the MitM between the user and his home accessories using the mDNS protocol. In the next section, we explain how to use the MitM in the attack to disclose the status of home accessories. First, being a MitM between the user and his home accessories means having the ability to read, change or drop the traffic between them.

We first describe the normal process where the iOS device connects to the accessory. Then, we explain how the attacker tricks the iOS device to position himself as a MitM. In the normal process, when the iOS device wants to connect to an accessory in the home, it tries to find the accessory address since the accessory's IP address may not be

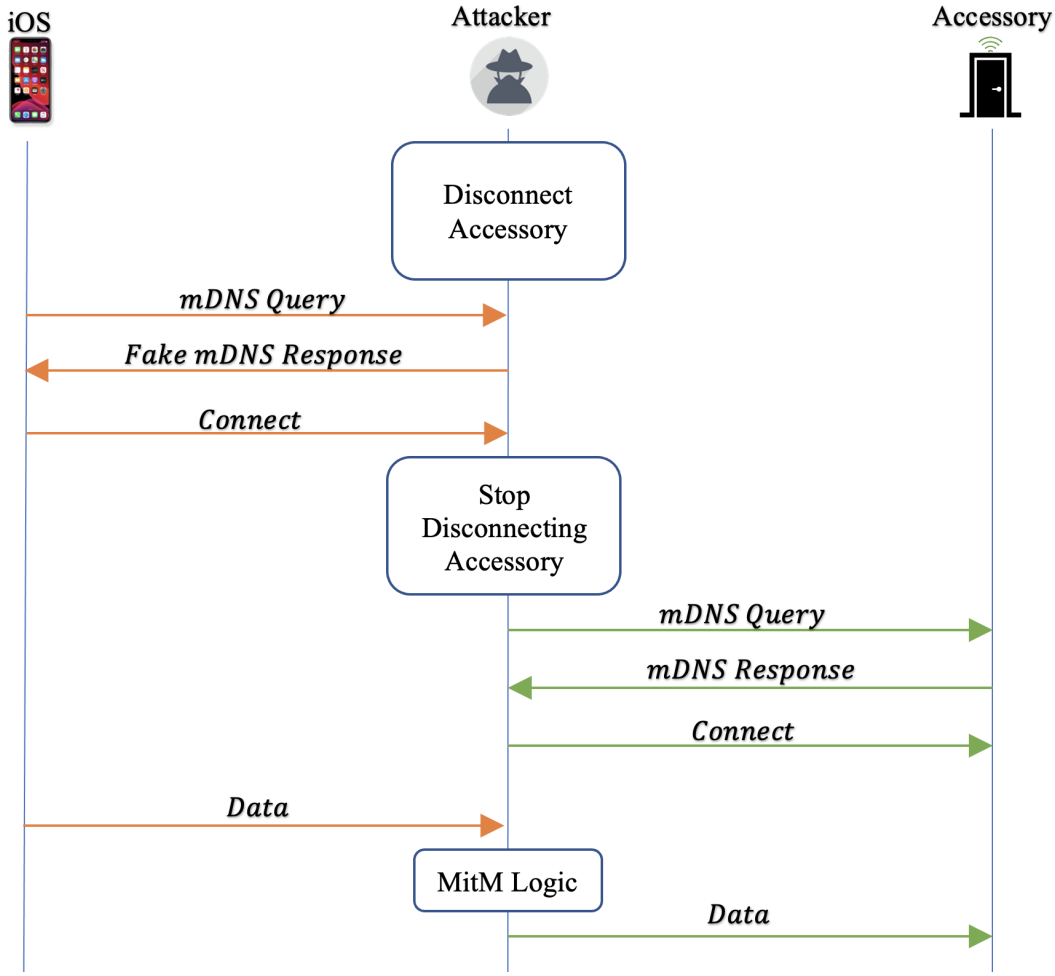


Figure 4.11: MitM Using mDNS

static and can change depending on the home DHCP server. The HAP protocol uses the mDNS protocol to find the accessory address. The connect process is as follow:

1. The iOS device sends an mDNS query (IP multicast message) in the home network.
2. The accessory receives the mDNS query.
3. The accessory sends its information in an mDNS response (IP multicast message).
4. The iOS device receives the mDNS response.
5. The iOS device checks if the received response is from the intended accessory by comparing the accessory ID from its database with the ID in the mDNS response.
6. The iOS device extracts the accessory address (IP, Port) from the mDNS response and connects with it.

Figure 4.11 illustrates how an attacker becomes a MitM using the mDNS protocol. First, the attacker disconnects the iOS device from the accessory to force the iOS device

to perform the connect process. The attacker forces the connect process by disconnecting the accessory from the home network (using Wi-Fi deauthentication attack [18]) and keeps it disconnected until the victim connects to the attacker. When the iOS device sees that the accessory has disconnected, it tries to reconnect with it. It first sends an mDNS query to find the accessory address. Then, the attacker performs an mDNS spoofing and sends fake mDNS responses to the iOS device. These fake responses include the original accessory ID and the attacker's IP address. The attacker knows the accessory ID since it was sent in previous mDNS responses, and it is sent in multicast to all the devices in the network and can be known to the attacker in the network. The iOS device now receives the attacker's mDNS response, validates that the received ID matches the accessory ID, and connects to the attacker's address. The attacker now allows the accessory to connect to the network and receive its real address. Then, the attacker connects to the accessory and starts relaying the user traffic to the accessory. Finally, the attacker acts as a MitM between the user and the accessory, where he can either read the traffic or drop it. He can also become a MitM between the rest of the accessories in the home by repeating this process with every accessory.

4.3.3 Disclosing the Status of Accessories

Now we are ready to describe the attack. Consider an attacker that can eavesdrop on the user's commands using the MitM capability. He can utilize the fact that different commands have different lengths to reveal the command value. For example, the attacker eavesdrops on the user's door commands to monitor a door state until he gets three different commands with different lengths. Using the previous equation, the attacker can also find the length of the original command. The shortest command is the close or almost close command, the next in length is the partial command, and the longest command is the open command. After identifying the length of each command, the attacker can identify the door state according to the last sent command from the user.

We described the attack on a particular accessory. However, the attack applies to any accessory with a characteristic with different value lengths. For example, accessories with a Boolean characteristic have two values, *true* and $false$. The $false$ value is longer than the *true* value in one byte. Thus, the two commands have two different lengths.

4.4 DoS Attack on Adding New Accessories

In this attack, the attacker exploits a bug in the iOS device to perform a DoS attack on the user. The DoS attack prevents the user from adding new accessories to the home. We observed that advertising accessories at a high rate while the iOS device searches for new accessories freezes the Home App. The attack utilizes the ability to advertise HAP accessories in the home using the mDNS protocol, where each mDNS message is handled and parsed by the iOS device and presented to the user in the Home App.

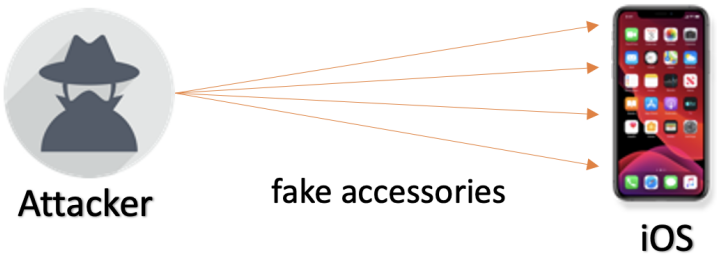


Figure 4.12: Sending Fake Accessories in the Network

Before we describe the attack, we describe the process in which a user uses the Home app to search for new accessories in his network.

1. The user opens the Home App.
2. The user clicks the *Add Accessory* button.
3. The Home app opens the *Add Accessory* window, which opens the camera to scan the accessory's setup code.
4. The user clicks on the "I Don't Have a Code or Cannot Scan" button to search for nearby accessories.
5. The app opens a new search window that presents the user with the existing accessories on the network.

Now that we are ready to describe the technical details of the attack. Consider an attacker in the user's network. Suppose that the attacker knows that the user opened the Home app search window. The attacker starts to advertise fake accessories at a high rate, as illustrated in Figure 4.12, where each accessory is represented by an mDNS response message. Each mDNS response message includes the accessory info as described in Section 2.5. The attacker crafts the mDNS responses such that each response includes a unique accessory ID and sets the accessory's status to unpaired status. The iOS device mDNSResponder daemon receives the attacker's mDNS responses, parses them, and passes them to the Home app, where they are shown to the user as illustrated in Figure 4.13. After a certain number of accessories, the mDNSResponder daemon crashes, and the Home app stops showing any new accessories and stops responding. The iOS device touch screen also stops responding, and the user needs physical buttons to exit from the Home app. When the user enters the app again and tries to add a new accessory again, the add button will not work due to the attack. Thus the victim cannot add new accessories to the home.

4.4.1 Analyzing the Home App Bug

We observed a couple of things regarding the iOS device behavior during the attack:

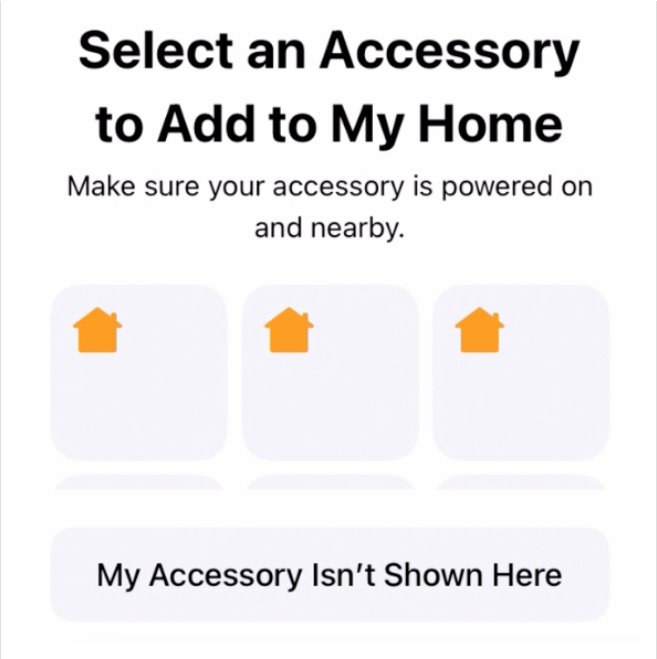


Figure 4.13: Flooding the Home App with Fake Accessories

1. The Home app freezes in less than five seconds from the attack's start.
2. The freeze in the Home app makes the touch screen unresponsive.
3. The mDNSResponder daemon has excessive memory usage, exceeding its 10 MB memory limit.
4. The iOS device jetsam mechanism terminates the mDNSResponder process once it reaches its memory limit.
5. The mDNSResponder daemon keeps terminating and restarting over and over.

Although the attacker can stop the attack, allowing the user to return his iOS device to be responsive, the Home app encounters a bug in the add button that prevents the user from adding new accessories. We observed a couple of things regarding this behavior:

1. The add button in the Home app can stay unresponsive for a long time. For example, performing the DoS attack for 30 minutes left the Add Button unresponsive for more than one day.
2. When the Home app stops responding, it continues to process the previously received mDNS responses from the attacker.
3. The Home app keeps processing the fake responses, and only when it finishes does it allow the user to perform a new add.

Chapter 5

Remote Commands

Many users want to control their home accessories when they are away from home. For example, to check if the home's door is locked after leaving the home or to turn the air condition on before getting back home.

Apple provides its users with the ability to remotely control their home accessories. Users who wish to have this ability install a home hub. A home hub is a device in the home network that connects to the Apple cloud server and the home accessories. It can be an iPad, a HomePod, or an Apple TV. The remote command path is illustrated in Figure 5.1. The command path starts from the remote user. His iOS device sends the command to the Apple server. Apple server redirects the commands to the user's home hub (illustrated in the shape of a HomePod mini), and the home hub sends it locally to the intended home accessory.

This chapter introduces a new attack on the Apple remote command protocol. We first explain how this protocol is designed and, in particular, the Apple push notification service (APNS) protocol responsible for receiving and sending the remote commands. We then introduce our attack on the APNS protocol. It allows a remote attacker to impersonate the user to the APNS server. This impersonation allows the attacker to receive messages intended for the victim and send messages on the victim's behalf. This type of attack can also be applied to other applications that use the APNS service, such as iMessage.

5.1 Apple Push Notification Service (APNS)

The communication between iOS devices and the APNS server is performed over the APNS protocol. iOS devices use the APNS service to send remote messages to other iOS devices or receive messages from other iOS devices. When the user wants to send a message to another user, the iOS device sends the message to the APNS server requesting to send it to the target user's iCloud account. The APNS server receives the message and redirects it to the user's target device.

The Home app uses the APNS server to send home commands from the user's remote

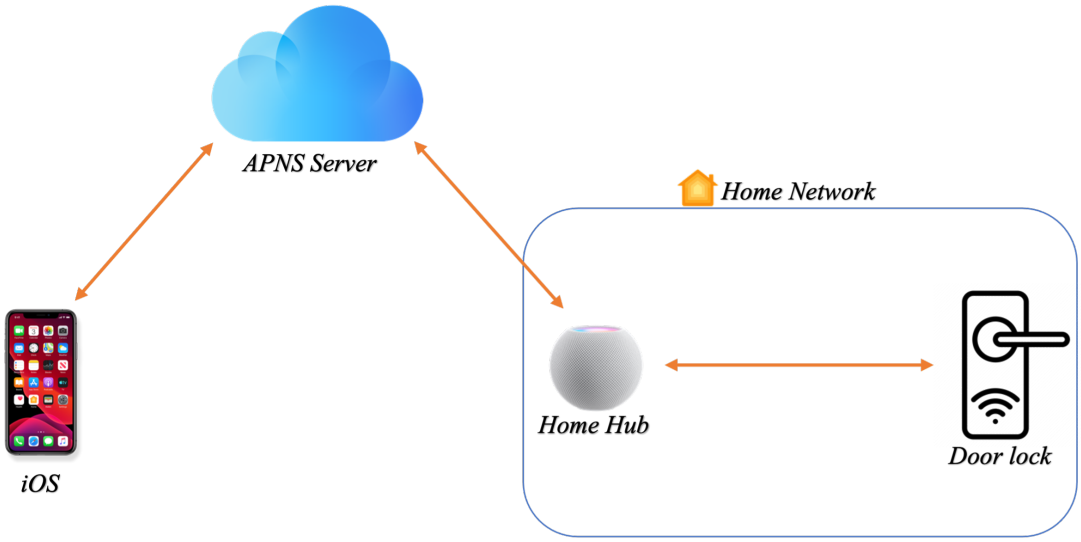


Figure 5.1: HAP Remote Commands Scheme

iOS device to the home hub or send invitations to other users to control accessories in the home.

The APNS server also serves other applications, such as Apple messaging app (iMessage).

5.1.1 Communication with the APNS Server

The APNS server uses the domain name *X.carrier.push.apple.com*, where X is a number from 0 to 255. For example, *22.carrier.push.apple.com*. The iOS device connects to the APNS domain name, and they communicate using the APNS protocol, which is secured by TLS. The iOS device authenticates the APNS server during the TLS handshake using the server certificate, while the APNS server authenticates the iOS device after the TLS handshake. When the TLS handshake is complete, the iOS device sends its certificate and a signature to prove its identity to the APNS server. The APNS server validates the iOS identity, and once it is validated, they can start exchanging messages. When the user wants to send a remote message (e.g., a remote command), he instructs his iOS device to send the message to the APNS server, and in turn, it sends the message to the intended device.

5.2 The APNS Protocol

The iOS devices and the APNS server communicate using the APNS protocol. The APNS protocol is an Apple proprietary protocol, and only proprietary applications, such as the Home app, can use it. Since the protocol was never officially published, we had no choice but to reverse engineer it to understand how it works. Our findings show that the APNS protocol consists of several messages. The main messages are as follows:

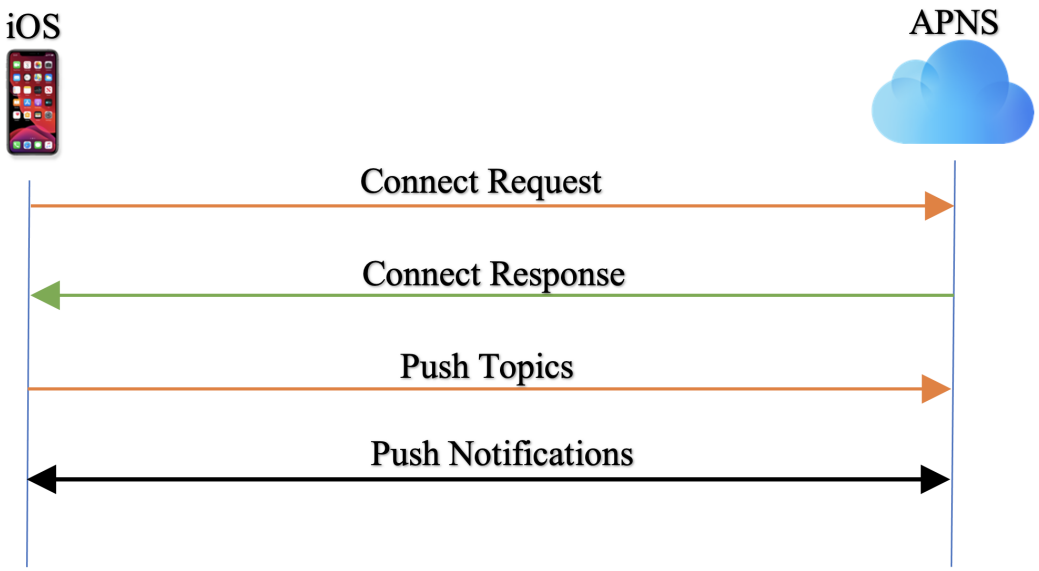


Figure 5.2: Connecting iOS Device to APNS Server

1. **Connect Request**: An iOS device requests to connect to the APNS server and includes a proof of its identity.

2. **Connect Response**: The APNS server responds to the user's connect request (success/failure).

3. **Push Topics**: This message tells the APNS server which services the iOS device wishes to subscribe to.

4. **Push Notification**: This message is used to send a user message to another user. The iOS device of the sending user sends a push notification message (with the user message included in it) to the APNS server, which in turn sends a push notification message to the iOS device of the other user.

5. **Push Notification Response**: Acknowledgment of receiving the push notification message. The receiving user of the push notification message acknowledges receiving it by sending a push notification response, and the APNS server forwards it to the original sending iOS device.

Figure 5.2 shows the process of connecting to the APNS server.

We concentrate on the connect request since it includes the user authentication data. By reverse engineering the iOS device code, we found out that the connect request includes the following data:

$$Connect\_Request = Token||Certificate||Timestamp||Nonce||Signature,$$

where:

- $Token$ – a 32-byte number generated by apple (once the iOS device is activated). Used to identify the device. Notice that this token is different from the certified accessories tokens.
- $Certificate$ – the certificate of the iOS device. Issued by Apple.
- $Timestamp$ – epoch time in milliseconds. Represented by eight bytes.
- $Nonce$ – eight random bytes generated by the iOS device.
- $Signature$ – $Digital\_signature(Timestamp||Nonce)$ using RSA with SHA1. The signature is calculated using the iOS device's private key (that corresponds to the certificate's public key).

When the user logs in to his iCloud account in the iOS device, the iOS device registers the device's token and certificate in the APNS server. Once registered, whenever the user receives a new message through the APNS service, the APNS server redirects it to the registered device.

When the APNS server receives a connect request, it validates it. To validate the connect request, the APNS server performs the following checks:

1. Validate that the token and certificate belong to a registered user to ensure that the user has previously logged in to this device.
2. Validate that the connect request arrival time is less than its $Timestamp + 5$ minutes.
3. Validate the signature, i.e., that the device indeed signed the received timestamp and nonce.

If the connect request is valid, the APNS server sends a success status in the connect response message. Otherwise, it sends a failure status and disconnects the iOS device.

When the iOS device receives a success status in the connect response message, it sends a push topics message. The push topics message contains the services the user wants to subscribe to. One of the services refers to receiving messages from the smart home, which is called the HomeKit topic. In order to receive remote home messages the user subscribes to the Homekit topic.

The iOS device stays connected to the APNS server to receive future messages from other users or devices. The iOS device can now send and receive remote commands through the APNS server using push notification messages.

5.3 APNS Replay Attack

An attacker who wants to connect to an APNS server and impersonate the user cannot generate a valid signature for message with the current timestamp. Only the legitimate

user can generate a valid connect request. However, if we look at the signed data in the request, we can see that the signature protects data that the iOS device itself generates (e.g., timestamp and nonce) and that the signature does not depend on any fresh data from the server. Therefore, the connect request is prone to a replay attack.

An attacker who obtains a legitimate connect request from a victim can use it to connect to the APNS server and trick it into thinking that the connection is legitimate. Once connected, the attacker can send and receive remote messages as the victim. Notice that the connect request itself is valid for five minutes from the moment the iOS device generates it. However, once it is used to log in to the APNS server, the attacker can stay connected as long as he wishes (by sending keep-alive messages to the server).

It seems to be a difficult task for the attacker to obtain a valid connect request, since the connect request is protected by a TLS session between the iOS device and the APNS server. We suggest two approaches to obtain a legitimate connect request. The first approach requires a jailbroken iOS devices, and the second is a more complicated approach that works on non-jailbroken iOS devices.

We describe the two methods to obtain legitimate connect requests in the following subsections. We then provide full details of the attack.

5.3.1 Obtaining a Connect Request – With Jailbreak

This technique assumes that the attacker already owns a jailbroken iOS device. In order to obtain a connect request, the attacker uses an instrumentation tool such as Frida [4] on the jailbroken device. Using Frida, the attacker hooks the iOS code and reads the TLS traffic before being encrypted. The following steps describe how the attacker obtains a connect request on a jailbroken device:

1. Use Frida to read the decrypted TLS traffic by hooking the TLS write function ($boringssl\_session\_write()$).

2. Force the iOS device to make a new connection with the APNS server (by closing the currently open connection, e.g., turning the iOS device's network off and then back on).

3. Save the first sent message (the connect request).

The obtained connect request is valid for five minutes from the included timestamp. The attacker can bypass this restriction by changing the time in the connect request to a future time. Notice that the APNS server may reject messages with future timestamps. However, the attacker accomplishes his goal once the messages are prepared for sending by the iOS device, even before the message is actually sent to the APNS server.

The attacker may change the time using one of the following approaches:

1. Change the iOS device time: this change is manually performed and does not require a jailbreak.

2. Change the timestamp before it gets signed: this change requires hooking, which requires a jailbreak.

Both approaches allow the attacker to change the time in the connect request, which makes it valid for his desired attack time.

An attacker who needs multiple connect requests (each with a different timestamp) can use Frida in a different way in order to efficiently obtain many signed connect requests. He can use Frida to call the request's constructor method directly ([$APSCourier\ refreshCertNonceAndSignatureWithServerTime$]) hooking the time function with a modified timestamp, or even construct the connect request directly and call the signing function to sign it. In each option, he can repeatedly obtain a new signed connect request as many times as he wishes to.

5.3.2 Obtaining a Connect Request – Without Jailbreak

The attacker may also impersonate the APNS server, in which case he needs a seemingly looking valid APNS certificate. A common technique is to create a forged root CA and install its root certificate on the iOS device. The root CA also signs a certificate for the impersonated APNS server and gives both the. When the iOS device connects to the APNS server, it actually connects with impersonated APNS server. At the beginning of the connection, the iOS device authenticates the APNS server by its certificate, which is considered legitimate because it is signed by the forged root CA, and the forged root CA is trusted by the iOS device.

The connection (and the authentication) is performed over TLS. TLS supports a technique called SSL pinning that validates particular fields of the certificate in addition to testing the signature. It is used to protect against forged certificates and against forged root CAs. Therefore, we were very interested in knowing whether Apple uses SSL pinning to protect the APNS certificate and whether it protects against our attack.

After investigation, we found that the iOS device indeed uses an SSL pinning policy and validates it when connecting to the APNS server (the technical analysis of the SSL pinning policy is shown in Appendix A). The APNS policy contains the following:

1. The APNS certificate has a chain of trust of length 3.
2. The APNS certificate has Apple as the root CA.
3. The certificate's intermediate CA has the OID: 1.2.840.113635.100.6.2.12.
4. The certificate has the 1.2.840.113635.100.6.27.5.2 OID in the extension with the OID 1.2.840.113635.100.6.48.1.

The existing policy does not validate the certificate's public keys. Thus, an attacker can generate an impersonating certificate that follows the existing policy while including his forged keys. The attacker follows the existing policy using the following steps:

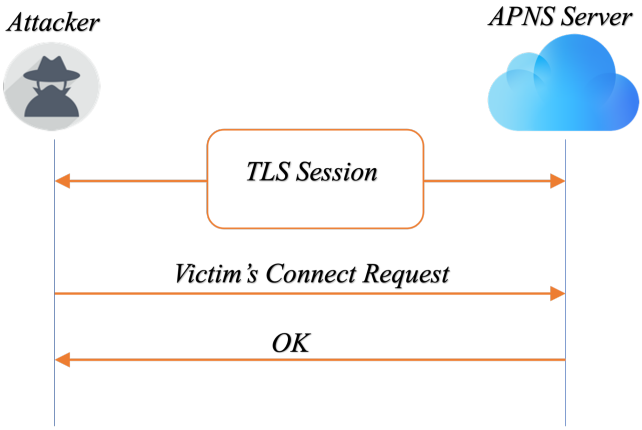


Figure 5.3: Impersonating the Victim

1. The impersonating APNS certificate has a chain of trust of length 3.
2. The rogue root CA in the iOS device has the Apple name.
3. The certificate has the pinned OID and is signed by the rogue root CA.
4. The impersonating certificate has the pinned OIDs and is signed by rogue intermediate.

After installing a rogue root CA on the victim's iOS device and generating an impersonating certificate, the attacker impersonating server can bypass the SSL checks.

The next step is to redirect the victim's iOS device traffic from the real APNS server to the attacker's server. As we mentioned, the iOS device connects to the APNS server on the domain name X.carrier.push.apple.com where X is a number from 0 to 255. There are multiple ways to redirect the traffic. Two of the methods are:

1. ARP poisoning in the user's network, to become a MitM between the victim and his router.
2. DNS poisoning to change the APNS server's IP to the attacker's IP.

Once the iOS device traffic is redirected to the attacker's server, the attacker receives the victim's connect requests. Additionally, if the attacker wants to obtain a valid connect request in some future time, he needs to go to the iOS device setting and manually change its local time to his desired time.

5.3.3 The Attack

In this attack, the attacker uses the obtained connect request to impersonate the victim to the APNS server. Figure 5.3 illustrates the impersonation attack. To start the attack, the attacker establishes a TLS session with the APNS server. The attacker adds a TLS extension that states that the application-layer protocol negotiation (ALPN) is

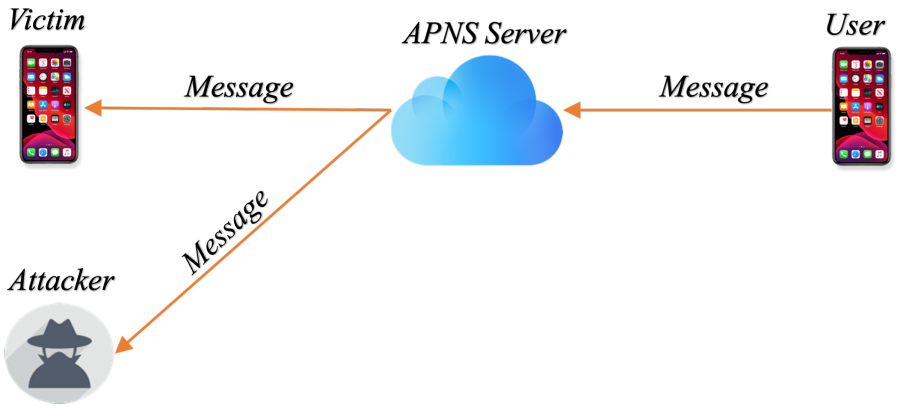


Figure 5.4: Receiving the Victim's Messages

apns-security-v3. Then, the attacker sends the obtained connect request in the TLS session and receives a connect response with success status since the connect request is a legitimate one made by the victim's iOS device. Figure 5.4 illustrates the message paths during the attack. When a user sends a message to the victim, the APNS server redirects this message to the victim's connected devices. However, the attacker is connected to the APNS as the victim. Thus, he also receives the victim's APNS message.

These messages are end-to-end encrypted (as Apple states). Therefore, only the actual user can understand them. However, each message includes the sender's iCloud account in plaintext, and the attacker can use that to identify the sender. Consequently, an attacker can exploit this attack to gather data about the user's activities and use it for many purposes that compromise his privacy.

We did not thoroughly investigate how Apple's end-to-end encryption scheme works and whether an attacker can decrypt or bypass it. However, a study from 2016 investigated the encryption scheme [9]. It shows that each message is encrypted using a symmetric key and authenticated using the sender's private key. The symmetric key is encrypted using the recipient's public key. The encryption scheme does not incorporate any mechanism to prevent replay or reflection of captured ciphertexts. Hence, it is prone to a replay attack. The replay attack can have severe implications on the smart home security since an attacker can exploit it to control the victim's smart home. For example, if the victim sends an unlock command to his home door, the attacker can capture it and use it later to unlock the door.

Even if the end-to-end encryption had been secure against all the attacks, an attacker could still use the APNS attack to compromise the user's privacy by the metadata. For example, he can identify users who have access to the victim's smart home or track people who contact the victim.

We tested this attack on iOS 14.4 (the latest tested iOS version) and the current version of the software of the APNS server.

Chapter 6

Mitigations and Reporting to Apple

In this chapter, we discuss mitigations to the attacks described in this thesis, including the Gift Attack (described in Chapter 3), the follow-up attacks (described in Chapter 4) excluding the DoS attack, and to the APNS attack (described in Chapter 5). Each of the following sections addresses mitigations to one of these attacks and includes Apple's response to our report.

6.1 Mitigation to The Gift Attack

The Gift Attack depends on the ability to extract the accessory's authentication token without breaking into the accessory. Preventing the extraction of the authentication token would prevent the attack. However, it is not easy. Instead, we suggest a change to the used authentication method, and to use one based on a digital signature similar to what was previously used in hardware authentication.

Recall, the digital signature in hardware authentication requires each accessory to have an RSA key with a certificate signed by Apple. The accessory proves to the iOS device that it is certified by giving its certificate and a signature on the ephemeral ECDH keys. The iOS device validates the certificate and the signature before trusting the accessory. In particular, no pairing is performed with an accessory if the signature verification fails or the certificate is not valid.

Our mitigation prevents the gift attack for the following two reasons:

1. The attacker cannot create a valid signature to perform the attack. The attacker needs the accessory's private key to create a valid signature. However, the private key is stored in the accessory's memory and does not leave it. Thus, the attacker cannot have it without breaking into the accessory.

2. If somehow the attacker succeeds in acquiring a signature from a previous session between the accessory and the iOS device, he cannot use it in other sessions since

the signature is on the ephemeral ECDH keys, and the ephemeral ECDH keys change in each new session.

We reported the gift attack to Apple, and they replied with the following statement: "After examining your report we do not see any actual security implications. The behavior you are seeing is expected". Based on their response, we understand that they do not intend to fix this vulnerability.

We observed that a rogue accessory could utilize its ability to access the Internet in order to receive commands from a remote attacker. To limit this ability, we recommend setting a defense mechanism such as a demilitarized zone (DMZ) in the home network to be used solely by home accessories. To set up a DMZ, the home accessories connect to a separate network from the main home network. The separate network is monitored by a firewall that prevents any traffic from the accessories' network to the Internet and vice versa. This DMZ limits the attacker's ability to control the rogue accessories from his remote station and reduces the security implications of the gift attack.

6.2 Mitigation to Hijacking Siri Commands

This attack hijacks Siri commands that are intended to other accessories. The attack exploits the following three vulnerabilities:

1. Home accessories can change their info at any time without the user's knowledge and consent.

2. Accessories can register hidden services that are not shown in the Home app but are still recognized by Siri.

3. Siri can be tricked to send the command to a rogue accessory using crafted names.

Our mitigation strategy to prevent this attack includes three countermeasures that together prevent this attack.

1. The iOS device should inform the user whenever an accessory changes its information and requests his approval. This way, the user can be aware of any suspicious changes.

2. Apple should prevent Siri from sending commands to hidden services that the user does not see in the Home app.

3. Improve Siri's parsing algorithm to parse commands more accurately. One improvement would be to warn the user when his voice command matches two potential accessories and ask him to choose one of them instead of Siri choosing one of them automatically.

We reported this attack to Apple, and they are still investigating it. Until the investigation ends, we recommend that users be careful with using Siri and periodically check the success of Siri commands by checking the accessory's state in the Home app.

When a command hijack occurs, the user may want to search for the rogue accessory that hijacked the command. Since the rogue accessory can use a hidden service, the user cannot see it in the Home app display. There are different ways to search for a hidden rogue accessory. One way is to turn one accessory off at a time, perform the malfunctioning Siri command, and check whether the command hijack stops. The hijack should stop if the rogue accessory is turned off.

6.3 Mitigation to Hijacking New Accessories

This attack is based on the fact that the accessory's NFC tag sends the setup code unprotected. An attacker can eavesdrop on the accessory's NFC transmission and retrieve the setup code. The setup code is then used to hijack the accessory.

Our mitigation includes a change to the accessory's NFC tag to protect against eavesdropping. We can achieve this protection by encrypting the sent setup code. There are many methods to achieve this protection. One of them is using public-key cryptography such as RSA. Using RSA, the iOS device sends its RSA public key to the NFC tag. The NFC tag encrypts the setup code under the iOS device public key and sends it to the iOS device. An attacker that eavesdrops on the NFC transmission cannot decrypt the setup code without knowing the iOS device's private RSA key.

Our suggested method is described using the following algorithm:

1. The iOS device generates RSA keys (private and a public key).

2. The iOS device sends his public RSA key to the NFC tag.

3. The NFC tag uses the iOS device public key and encrypts the setup code.

4. The NFC tag sends the encrypted setup code to the iOS device.

5. The iOS device uses his RSA private key and decrypts the setup code.

We reported this attack to Apple, and they replied with the following statement: "After examining your report we do not see any actual security implications. The behavior you are seeing is expected." Based on their response, we understand that they do not intend to fix this vulnerability. Without Apple fixing this vulnerability, the protocol stays prone to this attack, and users will stay unprotected when using the NFC option. Thus, we recommend suspicious users to enter the accessory's setup code manually instead of using the NFC option.

6.4 Mitigation to Disclosing Encrypted Command

In this attack, the attacker utilizes two characteristics in the encrypted commands to reveal their value. The two characteristics that the attack utilizes are:

1. The command structure, where different commands have different lengths.

2. The encryption scheme, where the length of the plaintext message can be calculated from the length of the encrypted message.

As a mitigation, we suggest a simple change to the commands that unifies their lengths. We suggest that each command uses a value with a fixed size, where leading zeros are added to short values. Thus, different values have the same length, and different encrypted messages have the same length. Alternatively, we can add a padding field to the command that its content's length completes the command to a fixed length.

We reported this attack to Apple, and it is being tracked for a future security update. This update is scheduled for the fall of 2022.

6.5 Mitigation to The APNS Attack

The APNS attack allows the attacker to impersonate the user to the APNS server. This attack is made possible since the user's authentication message (connect request) is prone to replay attack. To lunch the attack, the attacker uses a previously captured connect request, and performs a replay attack.

Our suggested mitigation to this attack includes two changes that together prevent the attack. The first change is regarding the TLS protocol, and the second change is regarding the APNS server logic.

1. Use mutual authentication in TLS: The APNS communication starts with a TLS handshake that only authenticates the APNS server. We recommend using mutual authentication to add authentication to the iOS device in the TLS handshake. During mutual authentication in the TLS handshake, the APNS server proves its identity to the iOS device, and in parallel, the iOS device also proves its identity to the APNS server by providing a certificate and a signature on the TLS session key. The APNS server should validate the certificate and the signature before trusting the iOS device identity.

2. The APNS server should test that the received certificate from the TLS handshake is identical to the received certificate in the connect request.

Using mutual authentication in the TLS handshake prevents the attacker from connecting to the APNS server using a certificate that is not his own. The test that the certificates are identical prevents the attacker from using his certificate in the TLS handshake and using the victim's certificate in the connect request. It also prevents the attacker from

replaying a victim's connect request since our mitigation requires him to use the victim's certificate in the TLS handshake (which he cannot do without the iOS private key).

Apple can prevent special variants of the attack by making some changes in the APNS server, without the need to change the iOS device code, for example:

- If the attacker obtained a connect request that the victim previously used: the APNS server would accept only a single connection per timestamp.
- If the attacker is connecting to the APNS server while the victim is connected: the APNS server keeps one connection with the same token and certificate. The newest connection is probably the attacker's connection. Thus, dropping it would prevent the attacker's connection.
- If the attacker is connecting from the same IP address as the victim: the APNS server would restrict multiple connections from the same IP address and allow only one connection per IP.

Keep in mind that these mitigations on the APNS server would prevent only very simple attack variants. These mitigations are not protecting all variants of the attack since an attacker can bypass these mitigations using more advanced variants (such as obtaining a valid connect request with his desired time, without letting the victim's iOS device to connect to the real APNS server).

We reported this attack to Apple, and they are still investigating it. However, after a couple of months, we observed that Apple has incorporated some of the mitigations we suggested above. Until the investigation ends, and as a temporary countermeasure, users can make it harder for the attacker to obtain a connect request by a few simple steps:

- Put a password on the iOS device to prevent attackers from installing malicious data on it (such as malicious root CA).
- Check for installed root CA certificates installed on the iOS device and remove suspicious ones.
- Make sure the latest iOS version is installed on the iOS device in order to prevent attackers from jailbreaking into the iOS device.

Users that suspect that their iOS device is compromised can reset their iOS device in an effort to stop any ongoing APNS attacks. To reset the iOS device, select Settings $\rightarrow$ General $\rightarrow$ Reset and click Erase All Content and Settings. This button performs a factory reset to the iOS device and removes all the installed data, including malicious data such as a root CA. A new certificate is generated for the iOS device, and the old certificate (that might be compromised) is invalid. Thus, an attacker that has previously obtained a connect request can no longer use it to perform a replay attack since it contains an invalid certificate. Keep in mind that a factory reset is not guaranteed to

stop the attack, especially on iOS devices compromised by an attacker with root access in them. For example, an attacker with root access on an iOS device can make a fake factory reset to prevent this mitigation.

Chapter 7

Conclusions

This thesis shows that smart homes are far less secure than they should be and that even the most secure smart home protocol has many flaws that compromise the home security and its residents. In particular, it shows that Apple's HomeKit Accessory Protocol (HAP) has many vulnerabilities. Millions of users around the world widely use Apple certified accessories. They are used in almost every home appliance, such as home locks, security systems, or surveillance cameras. Certified accessories are considered secure as they are tested and licensed by Apple. In contrast, uncertified accessories are not approved nor tested by Apple, and installing them triggers security warnings to the users.

Apple's users trust Apple and its security mechanisms, and the home ecosystem is not an exception. Thus, we first asked the question, "can users really trust a certified accessory?" Unfortunately, the answer is no. Attackers can easily create fake accessories and impersonate certified accessories. We introduced the gift attack that allows an attacker to install fake certified accessories in the victims' home and control sensitive components without the victims' knowledge. The gift attack also gives attackers the ability to launch additional attacks on the Apple smart home ecosystem components and further compromise its security.

We also introduced a token-sharing service that leverages the gift attack. It allows attackers to become online certification authorities for accessories, certifying rogue accessories remotely on demand. This service threatens Apple's certification mechanism and its purpose.

We have reported the discovered vulnerabilities to Apple, along with our suggested mitigations. Apple is planning to fix some of them, some are not scheduled to be fixed, and the rest are still under investigation.

Finally, a new (still under development) protocol, called "Matter", is expected to become the industry-wide smart home standard for all platforms. The protocol is developed as a collaboration of multiple companies, including Apple, Amazon, and Google. Apple stated in WWDC21 [17] that the new protocol leverages the HAP protocol. Hopefully, it will protect against our attacks and ensure the security of smart

homes.

Appendix A

APNS SSL Pinning Policy

This appendix provides technical details of the SSL pinning policy in iOS devices. The attacker then applies the policy to a fake SSL certificate to perform the APNS Replay Attack in Section 5.3.2 without jailbreak to the iOS device. The analysis includes two parts:

1. Finding the policy.

2. Understanding the policy.

During the TLS handshake, the iOS process *apsd* asks the process *trustd* to evaluate the certificate used by the host "courier.push.apple.com." *trustd* uses the received host domain name to find its pinned policy.

The pinning policies are found in sqlite3 DB, pinningrules, found in the path: "/Library/Keychains/pinningrules.sqlite3". The pinningrules DB has a rules table that saves each host name's policies. To get the host policy, the iOS performs the following SQL query:

```
SELECT DISTINCT policyName, policies, labelRegex
 FROM rules
 WHERE domainSuffix="push.apple.com".
```

The query result is shown in Figure A.1, and the policy is a plist containing the data

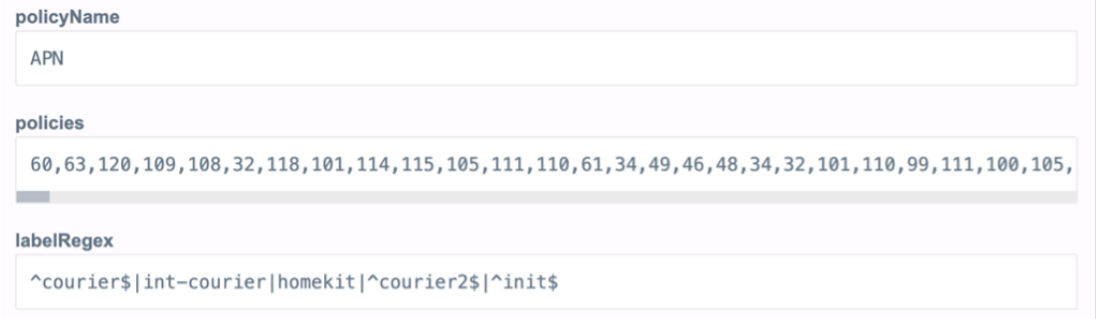


Figure A.1: APNS Policy Query Result

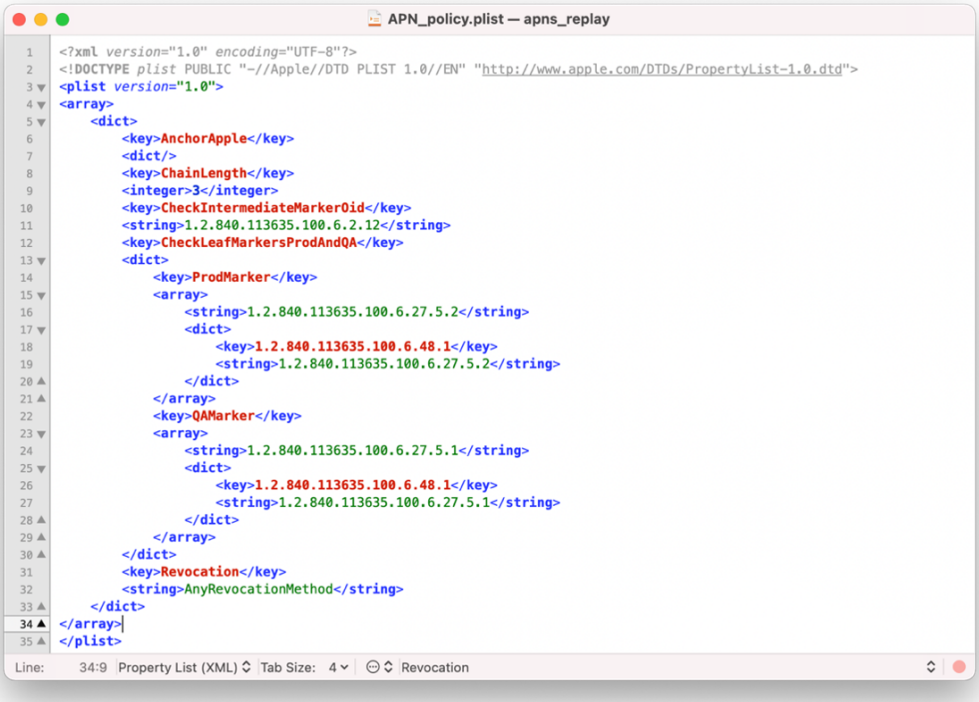
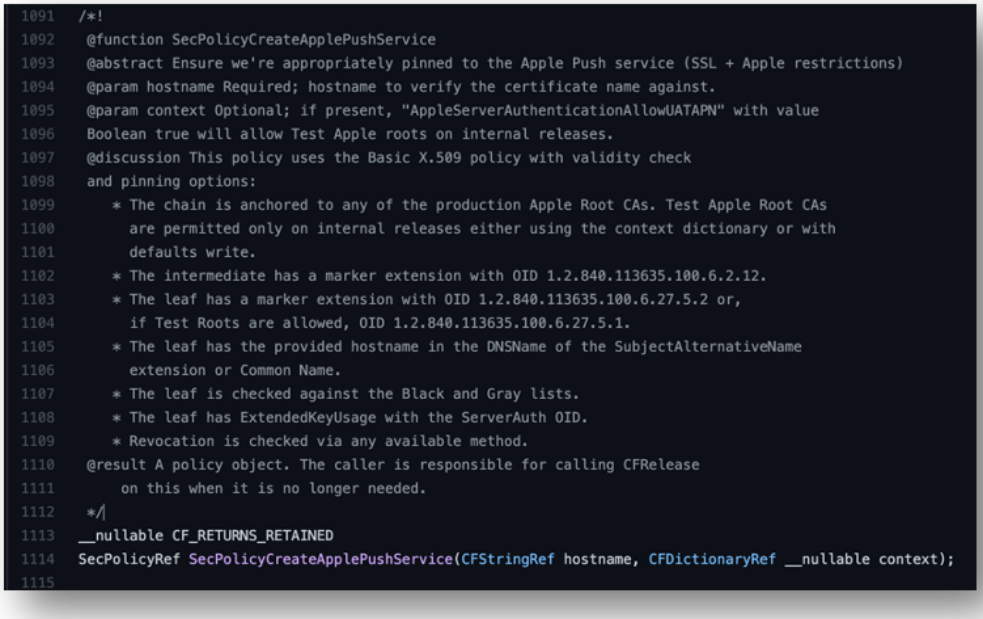


Figure A.2: APNS Policy – From DB

shown in Figure A.2.

We can also see what the policy means by looking at Apple's policy found on GitHub and shown in Figure A.3. To summarize, the APNS policy includes checks for the following:

1. The APNS certificate has a chain of length 3.
2. Apple is the root CA.
3. The intermediate CA has the OID: 1.2.840.113635.100.6.2.12.
4. The certificate has: 1.2.840.113635.100.6.27.5.2 OID in the extension with an OID 1.2.840.113635.100.6.48.1.



```
1091  /*!
1092   @function SecPolicyCreateApplePushService
1093   @abstract Ensure we're appropriately pinned to the Apple Push service (SSL + Apple restrictions)
1094   @param hostname Required; hostname to verify the certificate name against.
1095   @param context Optional; if present, "AppleServerAuthenticationAllowUATAPN" with value
1096   Boolean true will allow Test Apple roots on internal releases.
1097   @discussion This policy uses the Basic X.509 policy with validity check
1098   and pinning options:
1099      * The chain is anchored to any of the production Apple Root CAs. Test Apple Root CAs
1100        are permitted only on internal releases either using the context dictionary or with
1101        defaults write.
1102      * The intermediate has a marker extension with OID 1.2.840.113635.100.6.2.12.
1103      * The leaf has a marker extension with OID 1.2.840.113635.100.6.27.5.2 or,
1104        if Test Roots are allowed, OID 1.2.840.113635.100.6.27.5.1.
1105      * The leaf has the provided hostname in the DNSName of the SubjectAlternativeName
1106        extension or Common Name.
1107      * The leaf is checked against the Black and Gray lists.
1108      * The leaf has ExtendedKeyUsage with the ServerAuth OID.
1109      * Revocation is checked via any available method.
1110   @result A policy object. The caller is responsible for calling CFRelease
1111       on this when it is no longer needed.
1112   */
1113  __nullable CF_RETURNS_RETAINED
1114  SecPolicyRef SecPolicyCreateApplePushService(CFStringRef hostname, CFDictionaryRef __nullable context);
1115
```

Figure A.3: APNS Policy – From GitHub

Bibliography

[1] CVE-2017-13903. Available from MITRE, CVE-ID CVE-2017-13903., August 30 2017.

[2] CVE-2017-2434. Available from MITRE, CVE-ID CVE-2017-2434., January 1 2017.

[3] CVE-2020-9978. Available from MITRE, CVE-ID CVE-2020-9978., March 2 2020.

[4] Ole André. Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. `https://frida.re`, 2013.

[5] Apple. Homekit Accessory Protocol – HAP. `https://developer.apple.com/support/homekit-accessory-protocol`, 2014.

[6] Don Bailey. Sneaky Element – Real World Attacks Against Secure Elements. `https://www.forbes.com/sites/thomasbrewster/2018/04/26/skeleton-key-exploits-apple-mfi-trust/?sh=1996944d503c`, 2017.

[7] Inc. Belkin International. Wemo Stage Scene Controller. `https://www.belkin.com/us/smart-home/wemo/wemo-stage-scene-controller/p/p-wsc010`, 2021.

[8] Martin Bäckman. Number of Smart Homes. `http://www.berginsight.com/ReportPDF/ProductSheet/bi-sh8-ps.pdf`, 2021.

[9] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. Dancing on the lip of the volcano: Chosen ciphertext attacks on apple iMessage. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 655–672, Austin, TX, August 2016. USENIX Association.

[10] Google. OpenWeave Protocol. `https://openweave.io`, 2013.

[11] Gerhard P. Hancke. Practical Eavesdropping and Skimming Attacks on High-Frequency RFID Tokens. *Journal of Computer Security*, 2011.

[12] Keen Home. Certification. `https://medium.com/@KeenHome/what-it-takes-to-be-homekit-compatible-c253496e79d9`, 2017.

[13] Apple Inc. Introducing WAC in WWDC13. https://developer.apple.com/videos/play/wwdc2013/700, 2013.

[14] Apple Inc. MFi-Program for C ertification. https://mfi.apple.com, 2014.

[15] Apple Inc. HomeKit Accessory Protocol Specification (Non-Commercial Version). https://developer.apple.com/homekit/specification/, 2017.

[16] Apple Inc. Open-Source Version of HomeKit Accessory Development Kit. https://github.com/apple/HomeKitADK, December 2019.

[17] Apple Inc. Worldwide Developers Conference 2021. https://developer.apple.com/wwdc21, 2021.

[18] Chintan Kamani, Dhrumil Bhojanian, Ravi Bhagyoday, Vivek Parmar, and Deepti Dave. De-Authentication Attack on Wireless Network. *International Journal of Engineering and Advanced Technology (IJEAT)*, 8, February 2019.

[19] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, RFC Editor, May 2015.

[20] Espressif Systems. ESP8266, Integrated Wi-Fi MCU for Iot applications. https://www.espressif.com/en/products/socs/esp8266, 2021.

[21] Thomas Wu. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol. *IEEE P1363*, October 2002.

הבית, ואפילו לנעול את תושבי הבית בתוכו. לחילופין, ביצוע ההתקפה על מערכת האבטחה של הבית מאפשרת לתוקף לכבות אותה ולפרוץ לתוך הבית.

ההתקפות שאנו מתארים אינן מסתפקות רק במנעולים ומערכות אבטחה. בעזרת התקפת מתנה והתקנת אביזר זדוני בתוך בית הקורבן מתאפשרות לתוקף התקפות נוספות על פרוטוקול האפ. למשל השתלטות על אביזרים חדשים בתוך בית הקורבן, השתלטות על פקודות סירי שמיועדות לאביזרים שונים בתוך הבית, חשיפת פקודות מוצפנות שנשלחות לאביזרים שונים בתוך הבית, והתקפת מניעת שירות על האייפון שמונעת מהמשתמש להוסיף אביזרים חדשים לבית. כך ההתקפות שלנו מאפשרות גם שליטה מלאה על בית הקורבן, כולל שליטה על כלל האביזרים שמותקנים בו, כל המידע השמור בו, ריגול כנגד הבית ותושביו, ועוד.

אפל מספקת למשתמשים שלה את היכולת לשלוט בבית החכם שלהם מרחוק בזמן שהם לא נמצאים בבית. אנחנו מראים שמנגנון השליטה מרחוק חשוף גם הוא להתקפה, שבה אפשר להתחזות למשתמש מול אפל. בעזרת התקפה זו תוקף יכול להשיג מידע על המשתמשים בבית החכם של הקורבן, וגם לשלוח פקודות לצורך שליטה בבית החכם בלי להיות בקרבת הבית.

בנוסף, אנחנו מציעים הגנות כנגד ההתקפות שכוללות שינויים בפרוטוקול ובאפליקציה של אפל.

דיווחנו לאפל על החולשות שמצאנו. חלק מהחולשות עדיין בבדיקה על ידם וחלק כבר נבדקו. אפל דיווחה לנו שהיא מתכוונת לתקן חלק מהחולשות שנבדקו, אך אינה רואה סיכון באחרות, שאותן היא לא מתכוונת לתקן. אנו מקווים שבעתיד אפל בכל זאת תתקן אותן לצורך יצירת מערכת יותר מאובטחת לבתים החכמים.

תקציר

פרוטקול אפל הומקיט לאביזרים (HAP-האפ) מאפשר למכשירי אייפון לחבר אביזרים לבית המשתמש ולשלוט בהם בצורה מאובטחת. רוב האביזרים שנמכרים מאושרים ע"י אפל, לאחר שהם עוברים בדיקות אבטחה שאפל הגדירה. אפל גם תומכת באביזרים לא מאושרים ומאפשרת להם להתחבר למכשירי אייפון, אבל כאשר משתמש רוצה לחבר אביזר לא מאושר לבית שלו האייפון מציג לו התראת אבטחה שמתריעה למשתמש שהמכשיר לא מאושר ושהוא לא נבדק וייתכן שלא יעבוד בצורה מאובטחת. המשתמש חייב לאשר את הוספת המכשיר הלא מאושר כדי להוסיף אותו לבית. בחיבור זה נראה כי מנגנון אימות האביזרים מאושרים בפרוטוקול האפ אינו בטוח, ומאפשר לאביזרים זדונים שאינם מאושרים לעבור את האימות.

פרוטוקול האפ משתמש בשתי שיטות אימות לצורך אימות אביזרים מאושרים, אימות חומרתי שהיה האימות המקורי של האפ, ואימות תוכנתי חדש יותר שמיושם באביזרים החדשים. באימות תוכנתי האייפון מאמת את האביזר באמצעות טוקן שמיוצר ע"י אפל. כדי להוסיף אביזר חדש לבית, האייפון מבקש את הטוקן מהאביזר ושולח אותו לאפל לאימות. אפל מחזירה לאייפון תשובה אם הטוקן חוקי או לא. אם הטוקן אינו חוקי האייפון לא מאפשר לאביזר להתחבר לבית. כך אפל מבטיחה שאביזרים זדוניים או אביזרים לא מאושרים לא יכולים להתחזות לאביזר מאושר, ושהוספתם לבית תקפיץ התראת אבטחה למשתמש.

בחיבור זה אנחנו מציגים התקפה חדשה על אימות אביזרים מאושרים. בהתקפה אנחנו מנצלים את היכולת לחשוף טוקן של אביזר מאושר ומנצלים את העובדה שהטוקן אינו מוגן קריפטוגרפית. אנחנו קוראים להתקפה החדשה שלנו "התקפת מתנה". בשונה מהתקפת "האלמנט הערמומי" של דון ביילי המאפשרת לתוקף לקבל את סיסמת הרשת בעזרת התחזות לרכיב מאושר חומרתי, ההתקפה שלנו מנצלת חולשה באימות התוכנתי בפרוטוקול עצמו כדי לחבר אביזר זדוני לבית הקורבן, ואינה מסתמכת על היכולת לתקוף את הרכיב עצמו.

בהתקפה שלנו, התוקף נותן לקורבן כמתנה אביזר זדוני שמתחזה לאביזר מאושר. האביזר הזדוני שנשלט ע"י התוקף מצליח לעבור את האימות ומצליח למנוע מהאייפון להתריע למשתמש שהאביזר לא מאושר. ההתקפה שלנו מהווה איום משמעותי על אבטחת בתים חכמים מכיוון שאפשר לבצע אותה על כל אביזר, אפילו אביזרים שמספקים אבטחה לבית. למשל, ביצוע ההתקפה על מנעול דלת הכניסה של הבית מאפשר לתוקף שליטה על כל מי שנכנס ויוצא מהבית, מאפשר לו להכניס פולשים לתוך

המחקר בוצע בהנחייתם של פרופסור אלי ביהם ועמיחי שולמן בפקולטה למדעי המחשב.

תודות

אני רוצה להודות למנחיי, פרופ' אלי ביהם ועמיחי שולמן, שהנחו אותי במקצוענות רבה ותמכו בי לאורך העבודה על התזה. כמו כן, אני רוצה להודות למשפחתתי היקרה על האהבה והתמיכה.

אני מודה לטכניון, למרכז המחקר לאבטחת סייבר ע"ש הירושי פוג'יוארה ולמערך הסייבר הלאומי של ישראל על התמיכה הכספית הנדיבה בהשתלמותי.

שונא מתנות יחיה:
תקיפת בתים חכמים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

אשרף יאסין

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
שבט תשפ"ב חיפה ינואר 2022

שונא מתנות יחיה:
תקיפת בתים חכמים

אשרף יאסין