Wireless Networks
https://doi.org/10.1007/s11276-023-03376-8

Research on smart-locks cybersecurity and vulnerabilities

Cándido Caballero-Gil[1] · Rafael Álvarez[2] · Candelaria Hernández-Goya[1] · Jezabel Molina-Gil[1]

Accepted: 2 May 2023

Abstract

Smart-locks have become increasingly popular for access to homes and businesses in many countries, because of their ease of use and adaptability. These locks offer a simple and secure alternative to traditional key-based entry, making them an attractive choice for both residential and commercial properties. Nevertheless, it is essential to acknowledge the potential security threats that come with any new technology. The security of smart-locks is particularly critical, as a breach could result in unauthorized entry. Since the smart-locks can connect, there are different ways to check if vulnerabilities can be found easily or on the contrary, if the security level is high. Two of the main ways of checking the security level of this kind of IoT device are the information that can be obtained from the Android application and the security level of the Bluetooth connection. Many vulnerabilities can be found in the Android smart lock management application. This application is very useful to perform all the configurations with such a lock, but if it is not properly implemented and secured, it can provide clues for malicious users to perform unauthorized access to the system. Another security factor is the Bluetooth connection. This ensures that only authorized users have access to the property. In this work, we have analyzed the security level of different parts of smart-locks. In particular, we have analyzed the security of the applications for the most important smart-locks on the market. This study reveals relevant information such as whether the application is obfuscated or not, the encryption algorithm for the Bluetooth connection, or relevant URLs that applications use to connect to the cloud. The security of the Bluetooth connection between the smartphone application and two selected smart-locks was also analyzed. It was demonstrated that if no encryption is used for the Bluetooth connection, the smart-lock is not secure, but if AES encryption is used, the security level is high.

Keywords Smart-lock · Cybersecurity · Cyber-attack · Ethical hacking · Threat modeling · Bluetooth · Android app reversing

1 Introduction

Smart-locks (see Fig. 1) have been made possible by technological progress, and they offer greater protection for both homes and businesses. These locks allow doors to be opened or unlocked using various methods, such as a mobile device, remote control, NFC card, fingerprint, or numeric code. Not only can these locks provide different access options, but they can also be considered smart since they can store and retrieve an access log, and communicate and sync with calendars to manage access based on predetermined criteria. Furthermore, some of these locks even allow door access by entering a code upon touching the door.

However, as with any new technology, there are always potential risks and vulnerabilities that must be considered. Here we will discuss some of the cybersecurity risks

✉ Cándido Caballero-Gil
ccabgil@ull.edu.es

Rafael Álvarez
ralvarez@ua.es

Candelaria Hernández-Goya
mchgoya@ull.edu.es

Jezabel Molina-Gil
jmmolina@ull.edu.es

[1] Ingeniería Informática y de Sistemas, Universidad de La Laguna, San Cristóbal de La Laguna, S/C de Tenerife, Spain

[2] Department of Computer Science and Artificial Intelligence, Universidad de Alicante, Alicante, Spain

Published online: 27 May 2023

Fig. 1 Smart-locks

associated with smart-locks, and how to mitigate them. One of the most common risks associated with smart-locks is that of hacking [1]. If a hacker is able to gain access to the lock's security, for example, by capturing packets with the Wireshark tool [2] and a Bluetooth Sniffer, they could potentially unlock the door without the owner's permission. To mitigate this risk, it is important to ensure that the lock is sufficiently secure and that only authorised users have access.

Another risk that needs to be considered is that of physical manipulation. Smart-locks are typically installed on the exterior of doors, making them more susceptible to tampering. If someone were to physically tamper with the lock, they could potentially bypass the security measures and gain access to the property. To mitigate this risk, it is important to choose a smart-lock that is tamper-resistant and has a robust physical security design.

Finally, it is also important to consider the risk of human error. While smart-locks are designed to be user-friendly, there is always the potential for user error. For example, if a user forgets to lock the door, or fails to properly secure the lock, this could lead to a security breach. To mitigate this risk, it is important to educate users on the proper use of smart-locks, and to ensure that they are aware of the importance of security.

1.1 Smart-locks

Smart-locks are a type of electronic lock that can use an encrypted keypad, Bluetooth device, smartphone, WiFi Bridge, or other methods to grant access to a locked door. They offer a number of advantages over traditional mechanical locks, including the ability to add or remove users without re-keying the lock, and the ability to receive notifications when someone accesses the locked door. There are different smart-locks, but in this work, we have concentrated on locks that do not require any modification to the door mechanism. It is not necessary to disassemble the original lock because it is adhered to the door by a very strong 3M adhesive or screws (see Fig. 2).

Its installation is very simple. It adapts to the cylinder that the user has in the door and one of the keys is used inserted in the smart-lock to open the door, the key could also be used to open if the door has a double clutch cylinder, while the lock is installed, for its correct operation. It is glued directly over the one that is currently available.

In particular, smart-locks that have been analyzed physically in this work are the following:

1.1.1 Nuki smart-lock

The Nuki smart-lock [3] has been gaining considerable popularity in recent years. Nuki stands out for its security and easy installation. The Nuki lock is a motorized lock whose motor turns the key automatically.

The Nuki lock was the first smart-lock in Europe that opens doors with the help of a smartphone. In addition, it is the first most flexible lock, i.e. it can be installed on almost all existing European locks.

Nuki is supported by a physical button, as well as an app control, a keypad or a remote control, and has other features that allow it to be defined as smart. It is compatible with both horizontal and vertical locks and is designed to be strong enough to work with multi-point locks. It also doesn't matter how many turns the lock has. Nuki is designed to work with all models. Nuki has a remote, a keypad, a WiFi bridge (available separately), or the smartphone app that can be linked to the Nuki smart-lock to allow you to open your door (see Fig. 3).

It has other functions such as assigning secure access permissions via the application and digital key sharing. These functions are practical for occasions when you want to give someone access to your home without granting them permanent access. For example, you can give permission for the cleaner to access your home between 08:00 and 12:00 every Wednesday for example.

You can also control who has entered and who has left through the Activity Log. This log shows which family members have access to the house, as well as any other activity that has taken place at the door. It lists how the door has been opened, by whom, and by what method, whether manual, sensor or button, etc. Nuki has an API connection with tourist platforms like Airbnb or Booking. In this way, it generates a temporary code that can be issued to users to prevent keys from being lost or copied. This lock allows several users to register at the same time, sending them an invitation code, which can be withdrawn as soon as desired. The reason to choose this smart-lock was that this device is one of the most famous in the European market.

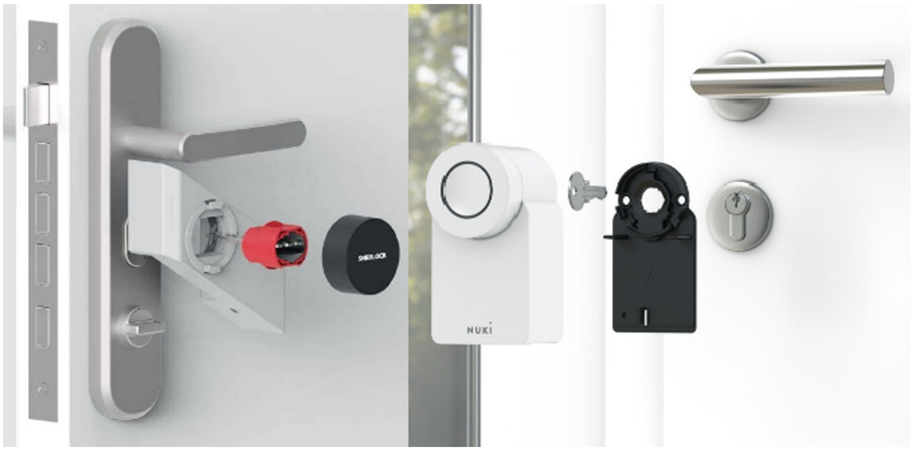


Fig. 2 Smart-locks physical mechanism

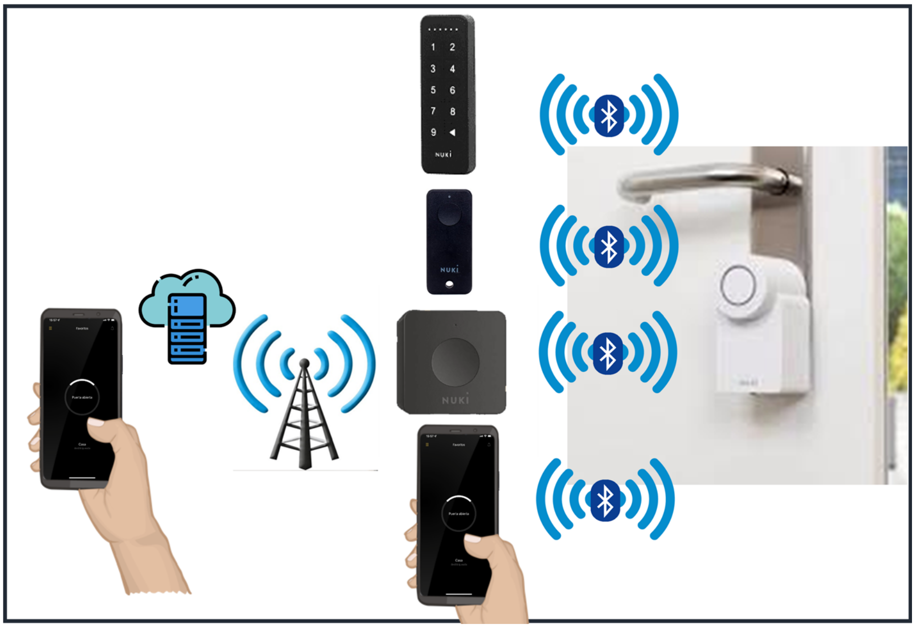


Fig. 3 Nuki smart-lock connection scheme

1.1.2 Sherlock S2 smart-lock

Xiaomi's Sherlock S2 [4] was one of the best-selling and best-rated locks on the market. Probably because it was one of the cheaper and easier to install. This is the second lock to be analyzed in this work. The Sherlock lock has multiple ways of unlocking. Using the original key, a SmartKey (a remote control that is configured from the mobile app to operate the lock), fingerprint unlocking (sliding the finger on the side of the lock) and, finally, using the Sherlock mobile app. The reason for choosing this smart-lock was the lack of security information from the manufacturer. Note that nowadays, this smart-lock and the associated Android app are not being maintained by its manufacturer.

2 State of art

In the state of the art of this work, we analyze papers related to smart-lock security, from Bluetooth security, mobile application security, secure smart-lock design, as well as general Bluetooth security issues. For example, authors in [5] present an approach for deobfuscating Android APKs based on probabilistic learning of large code bases. The experimental results indicate that DeGuard is practically effective: it recovers 79.1% of the program element names obfuscated with ProGuard, it predicts third-party libraries with an accuracy of 91.3%, and it reveals string decoders and classes that handle sensitive data in Android malware.

Paper in [6] demonstrates several practical attacks based on the threat models toward August smart-lock including handshake key leakage, owner account leakage, personal information leakage, and denial-of-service (DoS) attacks. Authors in paper [7] propose several defenses that mitigate different attacks. One of these defenses is a novel approach to securely and usefully communicate a user's intended actions to smart-locks, which was prototyped and evaluated.

Authors in [8] analyzed a realistic smart-lock solution and identified the main requirements that access control systems for IoT should satisfy. In their approach, an initial blueprint for developing access control mechanisms for edge-cloud-enabled IoT was incrementally extended to incorporate new access control capabilities. Paper [9] evaluates the security of the Verisure smart-lock system by identifying and attempting to exploit potential vulnerabilities using threat modeling and penetration testing. It concludes that the system under consideration is relatively secure. Authors of [10] cover a security assessment of a smart-lock, focusing on the firmware of the embedded devices as the main assets. Based on the identified threats, penetration tests are conducted to demonstrate the security of the firmware. The results show that the firmware could not be obtained and that the product constitutes a good example within consumer IoT for how to manage the firmware of embedded devices. Paper [11] presents an efficient access control scheme for smart-lock based on asynchronous communications. The proposed scheme avoids the break-in. They also present a lightweight and efficient tree-based access control solution. The experiment results prove that our scheme provides higher security and it requires low calculation load and storage resources, ensuring it can be implemented in IoT devices.

A smart door lock that can be completely monitored and controlled from a remote location using an Android application on a smartphone is proposed in paper [12]. Paper [13] outlines various features that are currently available in smart locks. Considering the different versions discussed, it presents a comprehensive understanding of how the smart-lock system enhances the security of the home or workplace by providing hassle-free access from anywhere.

Artificial intelligence and Internet of Things (AIoT) technologies were used to develop an integrated system for remote room authentication and power management. Through hardware and software integration, the study in [14] created the AIoT smart-lock, promoted the use of IoT devices in relevant industries, and enhanced the competitiveness of product research and development. Paper [15] conducts a case study on three different Bluetooth smart devices. They show how these devices can be attacked and abused to not work properly. They also present a vulnerability that is due to the behavior of BLE smart devices and the Just Works pairing mode. This vulnerability can be exploited to generate an attack that affects BLE availability. The authors propose a solution to mitigate the attack. Paper [16] analyzes the security of the BLE link layer, focusing on the scenario in which two previously-connected devices reconnect. Authors developed BLE Spoofing Attacks (BLESA). These attacks enable an attacker to impersonate a BLE device and to provide spoofed data to another previously paired device.

The study [17] aims to assess the security of a smart-lock unit. They first present a background including threat modeling and previously found vulnerabilities. Then a methodology section for different attacks performed as well as the results from doing them. Finally, a discussion where they assess the security implications of the results. The conclusion to be drawn from the results is that the smart lock has weaknesses in its design. Specifically, its file system encryption, resistance to disruption attacks, and consistency of access granted to guests. Authors in [18] present a methodology to carry out 'wireless spiking' attacks on smart-lock devices that would allow an unauthenticated adversary to open a lock, without direct physical tampering, through the manipulation of its electrical control circuitry using IEMI. They demonstrate the proposed methodology–reverse engineering, identification of attack points, development of an attack vector, and design and transmission of attack signals–on a commercially popular smart-lock.

The next section presents the attack models that we have performed to achieve the objective of this work. We try to verify the security level of current smart locks and show that such security is sometimes not sufficient.

3 Attack models and use cases

Smart-locks can work in different ways. These locks can use different types of authentication mechanisms such as biometric sensors, key fobs, mobile apps, or PIN codes to verify the identity of the user. However, like any other digital device, smart-locks are vulnerable to cyber-attacks. This paper is centered on the cyber-security of the smart-locks with an Android app, on a smartphone or tablet.

This section presents, firstly, the attack models that can be performed on smart locks and, secondly, the selected attacks performed to check the cybersecurity of some smart locks with different characteristics. Here are some attack vectors for smart locks:

Brute Force Attack In this attack, an attacker tries every possible combination of PIN codes or passwords to gain access in case the smart-lock has a keyboard. If the user has chosen a weak or easily guessable PIN code, the attacker may be able to unlock the smart-lock by trying all the possible combinations.

Bluetooth Hacking Smart-locks that use Bluetooth technology to communicate with the user's smartphone can be vulnerable to Bluetooth hacking. An attacker may intercept the Bluetooth signal and gain access to the smart-lock without the user's permission.

Physical Tampering Smart-locks that use biometric sensors, such as fingerprint or facial recognition, may be vulnerable to physical tampering. An attacker can create a fake fingerprint or face to fool the biometric sensor and gain access to the smart-lock.

Man-in-the-Middle (MITM) Attack In this attack, an attacker intercepts the communication between the user's smartphone and the smart-lock. The attacker can then modify the communication to gain access to the smart-lock without the user's permission.

Malware Attack Smart-locks that use mobile apps for authentication can be vulnerable to malware attacks. An attacker may infect the user's smartphone with malware that can steal the user's login credentials and use them to unlock the smart-lock.

Denial of Service (DoS) Attack In this attack, an attacker floods the smart-lock with a large number of authentication requests, making it unresponsive. This attack can render the smart-lock useless and prevent the user from gaining access to the locked area.

It's essential to use strong authentication mechanisms and keep the smart-lock updated with the latest security patches to prevent these attacks.

After an analysis of the possible attacks that can be performed on smart-locks, in this work we have chosen to perform two different types of analysis. First, an Android app reversing allows us to check the possibility of performing brute force attacks or malware attacks, and can provide information to perform Bluetooth hacking or man-in-the-middle attack. Second, a Bluetooth replication attack can allow us to perform Bluetooth hacking and man-in-the-middle attacks. Physical tampering and denial of service are outside the scope of this work because these types of attacks do not allow access to the protected space.

3.1 Android app reversing

Reversing an Android app can reveal the underlying code and logic used in the app, including its functionality, user interface, and communication with servers or other devices. In the context of a smart-lock app, reversing can provide valuable information about how the app interacts with the smart-lock and its associated services. The first aspects that can be discovered by reversing an Android app are knowing if the app code is obfuscated as well as finding the IP paths that the app connects to.

Here are some more things that can be discovered by reversing an Android app for a smart-lock:

Communication Protocol This information can help identify potential vulnerabilities in the communication protocol or URLs to connect and provide insight into how an attacker could exploit these vulnerabilities.

Encryption Smart-lock apps must use encryption to secure the communication between the app and the smart-lock but not always do it. Reversing the app can reveal the encryption algorithm used and the keys used for encryption and decryption.

User Authentication Smart-lock apps typically require users to authenticate themselves before accessing the lock. Reversing the app can reveal how the app handles user authentication, including the storage and handling of user credentials.

Authorization The access controls used by the app to restrict access to the smart-lock can be revealed. This information can help identify potential vulnerabilities in the authorization process and provide insight into how an attacker could bypass these controls. Even it is possible to find Authorization tokens to the cloud database embedded in the app.

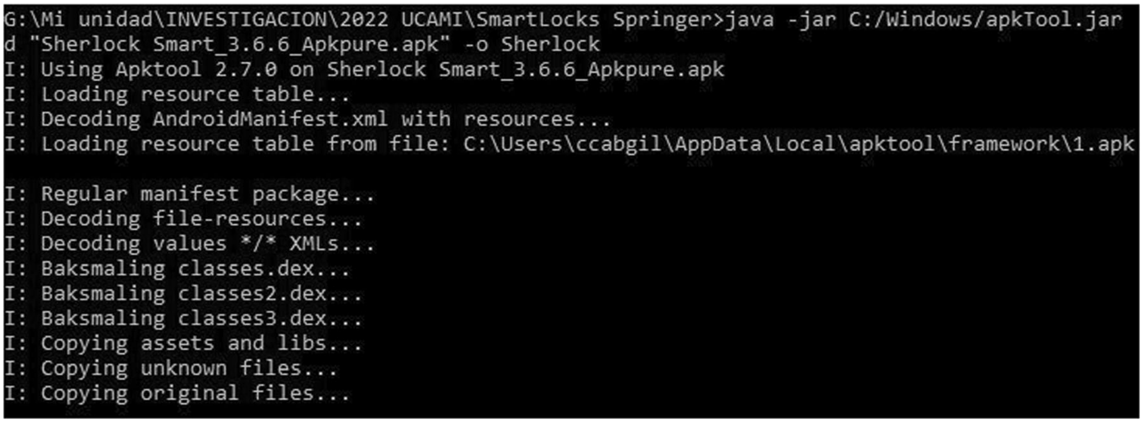
Device Security Smart-lock apps often include features that enhance the security of the lock, such as locking the app with a PIN or fingerprint. Reversing the app can reveal how these security features are implemented and whether they can be easily bypassed.

User Privacy Reversing the app can also reveal how the app handles user privacy, including the collection and storage of user data. This information can help identify potential privacy risks and provide insight into how user data is used by the app and its associated services.

It's important to note that reversing an Android app can be a complex and time-consuming process, and it may require specialized tools and expertise. It's also important to follow ethical hacking practices and not use the information obtained through reversing for malicious purposes.

In order to reverse an Android app, there are some steps that must be done. Firstly, the attacker has to obtain the APK file. It can be obtained from the smartphone by using different apps or downloaded directly from the internet using sites such as APKPure, APKMirror, APKDonwloader, or much more [19]. Once obtained the APK, there are some tools that can help to obtain the code:

ApkTool [20] This tool helps to decompile the application (Fig. 4) to get the SMALI code. Smali, also called baksmali, is a tool that was created to decompile Android executables into a language that could be readable. This way, you can decompile an application, modify it and recompile it to run on Android. The Smali code is reminiscent of assembly language code, but in this case, you see the Java and Android class names and method names. Its syntax is based on the Jasmin and Dedexer languages. In



```
G:\Mi unidad\INVESTIGACION\2022 UCAMI\SmartLocks Springer>java -jar C:/Windows/apkTool.jar
d "Sherlock Smart_3.6.6_Apkpure.apk" -o Sherlock
I: Using Apktool 2.7.0 on Sherlock Smart_3.6.6_Apkpure.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\ccabgil\AppData\Local\apktool\framework\1.apk

I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Baksmaling classes3.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Fig. 4 Android App reversing - Obtaining SMALI code

order to decompile the APK file, the next sentence must be executed.

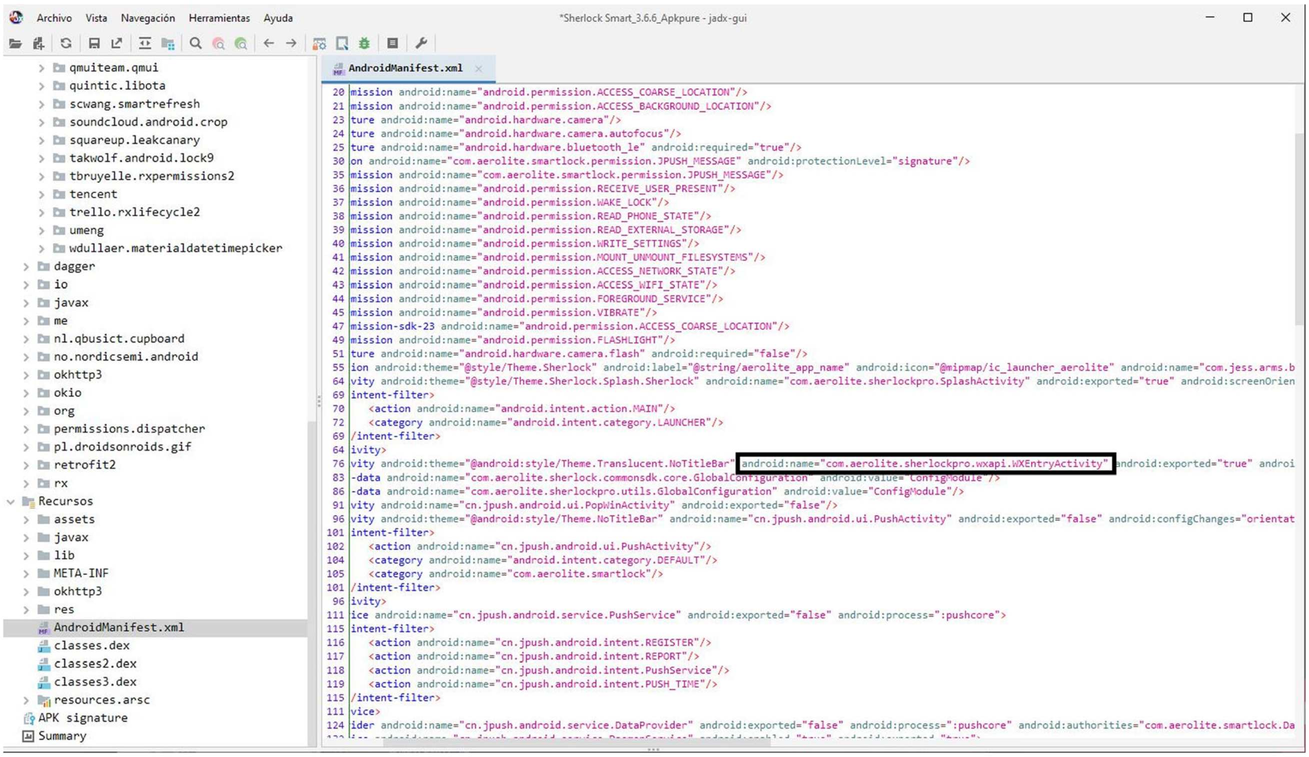
$$java - jarapktool.jard[victimpath.apk]$$
$$- o[pathfromdirectorynametocreate]$$

Jadx-Gui [21] This tool is used to be able to understand the code, in this case, this tool allows you to execute in graphical form, as well as at command level to obtain the Java code (Fig. 5).

Jad-Gui can also help in case the code of the app is obfuscated. It is even possible to modify the SMALI code and recompile the application with a modified behavior, but in this work, we will not perform this part.

Most of these applications are obfuscated. Jadx-Gui allows to de-obfuscated code with limited results. In many of these applications, relevant URLs were easily found. All these applications are complex with more than 20 or 30 permissions required to work. File AndroidManifest.xml shows relevant information such as the Landing Page or the activities. All the applications except the Sherlock lock uses AES encryption to communicate with the smart-lock.

Next Table 1 shows the main Android apps for smart-locks and some information from their code. After



```
mission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
mission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
ture android:name="android.hardware.camera"/>
ture android:name="android.hardware.camera.autofocus"/>
ture android:name="android.hardware.bluetooth_le" android:required="true"/>
on android:name="com.aerolite.smartlock.permission.JPUSH_MESSAGE" android:protectionLevel="signature"/>
mission android:name="com.aerolite.smartlock.permission.JPUSH_MESSAGE"/>
mission android:name="android.permission.RECEIVE_USER_PRESENT"/>
mission android:name="android.permission.WAKE_LOCK"/>
mission android:name="android.permission.READ_PHONE_STATE"/>
mission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
mission android:name="android.permission.WRITE_SETTINGS"/>
mission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
mission android:name="android.permission.ACCESS_NETWORK_STATE"/>
mission android:name="android.permission.ACCESS_WIFI_STATE"/>
mission android:name="android.permission.FOREGROUND_SERVICE"/>
mission android:name="android.permission.VIBRATE"/>
mission-sdk-23 android:name="android.permission.ACCESS_COARSE_LOCATION"/>
mission android:name="android.permission.FLASHLIGHT"/>
ture android:name="android.hardware.camera.flash" android:required="false"/>
ion android:theme="@style/Theme.Sherlock" android:label="@string/aerolite_app_name" android:icon="@mipmap/ic_launcher_aerolite" android:name="com.jess.arms.b
vity android:theme="@style/Theme.Sherlock.Splash.Sherlock" android:name="com.aerolite.sherlockpro.SplashActivity" android:exported="true" android:screenOrien
intent-filter>
   <action android:name="android.intent.action.MAIN"/>
   <category android:name="android.intent.category.LAUNCHER"/>
/intent-filter>
ivity>
vity android:theme="@android:style/Theme.Translucent.NoTitleBar" android:name="com.aerolite.sherlockpro.wxapi.WXEntryActivity" android:exported="true" androi
-data android:name="com.aerolite.sherlock.commonsdk.core.GlobalConfiguration" android:value="ConfigModule"/>
-data android:name="com.aerolite.sherlockpro.utils.GlobalConfiguration" android:value="ConfigModule"/>
vity android:name="cn.jpush.android.ui.PopWinActivity" android:exported="false"/>
vity android:theme="@android:style/Theme.NoTitleBar" android:name="cn.jpush.android.ui.PushActivity" android:exported="false" android:configChanges="orientat
intent-filter>
   <action android:name="cn.jpush.android.ui.PushActivity"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="com.aerolite.smartlock"/>
/intent-filter>
ivity>
ice android:name="cn.jpush.android.service.PushService" android:exported="false" android:process=":pushcore">
intent-filter>
   <action android:name="cn.jpush.android.intent.REGISTER"/>
   <action android:name="cn.jpush.android.intent.REPORT"/>
   <action android:name="cn.jpush.android.intent.PushService"/>
   <action android:name="cn.jpush.android.intent.PUSH_TIME"/>
/intent-filter>
vice>
ider android:name="cn.jpush.android.service.DataProvider" android:exported="false" android:process=":pushcore" android:authorities="com.aerolite.smartlock.Da
```

Fig. 5 Android App reversing - Jadx-Gui to obtain Java code

Table 1 Android apps for smart-locks and information after decompiling the apps

Android App	Version	Downloads	Rate	Unique app ID	Obfusca	API Route / URLs
Remote Lock	4.63.2	10k	<3	com.lockstate. remotelock	No	https://firebase.google.com/support/privacy/
Kwikset Kevo	2.9.1	500k+	3,3	com.unikey.kevo	Yes	https://mykevo.com/forgot_password https://mykevo.com/mobile_login/ https://tumblerprod.unikey.com/
August Home	23.1.0	1 M+	3,5	com.august.luna	Yes	http://logback.qos.ch https://github.com/grpc/grpc-java https://console.firebase.google.com
Lockitron (TTLock)	6.4.5	500 M+	3,2	com.tongtongsuo. app	No	https://servlet.ttlock.com https://qiniuroommaster.ttlock.com
Schlage Home	3.7.0	100k+	3,6	com.allegion.leopard	No	https://cdn.branch.io/ http://acs.amazonaws.com/groups/global/Auth
Wink-Smart Home	7.0.44	500 M+	3,0	com.quirky.android. wink	No	https://subscription-staging.wink.com https://subscription.wink.com
Yale Doorman	11.6.2	100 M+	4.3	com.august.bennu	Yes	https://global-config.autust.com https://api-production.august.com https://logger.august.com tcp://stage-mqtt.august.com:1883
Nuki	2023.2.1	500 M+	4.6	io.nuki	Partially	https://nuki.io com.google.firebase.encoders.json
Sherlock lock	3.6.6	Not working	–	com.aerolite.smartlock	Yes	https://api1.venus.aerolite.net

downloading and performing a reversing of the apps, some relevant information has been found.

3.2 Bluetooth replication attack for two smart-locks

The process of locking and unlocking the lock is very simple. The user must enter the app with the username used to register previously. Then the app will take him to the main menu of the application, where if there is already a lock associated, it will appear with the options to unlock, sliding the finger to the right, or lock, by sliding the finger to the left. Another way to unlock the smart-lock is by using a key fob or a keyboard with its corresponding PIN.

In order to study the Bluetooth security of the smart-lock, we will check how the sending of information between the application and the lock works. This attack has been performed for two different locks with different levels of cyber-security.

The objective of this process is to be able to capture the traffic when the lock connects with the smartphone and sends it the key to unlocking the door, and then try to replicate that traffic. For this purpose, we use a tool to make the connection between our computer, the application, and the lock. In this project, we use the Adafruit bluefruit LE sniffer, which allows us to capture and analyze the packets in transit between the devices. To analyze the traffic captured in the Wireshark application, we first need to link the information received by the sniffer. The nRF Sniffer for Bluetooth LE program is used for it. This tool allows us to see in real-time all the devices that are being captured by the sniffer in order to capture the data to later filter that content in Wireshark and check the information in detail. Thanks to this tool we can see all the available nearby Bluetooth devices with their respective MAC addresses. Furthermore, there is a guide to all the useful commands we can use with this tool.

Adafruit bluefruit LE Sniffer Adafruit bluefruit LE Sniffer [22] is programmed with a firmware image that makes it an easy-to-use firmware image that makes it an easy-to-use Bluetooth Low Energy (BLE) sniffer.

Bluetooth antenna CBT40NANO The CBT40NANO Bluetooth Nanoadapter [14] is used to establish a wireless connection with other Bluetooth devices. In this project, it is used to create a connection through a virtual machine, running the Linux operating system, to the lock. In this way, the traffic required to unlock the lock can be replicated.

Wireshark Wireshark is the most widely known and used packet analyzer in the world and is the program used in this project to analyze the packets sent between the mobile device and the lock. Thanks to this program, you can capture and analyze in detail all network traffic entering and leaving your PC. This free program allows you to perform an in-depth inspection of hundreds of protocols, as it supports the physical layer, link layer, network layer, transport layer, and application layer protocols.

Gatttool Gatttool [23] is a tool that allows obtaining information or manipulating attributes of a BLE device. Thanks to this tool, in this project we have implemented the writing of keys in the lock.

3.2.1 Sherlock S2 lock bluetooth attack

In order to carry out a Bluetooth attack, firstly, The Wireshark program detects the Sherlock lock once the device is located. The next step is to analyze its traffic in Wireshark. To do this, we select the lock and press the ’w’ key, this will redirect us directly to the Wireshark tool, filtering only the traffic of the selected Bluetooth device. Another method to capture the traffic between the two devices, implemented for the development of the project is through the HCI snoop log function [24]. This is a log file containing all Bluetooth transmissions that have been made from a smartphone. To be able to use this functionality, the developer mode must be activated on the mobile device and, once activated:

- Activate Bluetooth HCI logging, to enable the logs.
- And, USB debugging, to later be able to extract the logs via USB and view them on a computer.

Bluetooth is then turned ’on’ and ’off’ to enable data collection. Once Bluetooth is enabled, the lock can be unlocked to capture traffic.

In order to analyze the collected data on a computer, the log is passed through an *adb (Android Debug Bridge)* command line tool [25], which enables communication with a device, and finally, a *.log* file is generated that can be analyzed in Wireshark. This method was discarded because the use of a Bluetooth sniffer streamlines the data collection process. Once in Wireshark, we can start the analysis. To filter the traffic more effectively, the *’btatt’ filter* [26] is used. This filter shows all protocols related to Bluetooth traffic and, therefore, we can obtain only the packets that are sent between the lock and the application. When the filter has been introduced, we start capturing the traffic by blocking or unblocking the lock. In this way, information will start to reach the program, which we will later use to replicate the traffic and check if it is possible to hack it. A brief explanation of the details of the list of packets captured in the process is provided below.

The protocol used for the connection is ATT (Attribute Protocol) [27], an attribute-based protocol belonging to the BLE protocols, with client–server architecture, which allows the exchange of information. This protocol defines how data is represented and the methods by which this data can be read or written. In this case, it acts as a server, holding the data until the phone requests it. This data is stored in the BLE server as attributes. Image 6 shows the packets sent from the smartphone to Sherlock. The last column notifies us about the information being acquired at each packet in the connection. The point we are interested in for our investigation is when the key is sent from Master to Slave.

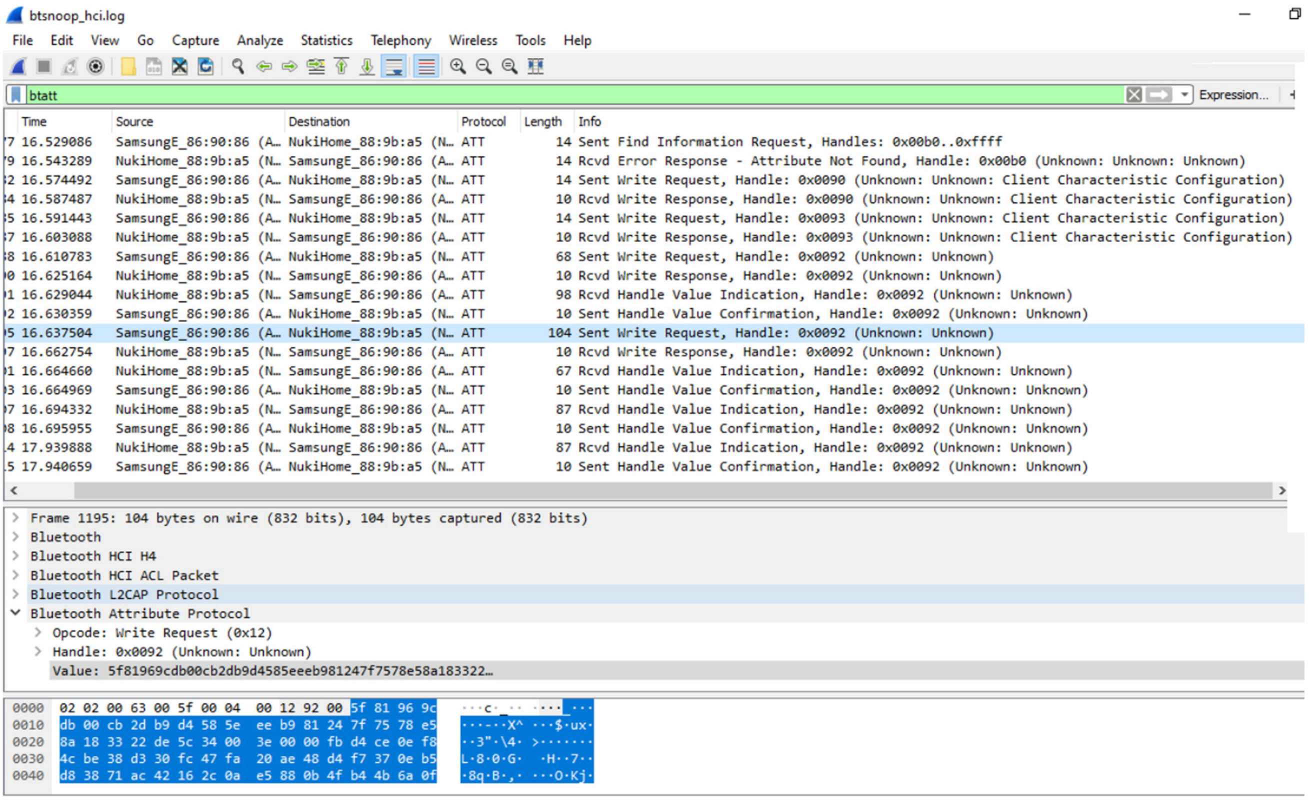
This is a packet that is making a written request with a *Handle 0x0019*. The Handles [28] are the components that process each of the packets that belong to the captures. The Handle extracts from the packet the necessary information, as in this case the value being sent. We can observe that three key writing requests are sent from the lock to the smartphone. Subsequently, the mobile application responds to the lock by sending a series of notifications informing that the value has been received. Finally, Sherlock again sends four key write requests to the device. This implies that unlocking the lock requires entering seven keys in total.

The next step is to look at the values that are written in each write packet. We can see this information by selecting the packet we want to analyze and a number of characteristics of that packet will appear. To see the value, we are interested in the information provided by the ATT protocol. The ATT protocol provides us with the three most important points to solve the problem:

- **Opcode** (Operation code) [29]. In this case, it is a write operation *’Write Command’*
- **Handle** (Pointer). Indicates the attribute to which the script is pointing.
- **Value**. Tells us the value of the key that is going to be written.

When we get all the keys that are sent, we can move on to replicating the packets. To do this, we need a Linux operating system or a virtual machine containing it. In our case, we opted for the latter option. Because of this, a Bluetooth adapter is needed as the virtual machine does not have a Bluetooth connection. First, we check that a signal is being received from the lock. To do this, we use the hcitool tool to scan for nearby Bluetooth devices. By executing the following command *’sudo hcitool lescan’* we can see the available BLE devices with their respective MAC address and check that we have a connection to our lock.

The next step is to use the hciconfig command,
which is used to configure Bluetooth devices. **hciX** is the name of a Bluetooth device installed on the system. In our case, the one we have installed is *hci0*. To be able to work with Bluetooth devices it is required to initialize it.



```
btatt
```

```
  Time         Source                    Destination               Protocol  Length  Info
7 16.529086    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           14  Sent Find Information Request, Handles: 0x00b0..0xffff
9 16.543289    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           14  Rcvd Error Response - Attribute Not Found, Handle: 0x00b0 (Unknown: Unknown: Unknown)
2 16.574492    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           14  Sent Write Request, Handle: 0x0090 (Unknown: Unknown: Client Characteristic Configuration)
4 16.587487    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           10  Rcvd Write Response, Handle: 0x0090 (Unknown: Unknown: Client Characteristic Configuration)
5 16.591443    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           14  Sent Write Request, Handle: 0x0093 (Unknown: Unknown: Client Characteristic Configuration)
7 16.603088    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           10  Rcvd Write Response, Handle: 0x0093 (Unknown: Unknown: Client Characteristic Configuration)
8 16.610783    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           68  Sent Write Request, Handle: 0x0092 (Unknown: Unknown)
0 16.625164    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           10  Rcvd Write Response, Handle: 0x0092 (Unknown: Unknown)
1 16.629044    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           98  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
2 16.630359    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
5 16.637504    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          104  Sent Write Request, Handle: 0x0092 (Unknown: Unknown)
7 16.662754    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           10  Rcvd Write Response, Handle: 0x0092 (Unknown: Unknown)
1 16.664660    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           67  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
3 16.664969    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
7 16.694332    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           87  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
8 16.695955    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
4 17.939888    NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT           87  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
5 17.940659    SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT           10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
```

```
> Frame 1195: 104 bytes on wire (832 bits), 104 bytes captured (832 bits)
> Bluetooth
> Bluetooth HCI H4
> Bluetooth HCI ACL Packet
> Bluetooth L2CAP Protocol
v Bluetooth Attribute Protocol
    > Opcode: Write Request (0x12)
    > Handle: 0x0092 (Unknown: Unknown)
      Value: 5f81969cdb00cb2db9d4585eeeb981247f7578e58a183322…
```

```
0000  02 02 00 63 00 5f 00 04  00 12 92 00 5f 81 96 9c   ···c·_·· ········
0010  db 00 cb 2d b9 d4 58 5e  ee b9 81 24 7f 75 78 e5   ···-··X^ ···$·ux·
0020  8a 18 33 22 de 5c 34 00  3e 00 00 fb d4 ce 0e f8   ··3"·\4· >·······
0030  4c be 38 d3 30 fc 47 fa  20 ae 48 d4 f7 37 0e b5   L·8·0·G·  ·H··7··
0040  d8 38 71 ac 42 16 2c 0a  e5 88 0b 4f b4 4b 6a 0f   ·8q·B·,· ···O·Kj·
```

Fig. 6 Capturing packet traffic from Sherlock using Wireshark, with requests and responses

The last tool used and the most important for our research is *Gatttool*. This tool allows us to connect to BLE devices with the MAC address of the device and manipulate its attributes with a series of commands available from the tool. In such a way, it will be possible to replicate the packets we have previously captured. In case the MAC address of the lock to be analyzed is not known, there are two options for searching for it. Firstly, we can obtain this information from packet capture in Wireshark, which shows the addresses of the two communicating devices. Secondly, thanks to the hcitool tool, mentioned above, we can easily see the BLE devices found and their associated MAC address. The Gatttool tool has several interesting commands for manipulating Bluetooth devices or simply acquiring relevant information about them. Bluetooth devices or simply to acquire relevant information from them.

Once the commands necessary to replicate packets are clear, we connect to the lock and send the keys obtained. At first, in order to connect to the device, each key is entered individually by using Gattool.

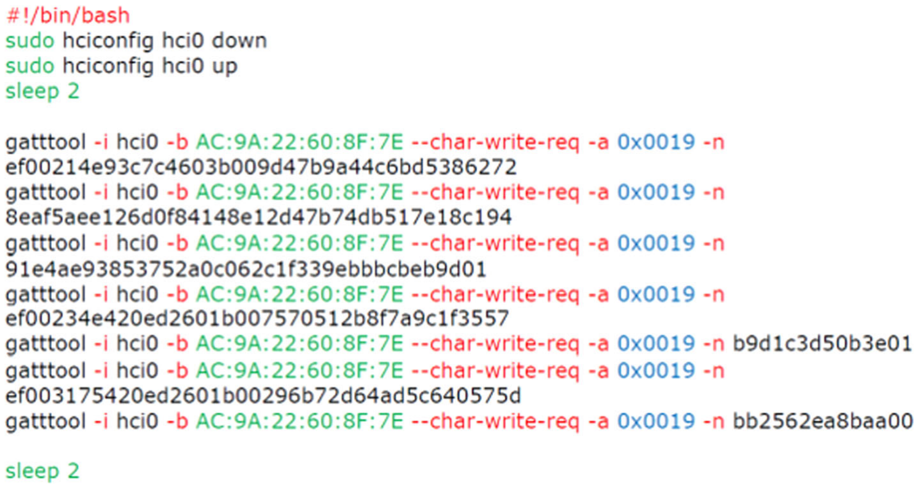
This was not feasible because the connection to the lock was lost a few seconds after initialization, and since 7 keys had to be entered, it was unfeasible to establish the connection for each key that had to be entered. Therefore, a simple bash script has been implemented, see Fig. 7) which contains all the necessary commands for connecting and writing, making the process of sending packets easier and faster.

In this way, all the commands explained above are executed together saving a lot of time when hacking the lock. And finally, the lock is unlocked by replicating the packets. It's a fairly simple process and one that raises questions about the security of this brand of smart-lock. However, even though it is easy and quick to hack, Sherlock does have some security measures, such as changing the keys after several minutes have passed since the door has been opened.

However, there is still ample time to perform packet replication. In addition, in the future, a program could be implemented or the script could be optimized to make the data collection more effective and faster, thus unlocking the lock in a matter of seconds.

3.2.2 Nuki lock bluetooth attack

Nuki lock has been studied in this project. Next, some technical concepts about it and its operation will be shown, and then proceed to the analysis of its security. It is especially noteworthy that the Nuki lock operates at the highest level of security encryption, as it uses AES with 256-bit keys. AES is a symmetric block cipher, which



```
#!/bin/bash
sudo hciconfig hci0 down
sudo hciconfig hci0 up
sleep 2

gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n
ef00214e93c7c4603b009d47b9a44c6bd5386272
gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n
8eaf5aee126d0f84148e12d47b74db517e18c194
gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n
91e4ae93853752a0c062c1f339ebbbcbeb9d01
gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n
ef00234e420ed2601b007570512b8f7a9c1f3557
gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n b9d1c3d50b3e01
gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n
ef003175420ed2601b00296b72d64ad5c640575d
gatttool -i hci0 -b AC:9A:22:60:8F:7E --char-write-req -a 0x0019 -n bb2562ea8baa00

sleep 2
```

Fig. 7 Bluetooth script for sending Sherlock lock keys

means that it encrypts and decrypts data in blocks of 128 bits each. To do this, it uses a specific cryptographic key, which is effectively a set of protocols for manipulating information. This key can be 128, 192, or 256 bits long. AES-256, the 256-bit key version of AES, is the encryption standard used by LE VPN. It is the most advanced form of encryption and consists of 14 rounds of substitution, transposition, and mixing for an exceptionally high level of security. Its larger key size makes it essentially unbreakable, meaning that even if hacked, the data would be impossible to decrypt. The Nuki lock stands out in Europe for its good reviews regarding its security, therefore, it has been analyzed if a package replication would hack it.

With the Adafruit bluefruit LE sniffer and its nRF Sniffer for Bluetooth program to detect nearby BLE devices, we connect to the Nuki lock to later analyze the packets sent and received in Wireshark. Once Wireshark has been started and the ’btatt’ filter has been introduced to acquire only Bluetooth traffic, we can unlock the lock and check what information is reaching us and if it will be possible to hack it. In Fig. 8 we can appreciate the communication between the Nuki lock and our Samsung mobile device.

The protocol used for the connection is also ATT where the attributes sent are stored, but in the column indicating the packet information, slight formatting changes can be observed, which will be discussed later. In this case, we can see that several write requests are made to the lock to send the keys with Handle 0x0092 with the command:

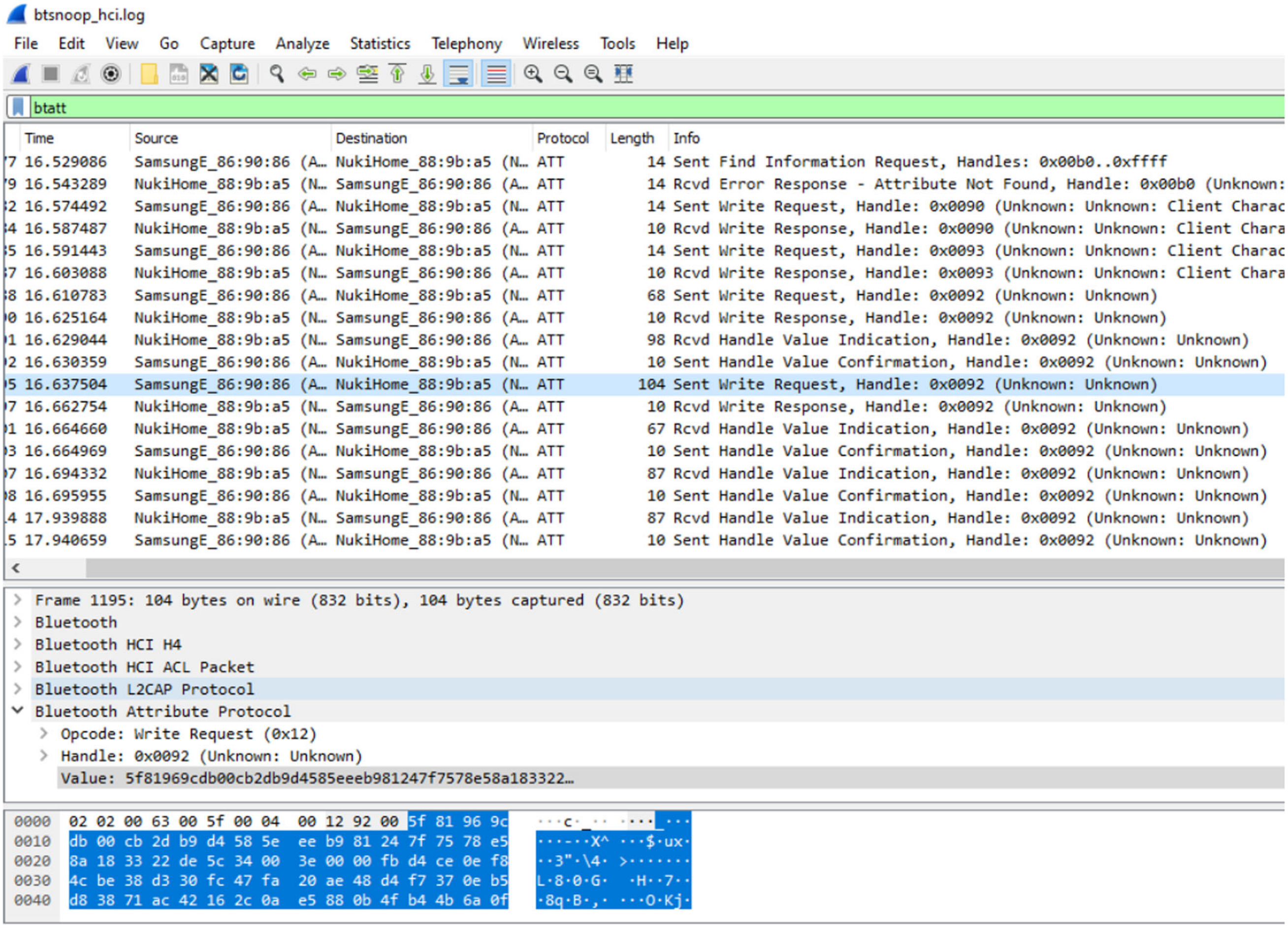
Sent Write Request, Handle 0x0092

Therefore, we must collect all the values that are sent in order to later perform their replication. In order to see these values, we are interested in the information provided by the ATT protocol. Once we have all the keys that are sent, we can replicate the traffic again from our virtual machine. First, we check that we are receiving a connection from the Nuki lock with the hcitool tool and, since we get a signal, we can run our bash script, previously modified with the values found in Wireshark. However, even though the values are correctly written to the lock, it fails to unlock it. This is because, when unlocking the door, Nuki instantly changes the key for the next opening, being unfeasible to hack the lock with this method, since the previously collected keys would be automatically obsolete.

In the following, we will explain in detail how the encryption works in the Nuki lock [30]. This lock uses the principle of end-to-end encryption, i.e., it applies a cipher to the key in such a way that only the receiving device can decrypt it. To establish communication between the Nuki app and the smart-lock, a proprietary key is used that is known only to both devices. To protect against attackers, the data is encrypted before it is transmitted by the sender (the Nuki app). This is done using the NaCl (Networking and Cryptography library) process [31]. In this process, unique combinations of numbers and letters are used only once. This data is transferred via Bluetooth and decoded again when received by the receiver (Nuki smart-lock).

1. The Nuki app sends the “unlock” instruction and encrypts it in such a way that only the app and Nuki smart-lock know the key.
2. The Nuki app transfers the encrypted message via Bluetooth to the lock.
3. Nuki smart-lock knows the key and can therefore decrypt the contained message and execute the “unlock” command.
4. In the unlocking process, the Nuki app receives a random number. The “unlock” instruction can only be sent to the lock when it necessarily contains an identical random number.

If another unlock instruction with the same random number is subsequently sent to the door lock, Nuki smart-lock rejects the instruction. This analysis shows us that the security level of Nuki is high, as reported by the


btsnoop_hci.log
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
btatt

   Time        Source                   Destination              Protocol  Length  Info
7  16.529086   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          14  Sent Find Information Request, Handles: 0x00b0..0xffff
9  16.543289   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          14  Rcvd Error Response - Attribute Not Found, Handle: 0x00b0 (Unknown:
2  16.574492   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          14  Sent Write Request, Handle: 0x0090 (Unknown: Unknown: Client Charac
4  16.587487   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          10  Rcvd Write Response, Handle: 0x0090 (Unknown: Unknown: Client Chara
5  16.591443   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          14  Sent Write Request, Handle: 0x0093 (Unknown: Unknown: Client Charac
7  16.603088   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          10  Rcvd Write Response, Handle: 0x0093 (Unknown: Unknown: Client Chara
8  16.610783   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          68  Sent Write Request, Handle: 0x0092 (Unknown: Unknown)
0  16.625164   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          10  Rcvd Write Response, Handle: 0x0092 (Unknown: Unknown)
1  16.629044   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          98  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
2  16.630359   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
5  16.637504   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT         104  Sent Write Request, Handle: 0x0092 (Unknown: Unknown)
7  16.662754   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          10  Rcvd Write Response, Handle: 0x0092 (Unknown: Unknown)
1  16.664660   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          67  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
3  16.664969   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
7  16.694332   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          87  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
8  16.695955   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)
4  17.939888   NukiHome_88:9b:a5 (N…   SamsungE_86:90:86 (A…   ATT          87  Rcvd Handle Value Indication, Handle: 0x0092 (Unknown: Unknown)
5  17.940659   SamsungE_86:90:86 (A…   NukiHome_88:9b:a5 (N…   ATT          10  Sent Handle Value Confirmation, Handle: 0x0092 (Unknown: Unknown)

> Frame 1195: 104 bytes on wire (832 bits), 104 bytes captured (832 bits)
> Bluetooth
> Bluetooth HCI H4
> Bluetooth HCI ACL Packet
> Bluetooth L2CAP Protocol
v Bluetooth Attribute Protocol
    > Opcode: Write Request (0x12)
    > Handle: 0x0092 (Unknown: Unknown)
      Value: 5f81969cdb00cb2db9d4585eeeb981247f7578e58a183322…

0000  02 02 00 63 00 5f 00 04  00 12 92 00 5f 81 96 9c   ···c·_·· ····_···
0010  db 00 cb 2d b9 d4 58 5e  ee b9 81 24 7f 75 78 e5   ···-··X^ ···$·ux·
0020  8a 18 33 22 de 5c 34 00  3e 00 00 fb d4 ce 0e f8   ··3"·\4· >·······
0030  4c be 38 d3 30 fc 47 fa  20 ae 48 d4 f7 37 0e b5   L·8·0·G·  ·H··7··
0040  d8 38 71 ac 42 16 2c 0a  e5 88 0b 4f b4 4b 6a 0f   ·8q·B·,· ···O·Kj·
```

Fig. 8 Capturing packets traffic of Nuki Smart-Lock using Wireshark

manufacturers, and that overall it is a lock that the market can currently rely on. However, since the packet replication method used in this project is only one of the options for hacking, this research does not imply that there may not be another procedure for unlocking the lock and that this lock may end up being vulnerable to attack.

3.2.3 Bluetooth cyber-security comparison (Nuki-Sherlock S2)

Having analyzed the operation of both locks and seen how the packets are sent between the two devices, we can make an analysis of the differences that have been found during the hacking procedure for both locks. First of all, it has been possible to verify that the Nuki lock has better security than the Sherlock lock, since during the investigation Sherlock was successfully unlocked, but not Nuki. In addition, there is more open information about the encryption used in the Nuki lock than in the case of the Sherlock, and in fact, it is known that the Nuki lock uses AES with 256-bit keys.

In conclusion, there are a number of points that show the main differences between the two locks and why one is more vulnerable than the other.

- *Packet writing format* Although in both connections write requests are made and keys are stored in the ATT protocol, the request is not sent in the same format. In the Sherlock lock *"Rvcd write command"* is used which means that it has received the write command with the key value and in the Nuki lock *"Sent Write Request"* is used which implies that the requested write has been sent with that value. Moreover, in the first case, it is the lock that sends the value and in the second case, it is the mobile application that does the sending. With this information it can be assumed that we manage to unlock the Sherlock lock because we are attacking the lock, however, we were unable to hack the Nuki lock because we attacked the mobile device, which would not be present at the time of the cyberattack.
- *Key change time* AAs mentioned above, Sherlock keeps the same key for a few minutes after unlocking the door, making it easy to attack his security. However,

> Nuki controls time much more securely. This lock changes the key the instant the door is unlocked. In this way, there is no time margin for burglary. This is the most important and strongest point of Nuki in terms of its security and the one that makes it stand out from Sherlock.

4 Conclusions and future work

In conclusion, the security of smart-locks in general is quite good. Most of them use AES encryption for the keys used in Bluetooth communication and have obfuscated the Android app code. However, security in the use of Bluetooth technology is of utmost importance due to the potential risk of unauthorized access. As demonstrated in this research, smart-locks using Bluetooth can be vulnerable to man-in-the-middle attacks if there is no encryption is used, as is the case with the Sherlock S2 smart-lock. This attack can result in unauthorized access to the lock and compromise the security of the property. It is therefore important to be aware of the security gap that can exist in some smart-locks against such attacks. Smart-locks with AES encryption and one-time keys, such as the Nuki lock, are secure, and as you can see in the analysis of the smart-locks application code, most of them use AES and are generally secure.

Future Work There are several potential avenues for future work to investigate further the vulnerabilities of some types of smart-locks. First, research should focus on thorough code reviews to find bugs that allow unauthorized access or other types of errors. Second, efforts should be made to develop modified Android apps to bypass the authentication required to open the smart-locks. Thirdly, check the connection between the Android app and the vendor API in the cloud using Burp Suite to see if there is any relevant information or any kind of SQL injection, GIT code online, etc. Finally, old versions of Android apps can be scanned for information and security vulnerabilities.

Overall, it is important to continue to research and look for vulnerabilities to ensure that smart-locks have robust security and cannot be compromised, and that users can continue to rely on them to secure their properties.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. Research supported by the Cátedra Institucional de Ciberseguridad Binter and the Cátedra Edosoft de Computación en la Nube e Inteligencia Artificial, both from the University of La Laguna.

References

1. Palle, S. (2017). Smart locks: Exploring security breaches and access extensions. PhD thesis, Oklahoma State University.
2. Lamping, U., & Warnicke, E. (2004). Wireshark user's guide. *Interface, 4*(6), 1.
3. Solutions, N.H. (2013). Nuki smart lock. https://nuki.io/es/smart-lock/.
4. Xiaomi. Sherlock smart lock. https://cerradurasinteligentes.net/xiaomi/xiaomi-sherlock-s2/.
5. Bichsel, B., Raychev, V., Tsankov, P., & Vechev, M. (2016). Statistical deobfuscation of android applications. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 343–355.
6. Ye, M., Jiang, N., Yang, H., & Yan, Q. (2017). Security analysis of internet-of-things: A case study of august smart lock. In *2017 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pp. 499–504. IEEE.
7. Ho, G., Leung, D., Mishra, P., Hosseini, A., Song, D., & Wagner, D. (2016). Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*, pp. 461–472.
8. Ahmad, T., Morelli, U., Ranise, S., & Zannone, N. (2018). A lazy approach to access control as a service (ACaaS) for IoT: an AWS case study. In *Proceedings of the 23nd ACM on symposium on access control models and technologies*, pp. 235–246.
9. Hassani, R. (2020) Security evaluation of a smart lock system.
10. Borg, A., & Francke, C.A. (2020). IoT Pentesting: Obtaining the firmware of a smart lock.
11. Han, Z., Liu, L., & Liu, Z. (2019). An efficient access control scheme for smart lock based on asynchronous communication. In *Proceedings of the ACM Turing celebration conference-China*, pp. 1–5.
12. Pinjala, S.R., & Gupta, S. (2019). Remotely accessible smart lock security system with essential features. In: *2019 international conference on wireless communications signal processing and networking (WiSPNET)*, pp. 44–47. IEEE
13. Aluri, D. C. (2020). Smart lock systems: An overview. *International Journal of Computer Applications, 177*(37), 40–43.
14. Lin, P.-J., Ho, C.-T. (2020). Smart lock security system based on artificial internet of things. In *2020 IEEE Eurasia conference on IOT, communication and engineering (ECICE)*, pp. 79–81. IEEE
15. Lounis, K., & Zulkernine, M. (2019). Bluetooth low energy makes "just works" not work. In *2019 3rd cyber security in networking conference (CSNet)*, pp. 99–106. IEEE
16. Wu, J., Nan, Y., Kumar, V., Tian, D.J., Bianchi, A., Payer, M., & Xu, D. (2020). Blesa: Spoofing attacks against reconnections in Bluetooth low energy. In *WOOT@ USENIX Security Symposium.*
17. Persman, P., & Öjebrant, S. (2021). Security analysis of a smartlock.

18. Mohammed, A.Z., Singh, A., Dayanıklı, G.Y., Gerdes, R., Mina, M., & Li, M. (2022). Towards wireless spiking of smart locks. In *2022 IEEE security and privacy workshops (SPW)*, pp. 251–257. IEEE.
19. Price, D. The 7 Best Sites for Safe Android APK Downloads. https://www.makeuseof.com/tag/safe-android-apk-downloads/.
20. Connor Tumbleson, R.W. ApkTool: A tool for reverse engineering Android apk files. https://ibotpeaches.github.io/Apktool/.
21. Github. jadx - Dex to Java decompiler. https://github.com/skylot/jadx.
22. Adafruit. Bluefruit LE Sniffer - Bluetooth Low Energy (BLE 4.0) - nRF51822. https://www.adafruit.com/product/2269.
23. Gatttool: Gatttool. http://manpages.ubuntu.com/manpages/cosmic/man1/gatttool.1.html.
24. snoop log, H. HCI snoop log. https://www.mybluetoothreviews.com/what-is-bluetooth-hci-snoop-log/.
25. Google. Android Debug Bridge (adb). https://developer.android.com/studio/command-line/adb?hl=es-419.
26. Wireshark. btatt Filter. https://www.wireshark.org/docs/dfref/b/btatt.html.
27. Programmerclick.com. ATT (Attribute Protocol). https://programmerclick.com/article/68241335665/.
28. Wikipedia. Handle. https://es.wikipedia.org/wiki/Handle.
29. Wikipedia. Opcode. https://es.wikipedia.org/wiki/Cdigo_de_operacin.
30. Nuki. Nuki Encryption. https://nuki.io/es/blog/sientete-seguro/seguridad-en-primer-plano-concepto-de-cifrado-de-nuki-explicado-de-forma-sencilla/.
31. Wikipedia. NaCl. https://en.wikipedia.org/wiki/NaCl_(software).

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Cándido Caballero-Gil is a Senior Lecturer in the area of Computer Architecture and Technology at the University of La Laguna, Tenerife, Spain. He received his degree in Computer Science Engineering from the University of Las Palmas de Gran Canaria in 2007 and his Ph.D. from the University of La Laguna in 2011. His research interests include ad hoc networks and VANET security, cryptography, especially in the area of key management and privacy. He is a member of the CryptULL research group, dedicated to the development of cryptology projects (since 2007), and is involved in several projects and publications in this area. He has authored several conference and journal papers.

Rafael Álvarez is a Senior Lecturer in the Department of Computer Science and Artificial Intelligence at the University of Alicante, Spain. He received a combined Bachelor's and Master's degree in Computer Science in 2001, a Ph.D. in Computer Science in 2005, and the Extraordinary Doctorate Award in 2009. He is a member of the Computational Security and Cryptology Research Group and is interested in security, cryptography, machine learning, and their applications in computer science. He has participated in numerous international conferences and has been published in prestigious journals.

Candelaria Hernández-Goya was born in Santa Cruz de Tenerife, Spain, on June 18, 1970. She received the M.S. and the Ph.D. degrees in Mathematics from the University of La Laguna, Spain in 1995 and 2003, respectively. She has been Lecturer at the University of La Laguna since 1998 and Senior Lecturer since 2010. Her major interests are security in vehicular ad hoc networks, authentication, and cryptographic protocols.

Jezabel Molina-Gil is a Ph.D. Assistant Professor of Computer Science and Artificial Intelligence at the University of La Laguna, Tenerife, Spain. She received her degree in Computer Science Engineering from the University of Las Palmas de Gran Canaria in 2007 and her Ph.D. from the University of La Laguna in 2011. Her research focuses on cryptography and VANET security, especially in the areas of cooperation and data aggregation. She is a member of the CryptULL research group, dedicated to the development of projects in cryptology (since 2007), and is involved in several projects and publications related to this area. She has authored several conference and journal papers.