

국제학석사 학위논문

Smart Home IoT Forensics

-Study on Data Security and Challenges to Data Acquisition-

스마트 홈 IoT 포렌식

-데이터 보안 및 데이터 획득 대한 해결 방안 제시-

Birhanu, Addisu Afework (발하누 아디수 아페워크)

International Studies (국제학과)

Legal Informatics and Forensic Science (정보법과학전공)

한림대학교 대학원

Graduate School, Hallym University

ክሊሜንቲን ሰላሳ ገብረ

Smart Home IoT Forensic
-Study on Data Security and Challenges to Data Acquisition-

2019

Birhanu Addisu Afework

국제학석사 학위논문

Smart Home IoT Forensics

-Study on Data Security and Challenges to Data Acquisition-

스마트 홈 IoT 포렌식

-데이터 보안 및 데이터 획득 대한 해결 방안 제시-

Birhanu, Addisu Afework (발하누 아디수 아페워크)

International Studies (국제학과)

Legal Informatics and Forensic Science (정보법과학전공)

한림대학교 대학원

Graduate School, Hallym University

장윤식, Joshua I. James 교수지도

국제학석사 학위논문

발하누 아디수 아페워크의 석사학위논문을

합격으로 판정함

2019 년 6 월 28 일

심사위원장 박노섭

심사위원 안정민

심사위원 장윤식

심사위원 Joshua I. James

Table of Contents

| | |
|---|----|
| List of Figures | IV |
| List of Tables | VI |
| List of Listings | VI |
| CHAPTER 1. INTRODUCTION | 1 |
| 1.1. Background..... | 1 |
| 1.2. IoT Ecosystem and Security..... | 3 |
| 1.3. IoT Security Framework | 4 |
| 1.3.1. Companion Apps..... | 5 |
| 1.3.2. IoT Backend Cloud | 8 |
| 1.3.3. IoT Communication and Security..... | 12 |
| 1.3.4. IoT Hubs and Sensors Security | 12 |
| 1.4. Thesis Statement | 13 |
| 1.5. Research Questions | 13 |
| 1.6. Contribution | 14 |
| 1.7. Thesis Scope..... | 14 |
| 1.8. Thesis Structure | 15 |
| CHAPTER 2. BACKGROUND RESEARCH..... | 16 |
| 2.1. Smart home IoT Devices Security and Privacy Issues | 16 |
| 2.1.1. General IoT Security Issues..... | 16 |
| 2.1.2. IoT Companion Apps Data Security | 18 |
| 2.1.3. IoT Backend Cloud APIs Security..... | 19 |
| 2.2. Forensic Investigation of IoT Ecosystem | 21 |
| 2.3. IoT Forensic Challenges | 22 |
| 2.4. Summary..... | 23 |
| CHAPTER 3. RESEARCH METHODOLOGY | 24 |
| 3.1. Followed Research Process..... | 24 |
| 3.2. Followed Research Procedures | 25 |
| 3.2.1. IoT Android Companion (Client) Apps Investigation | 25 |
| 3.2.2. Companion Apps Security Analysis using Reverse Engineering | 26 |

| | |
|--|----|
| 3.2.3. Companion Apps Live Forensics Analysis..... | 30 |
| 3.2.4. Network Investigation..... | 31 |
| 3.2.5. Cloud API Security Analysis and Demo..... | 32 |
| 3.3. Demonstration Tool Development | 33 |
| 3.4. Required Research Materials and Tools..... | 33 |
| 3.5. List of IoT Case Studied Devices | 33 |
| 3.6. IoT Devices Set up Network Design | 34 |
| 3.7. Assumptions..... | 34 |
| 3.8. Summary | 35 |
| CHAPTER 4. CASE STUDIES..... | 36 |
| 4.1. SKT Nugu AI speaker..... | 36 |
| 4.1.1. SKT Nugu (Aladdin) App Analysis | 37 |
| 4.1.2. Nugu AI Speaker Device Network Investigation..... | 45 |
| 4.1.3. Results Analysis..... | 46 |
| 4.2. Clova AI speaker | 47 |
| 4.2.1. Naver Clova App Analysis..... | 48 |
| 4.2.2. Clova AI Speaker Device Network Investigation | 57 |
| 4.2.3. Results Analysis | 58 |
| 4.3. Xiaomi Smart Home Kit..... | 58 |
| 4.3.1. Mi Home App Analysis | 60 |
| 4.3.2. Xiaomi Lumi-gateway Network Investigation..... | 68 |
| 4.3.3. Results Analysis | 69 |
| 4.4. Sen.se Mother | 70 |
| 4.4.1. Pocket Mother App Analysis..... | 72 |
| 4.4.2. Sen.Se Mother Network Investigation..... | 75 |
| 4.4.3. Results Analysis | 76 |
| 4.5. Summary | 76 |
| CHAPTER 5. RESULTS DISCUSSION..... | 77 |
| 5.1. Companion Apps Data Storage Security | 78 |

| | |
|--|-----|
| 5.2. Communication between Companion Apps and the Cloud | 79 |
| 5.3. Communication between IoT Devices/Hub and the Cloud | 81 |
| 5.4. IoT Cloud APIs Security Investigation | 81 |
| 5.4.1. Naver Clova | 82 |
| 5.4.2. SKT Nugu | 82 |
| 5.4.3. Xiaomi Smart Home Kit | 83 |
| 5.4.4. Sen.se Mother | 83 |
| 5.5. Forensic Implications of the Securities..... | 84 |
| 5.6. Privacy Implications of the Securities | 86 |
| 5.7. Cloud Data Acquisition Tools | 88 |
| 5.8. Summary | 89 |
| CHAPTER 6. CONCLUSION | 90 |
| 6.1. Future Works | 93 |
| REFERENCES | 94 |
| ENGLISH ABSTRACT | 98 |
| 국문 초록 | 100 |

List of Figures

| | |
|--|----|
| Figure 1: Simplified IoT Architecture..... | 4 |
| Figure 2: Android data storage model (by Altuwaijri and Ghouzali) | 6 |
| Figure 3: Service Oriented Architecture APIs | 9 |
| Figure 4: IoT Research network design and configuration | 35 |
| Figure 5: Followed research process | 25 |
| Figure 6: Example of an unzipped apk file | 29 |
| Figure 7: SKT Nugu AI speaker operation mode..... | 36 |
| Figure 8: Nugu Unzipped apk file..... | 38 |
| Figure 9: Nugu App information after decompiled using MobSF | 39 |
| Figure 10: Nugu App data creation code snippets | 40 |
| Figure 11: Nugu app database dynamic analysis result from Inspeckage | 40 |
| Figure 12: SKT Nugu shared preference manager code snippet..... | 42 |
| Figure 13: SKT Nugu shared preference dynamic analysis Inspeckage output | 42 |
| Figure 14: com.skt.nugu.xml file | 43 |
| Figure 15:Permissions in the SKT Nugu Androidmanifest file | 43 |
| Figure 16: Communication security between SKT Nugu App and SKT Cloud | 44 |
| Figure 17: SKT Nugu app sslError handling method code snippet..... | 44 |
| Figure 18: Intercepted Nugu Android App information | 45 |
| Figure 19: Naver Clova AI speaker operation mode..... | 47 |
| Figure 20: Naver Clova App information after decompiled using MobSF..... | 49 |
| Figure 21: Naver Clova app database creation code snippet..... | 50 |
| Figure 22: Naver Clova app linenotice_pref.db file | 52 |
| Figure 23: Naver Clova app shared preference file editor code snippet | 52 |

| | |
|--|----|
| Figure 24: Naver Clova app secured shared preference code snippet..... | 53 |
| Figure 25: Naver Clova clova.xml file with partially encrypted keys and values..... | 54 |
| Figure 26: Naver Clova app clovatoken.xml file with current token values..... | 54 |
| Figure 27: Naver Clova APIs base URL in the extracted apk file | 55 |
| Figure 28: Naver Clova app permissions in the Android manifest file..... | 55 |
| Figure 29: Communication security between Naver Clova App and Naver Cloud | 55 |
| Figure 30: Naver Clova Bearer Token intercepted using MITM attack | 56 |
| Figure 31: Naver Clova AI speaker device and cloud encrypted network traffic | 57 |
| Figure 32: Xiaomi Smart Home Kit operation mode | 59 |
| Figure 33: Mi Home app dex to jar converted files | 61 |
| Figure 34: Mi Home app database creation code snippet..... | 62 |
| Figure 35: Mi Home app dynamic analysis Inspeckage output | 63 |
| Figure 36: Mi Home app databases handler class files from MobSF security | 63 |
| Figure 37: Mi Home app database files | 64 |
| Figure 38: Mi Home app miiio2.db file analysis result using SQLite DB browser | 65 |
| Figure 39: Mi Home app shared preference manager code snippet | 66 |
| Figure 40: Mi Home app external storage permissions in the Androidmanifest file | 66 |
| Figure 41: Communication security between Mi Home App and Xiaomi Cloud..... | 67 |
| Figure 42: ADB logcat output for Xiaomi Mi Home App | 68 |
| Figure 43: Network traffic between Xiaomi Lumi gateway and Cloud..... | 69 |
| Figure 44: Sen.se Mother set up architecture with Sen.se Cloud..... | 71 |
| Figure 45: Pocket Mother app shared preference file code snippet | 74 |
| Figure 46: Communication security between Pocket Mother app and Sen.se Mother Cloud..... | 74 |
| Figure 47: Pocket Mother app man-in-the-middle attack | 75 |
| Figure 48: Network traffic analysis between mother and Sen.se Cloud | 75 |

| | |
|---|----|
| Figure 49: SKT Nugu and Naver Clova AI Speakers Cloud Data Acquisition tools..... | 89 |
|---|----|

List of Tables

| | |
|---|----|
| Table 1: Gartner 2017 IoT prediction (Source Gartner 2017)..... | 2 |
| Table 2: Summary of API authorization and authentication options | 11 |
| Table 3: Selected smart home IoT devices..... | 34 |
| Table 4: Used research materials and tools | 33 |
| Table 5: Summary of the companion apps data storage security implementation | 79 |
| Table 6: Summary of communication security between IoT companion Apps and Cloud..... | 80 |
| Table 7: Summary of communication security between IoT devices and the Cloud | 81 |
| Table 8: Summary of the Cloud API security Methods for the selected IoT devices | 84 |

List of Listings

| | |
|--|----|
| Listing 1: adb commands to access smartphone | 28 |
| Listing 2: adb command to list installed applications | 28 |
| Listing 3: Example of listing package name | 28 |
| Listing 4: adb pull command to download installed apps | 28 |
| Listing 5: dex to jar converting command | 29 |
| Listing 6: dump command returns the current state of the tables in the database..... | 41 |
| Listing 7: Live Analysis of linenotice_pref.db file using the sqlite3 command | 51 |
| Listing 8: Live Analysis of miio.db file using the sqlite3 command | 65 |

CHAPTER 1. INTRODUCTION

1.1. Background

Technologies such as the Internet and wireless communication revolutionized the way humans interact with each other and execute businesses around the world [1]. Now, on top of that, the Internet of Things (IoT) is adding another layer of advancement to basic interaction between human and physical objects, an advancement that altered the interaction methods [2]. Objects that required manual interactions to operate are becoming more automated. IoT is becoming one of the mainstream technologies integrated into society. But, what does IoT mean and how is it applicable in today's society?

The term "Internet of Things" referred as IoT, was first coined by Kevin Ashton in 1999 while he was the Executive Director of Auto-ID Center; a research center focused on the application of RFID technologies [3]. According to Ashton's usage, the term refers to the ability of things to generate information that, in turn, enables us to monitor and control them efficiently [4]. However, there is no worldwide accepted single definition of IoT, in spite of Ashton's claim to coining the term. Different bodies define the term differently depending on their contexts. As a result, IoT could have different meanings from the academic and research, standardization organizations or governments point of view. However, for this thesis, we will use an academic definition.

In the academic community, the Internet of Things (IoT) refers to the interconnection of intelligent devices with actuators and sensors through communication networks to automate and minimize manual interactions of humans with physical objects [5]. Smart home automation, healthcare services, manufacturing, power grids, transportation, and smart cities are the main application areas of IoT devices in today society. According to Gartner's 2017 prediction, by 2020, there will be 20.4 billion IoT devices connected to the Internet [6]. That means, IoT devices will increase roughly by three folds from the number of connected devices in 2017, which was

around 8.4 billion. From that, the consumer side application area such as smart home IoT devices deployment will take about 65% of the share [6].

Table 1: Gartner 2017 IoT deployment prediction (Source Gartner, 2017). The table shows that by 2020, the number of connected IoT devices will increase by almost 3 times than the number of connected IoT devices in 2016.

| Category | 2016 | 2017 | 2018 | 2020 |
|-----------------------------|-------------|-------------|-------------|-------------|
| Consumer | 3,963.0 | 5,244.3 | 7,036.3 | 12,863.0 |
| Business: Cross-industry | 1,102.1 | 1,501.0 | 2,132.6 | 4,381.4 |
| Business: Vertical-specific | 1,316.6 | 1,635.4 | 2,027.7 | 3,171.0 |
| Grand Total | 6,381.8 | 8,380.6 | 11,196.6 | 20,415.4 |

Smart homes are homes with computing devices and appliances that offer context-aware services and adjust the home environment based on the context and user preferred settings [7]. In short, smart home refers to homes that use IoT devices to automate routine user activities, including providing security services.

These smart home IoT devices collect users' information such as birthday, location, daily activities and health records to perform their intended functionality. Moreover, data generated by one IoT device can be sent to another IoT device to trigger intended actions. For instance, sensors attached to the main door of a house collect information when the door opens and closes. Then, this data can be sent to other devices such as the light and room temperature controllers. Based on the information generated from the single sensor attached to the door, the whole house environment can be adjusted to suit the required context.

On the other hand, the same data can be used to infer the routine activities of the user, which can be used in favor of or against the user depending on the situations. Therefore, the smart home IoT devices should be secure enough to protect themselves, and the privacy of the users from cyber-attacks. However, as recent cyber-attacks research on IoT devices and their ecosystem indicate, these devices are not as secure enough as they could be. Moreover, apart from being a direct target of cyber-attacks, smart home IoT devices may be involved in crimes either as

enabling tools or witnesses to crimes committed in smart homes. As a result, there is a need to forensically investigate smart home IoT devices found in crime scenes.

However, in smart home IoT devices forensic investigations, investigators face challenges due to the nature of IoT devices, their applications and implemented securities. Some of the challenges are identifying and collecting the devices, acquiring data from different data sources and pre-filtering the forensic relevancy of the data [8], [9]. Moreover, since the smart home IoT devices depend on the cloud to process and store collected data as stated above, the forensic process moves to a cloud forensic from local device levels. As a result, IoT forensic investigators will face the cloud forensic challenges described by [10], [11].

The security weakness in IoT applications can contribute to the digital forensic investigation process by allowing the acquisition of data from smart home IoT data sources. This thesis aims to identify how four smart home IoT application developers secure user data and how those security techniques challenge the data acquisition process from the IoT ecosystem for digital forensics investigation purposes. For this thesis, we investigated two AI speakers (SKT Nugu, Naver Clova) and two smart home kits (Xiaomi and Sen.se Mother).

In the next sections, we will present the IoT ecosystem security requirement, thesis statement, and research questions.

1.2. IoT Ecosystem and Security

As stated in the introduction section, we define the Internet of Things (IoT) as the interconnection of intelligent devices with actuators and sensors through communication networks to automate and minimize manual interactions with physical objects [5]. Smart home automation, healthcare services, manufacturing, power grids, transportations, and smart cities are the main application areas of current IoT devices.

Regardless of the difference in application areas, most IoT devices connect to the Internet connection and exchange data through backend cloud. Most IoT device architectures are composed of: (1) backend cloud to process and store data, (2) sensors and actuators to sense and trigger certain actions, (3) hubs to facilitate interaction among sensors and cloud, and (4) companion/client applications (visualization and monitoring applications) to facilitate user interactions to the devices and data. Figure 1 shows a simplified IoT architecture with the main components.

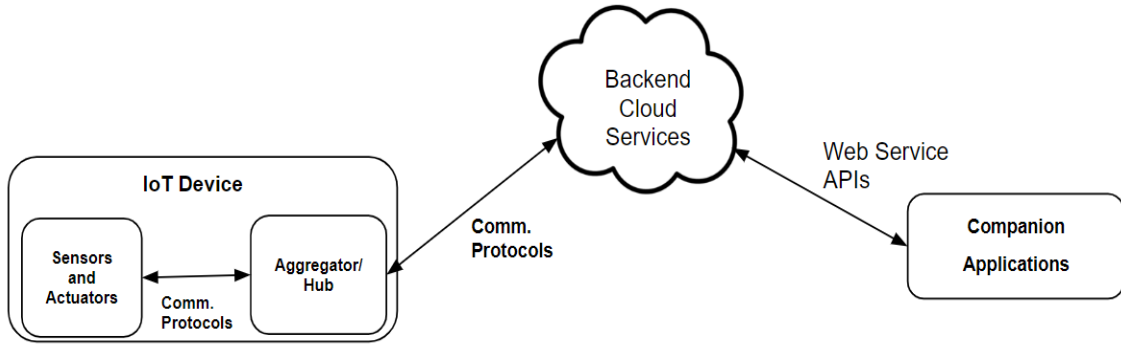


Figure 1: Simplified IoT Architecture. The backend cloud serves as a bridge between the IoT devices and the Companion Applications using Web service APIs to provide the required interface for data exchange and management services.

1.3. IoT Security Framework

There are multiple initiatives from specific governments (UK, ENISA, NIST, and Japan), industry alliances (Embedded Microprocessor Benchmark Consortium (EEMBC), International Electrotechnical Commission (IEC), IoT Security Foundation (IoTSF)) and Open communities (such as OWASP) to develop IoT security standards and guidance. For this research, due to the comprehensiveness of the guidance for IoT application developers from all IoT ecosystem aspects, we will use the OWASP's IoT Security Guidance. The guidance provides the basic level of security requirements to be considered during IoT application development. "OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted [12]. In this thesis, the usage will be limited to the scopes.

In this document, wherever claims that state the “recommended security” is used we explicitly refer to this standard.

Since the goal of this thesis is to identify security methods implemented to protect user data from the client side, we will discuss general data storage and access protection mechanisms.

1.3.1. Companion Apps

In IoT architecture, data visualization and monitoring apps are called companion apps. They are the interfaces between the IoT ecosystem and the user. They serve as a setting interface, managing IoT devices and generated data. In some IoT devices, they are also used to issue commands. They store user information such as account information required for registering the device and depending on the developers; some companion apps also cache user data. Mostly the cache data is a copy of the user data saved in the cloud.

Companion Apps can be developed in different flavors such as Android, iOS and Windows. Moreover, some developers may provide a web-based user interface to achieve the same functionality as the companion apps do. For this research, as specified in the scope, the Android type of Companion Apps of the selected devices will be investigated.

In the Android OS data storage model shown in figure 2, applications save data in their own space in the data section of the built-in flash memory. Based on this model, the OS provides a data isolation mechanism for each app. That means access to specific app data by another app is not allowed unless explicitly allowed access to the data. To achieve this control, there are two kinds of access control models in Android OS. Older Android versions (below version 4.4), use the Discretionary Access Control (DAC) model. In this model, the owner decides what information should be shared to other apps during run time. However, since this model is dependent on the user’s willingness, less security savvy users grant too many privileges to apps, which makes them vulnerable to attacks. Moreover, in DAC, granted permission can be transferred to other apps. To overcome these issues, the Android OS version above 4.4 uses the

Mandatory Access Control (MAC). In the MAC model, the system decides access to the app's data based on the privilege level they have [13]. This privilege is determined during the installation of the app by using the grant types specified in the App Manifest file. App Manifest file (commonly known as AndroidManifest) file is an Extensible Markup Language (XML) format file associated with each Android Apps, which describes the information of the App including the required and allowed permissions [14].

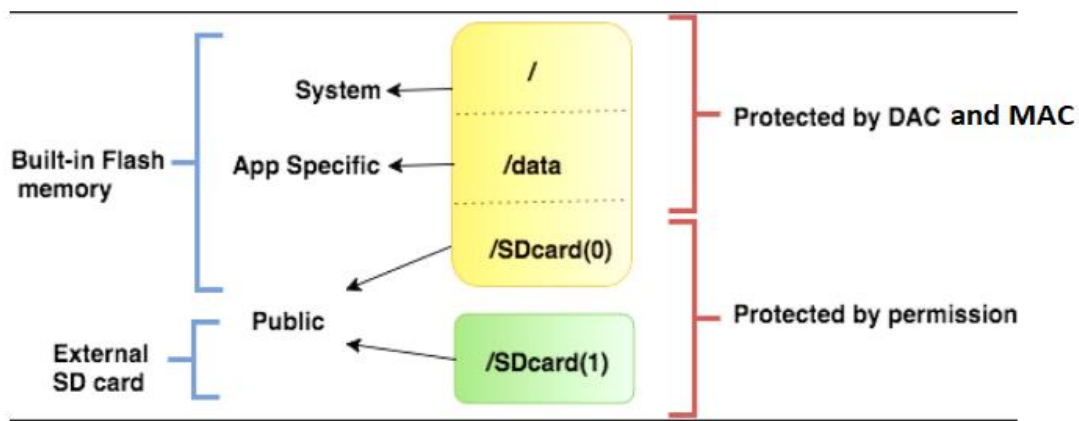


Figure 2: Android data storage model by Altuwaijri and Ghouzali in [13]

In general, Android provides four different options for developers to save data based on the particular app's requirements, such as data size, type of data, and access level (private or public mode). Databases, SharedPreferences, internal and external memory are options available for developers to save data [15].

In Android OS, databases are used to save structured data. SQLite is a database that is used to save data in Android Apps. These databases are created in private access level by default (i.e. other apps do not have access to another app's database). The private access mode is an access control mechanism used by the Android OS to limit apps from accessing another app's data without the explicit grant. If the apps want to share the data stored in the database, they have to use access granting mechanisms and implement access interfaces provided by Android OS.

SQLite databases are used to save user information such as usernames, email IDs, passwords, logs, etc.

The other data storage option is a shared preference. They are used for data types that do not need a structured format. In shared preference, the data is stored in XML format with a key and value pair structure.

On the other hand, for temporary data, internal and external storage options are recommended. In the case of the internal storage, data is saved in the inbuilt flash memory and is accessible only by the app itself [16]. However, the system may clear these data during memory recovery operations, even if the developer did not implement clearance methods. In the case of the external storage options, the files are saved in extended storage, which is managed by the user of the device. Also, files saved on external storage are accessible by other apps without any access grant mechanisms [15].

1.3.1.1. Companion Apps Data Storage Security

As stated above, companion apps are interfaces to IoT devices and user data management. User information used for registering the device and cached cloud data is stored on these companion apps. As a result, this information should be protected from malicious operations. Using the default private access mode protects access to data. However, the protection breaks in case of rooted phones or malware that run at the root level. Therefore, implementing data encryption technologies on the top of the PRIVATE MODE is recommended to protect this data [16].

Similarly, Android provides full disk encryption options for versions above Android version 4.4 [17]. However, according to the Open Web Access Security Project (OWASP) report, insecure data storage is one of the top 10 leading security issues in leaking sensitive information from smartphones [18]. Therefore, in addition to Android full disk encryption and private mode access control, IoT companion app developers should implement an additional layer of encryption to protect user information stored on Android storage [16].

Currently, Instant Messaging Apps such as Telegram [19], WhatsApp [20], Kakao [21] are using data encryption techniques, though the strength of the implemented security is another dimension to consider. As these kinds of data protection implemented to IoT companion apps, data acquisition and analysis will become more challenging for digital forensics investigators. However, many IoT companion app developers are not implementing necessary security mechanisms to protect user data. As already mentioned, these security weaknesses are safe gateways for digital forensics investigators to acquire and analyze user data from the Apps.

On the other hand, as stated above, the companion apps can store cache data downloaded from the cloud, which is limited both in size and duration. Therefore, to get the complete version of the data, there should be some access method to the original data saved in the cloud. This is where security implemented on cloud interfaces play a challenging role in the digital forensic investigation data acquisition process. In the next section, we will discuss what these interface methods are, and the recommended security methods. Lastly, we will state how vulnerabilities in those security techniques can be exploited for digital forensic investigation purpose.

1.3.2. IoT Backend Cloud

From an IoT devices data storage and processing perspective, the backend cloud plays a significant role. It communicates to the IoT devices and client apps to store and process data and perform device management. The interfaces between the client apps and the cloud are called Application Programming Interfaces (API). They are used to transport data and commands between the clients (companion apps) and the backend cloud service and infrastructure.

APIs are a new technology trend used to combine web applications easily. They are interfaces designed in a way that facilitates the transport of structured data between communicating client applications and web servers without revealing much of the underlying working principles. Figure 3 shows a service-oriented architecture API model. The web site [22] says:

“APIs/information hiding allows for the creation of a minimal interface that is relatively stable that can be used by other software systems to access or manipulate the underlying systems or data. This allows for enhancements to the underlying systems or data without disturbing the software systems that use the API” [22].

Often web-based services’ APIs are designed either in a Service Oriented Architecture Protocol (SOAP) and Representational State Transfer (REST) format. REST is an architectural style to represent the state of the resource at a specific time in a hypertext format [23]. REST APIs are preferred for stateless and limited resource web service implementations [24]. Standard HTTP methods - GET, POST, PUT and DELETE are used for REST APIs. JavaScript Notation (JSON), XML, HTML and plaintext are the resource representation formats used for REST APIs.

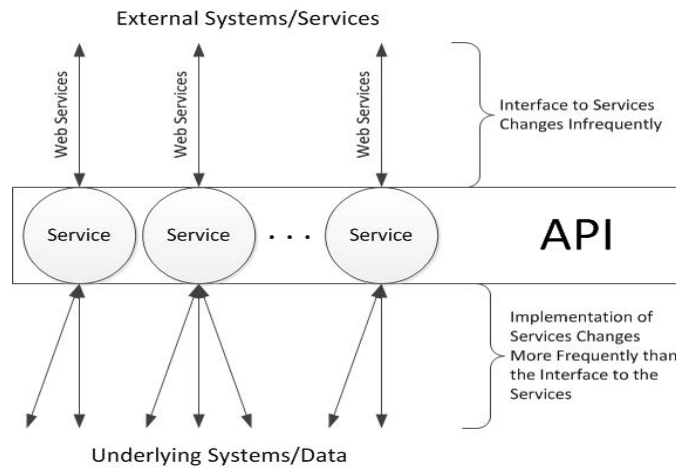


Figure 3: Service Oriented Architecture APIs (From service-architecture.com). APIs are used to interface the underlying systems/data stored in web servers to the client-side applications in the external systems.

For IoT backend services, XML and JSON are common data representation formats used for REST APIs. These APIs can be designed to be public, which is officially available for anyone or private APIs which are used only by the service provider client applications.

However, as they are the gateways to the cloud data or systems that are running behind the front end, they also pose security risks to both the data and systems they interface. As a result, proper security mechanisms should be considered during designing and implementing these APIs.

1.3.2.1. Backend Cloud Data Interfaces/APIs Security Methods

As stated above, one of the main benefits of developing RESTful APIs is their availability for client-side implementation without any specific language requirements [25]. Since these APIs are interfaces to data and other resources, the security aspect should be adequately addressed. Attacks such as credential stuffing – an automated trials of compromised credentials to gain access to systems [26], fuzzing - forcing systems to replay important information using random data [27], and data exfiltration through replay attacks using stolen cookies and stolen tokens happen due to lack of proper APIs security implementation [28].

The underlying security requirement for REST APIs is an access control mechanism. That is, every REST API request should be validated by the server using credentials. To better secure the APIs, these credentials should not depend on cookies or sessions for authorization and authentication. That means, they should be stateless [29]. Therefore, while developing APIs, the following issues should be addressed to validate the requests [25].

1. **Identity:** identifying who is trying to access the protected resource and what methods are used to verify the identity of the user.
2. **Authentication:** verifying if the identified user has proper credentials to access the protected resources.
3. **Authorization:** determining which resources and what actions are allowed for the user to perform on the protected resources.

In order to achieve these security requirements, applications can use different techniques to identify and authenticate users. Table 2 summarizes the available options.

The selection of which technique to use depends on the required security level, the type of access, the type of requesting client and the available infrastructure. However, in recent implementations, due to the security and the capability to use third-party resources for account

management, OAuth protocol has proven its use in current web services authorization and authentication method. Its version 2, OAuth 2.0, is the dominant technology recommended for many REST API authorization and authentication techniques. In the following statements, [30], stresses the advantages of using OAuth 2.0 for current API based applications.

“Use the latest and greatest OAuth - OAuth 2.0 (as of this writing). It means that Web or mobile apps that expose APIs don’t have to share passwords. It allows the API provider to revoke tokens for an individual user, for an entire app, without requiring the user to change their original password. This is critical if a mobile device is compromised or if a rogue app is discovered. Above all, OAuth 2.0 will mean improved security and better end-user and consumer experiences with Web and mobile apps”.

Table 2: Summary of API authorization and authentication options

| ID | Method | Description |
|----|-----------------------|--|
| 1 | Username and Password | Using the basic or digest authentication methods, a pair of username and password can be used to identify and authenticate users. |
| 2 | Sessions | After the first authentication is done using username and password, subsequent sessions can be authenticated using cookies from the first session. |
| 3 | Certificates | Digital certificates generated by mutually (client and server) trusted authority can be used to identify and authenticate users. |
| 4 | Open Authorization | The OAuth protocol can be used to authenticate users using another application (can be third-party) authorization servers. |
| 5 | Custom authentication | Proprietary protocols can be used to identify users and authenticate them to access the resources. |
| 6 | API keys | Tokens (a random and unique set of characters) generated by the server can be used to authorize and authenticate users during the first session. |

From the above description, we understand that the access control mechanisms are critical security methods to consider the resources behind APIs. However, the stated security techniques address one aspect of the security, which is protecting data at rest. However, the security aspect should also consider data confidentiality and integrity during transportation [31]. In general, APIs should be developed with additional security considerations such as using SSL/TLS, password hashing mechanisms, not sending tokens, username, and password as a parameter in the APIs, and validating the input parameters and the number of requests at the server side.

As shown above, communication security is one of the security dimensions to consider during API design to ensure the confidentiality and integrity of the data exchanged between the client applications and the cloud. In the next section, we will present what kind of communication security methods are required to protect data exchange between IoT endpoints; cloud to client apps and cloud to IoT devices.

However, in spite of the above recommendations, developers are negligent in considering the holistic view of the security during the design and development lifecycle of IoT devices and their ecosystems. This negligence that creates security weaknesses in protecting the data and systems the APIs interface is one benefit, however, is to the digital forensics investigators who need to acquire user data from IoT backend Clouds.

1.3.3. IoT Communication and Security

In IoT devices and their ecosystem, data communication includes communication between the companion Apps and the cloud, between the device and the cloud, between the device and the sensors, and between the device and the companion smartphone apps. Except for the security between the companion Apps and the cloud, which is commonly SSL/TLS, all the other communication channel security methods depend on the implemented communication method and protocols. For instance, if the communication method between the sensor and the device is Zigbee [32], Zigbee related securities are recommended. Also, for the device to cloud communication security, the implemented protocols affect the security methods chosen to protect the communication. However, the baseline requirement in all cases is, developers should consider implementing the safest security protocols available for each communication method selected starting from the design stage.

1.3.4. IoT Hubs and Sensors Security

These are embedded smart objects which include the hub, the sensors and the actuators that interact with the users and the physical objects they automate [33]. These devices are the primary

data sources - that collect, aggregate and transmit the data either to the cloud or in some cases, directly to companion apps. As a result, depending on the capability of the device, they store and preprocess user data which should be protected. However, since they are out of the scope of this thesis, detail description is not provided in this document.

1.4. Thesis Statement

In this section, we presented the thesis statement, the general and specific research questions, and the scope of the research. In this thesis statement, we claim that:

Smart home IoT application developers do not comprehensively design and implement securities in the IoT ecosystem. This means that some devices may implement strong security in one component of the ecosystem (e.g. client Android app security) and weak security in another part (e.g. access to the backend cloud used to store and process data generated from the devices). As a result, these security weaknesses can help digital forensic investigators to acquire digital evidence from smart home IoT ecosystems.

In the following section, we give research questions to frame the approach of the study.

1.5. Research Questions

The thesis aims to identify the current state of the selected smart home IoT application security implementations from digital forensic investigations point of view. In order to address this, we will discuss the following questions.

RQ1.1: How are the selected smart home IoT developers implementing user data protection?

RQ1.1: What are the various data (at rest, in process and transit) protection techniques implemented in IoT Companion Apps of the selected IoT devices?

RQ1.2: How do these securities affect digital forensic investigation?

RQ2: How are selected smart home IoT developers handling communication security in the selected smart home IoT devices?

RQ2.1: What security is used in selected smart home IoT devices to protect user data exchange between cloud and client App; and between device and cloud?

RQ2.2: What are the various security techniques used in selected smart home IoT devices cloud APIs?

RQ2.3: How do these securities affect digital forensic investigation?

RQ4: How can we access smart home IoT cloud data using APIs with the identified security techniques? The purpose of this question is to demonstrate the cloud acquisition process using the security weaknesses in two of the selected devices.

RQ5: What are the privacy implications of the security weaknesses and what could be done to protect the privacy of the IoT users?

1.6. Contribution

The purpose of this thesis is three folds. The first is providing a picture of the current state of the security techniques implemented to protect access to user data on the selected smart home IoT devices. The second is to provide insight into the design trend of these security techniques and implications to digital forensic investigation. The third is to demonstrate how these security weaknesses can be used to acquire user data from the IoT ecosystem; we will develop a simple cloud data acquisition tools for two of the selected devices.

1.7. Thesis Scope

Security analysis on smart home IoT devices and their ecosystems cover a wide range of features. For instance, even associated client Android Apps security analysis can be broad, which cannot be covered by this thesis alone. However, since the focus of this thesis project is analyzing the implemented IoT security from a data acquisition perspective for digital forensic

investigations purposes, the scope of the security investigation on the selected IoTs and their ecosystem is treated accordingly. The scope of the research covers Android app forensics and security analysis, network communication analysis and analysis of cloud API security techniques for the selected study devices. Finally, a cloud data acquisition tool for two devices will be provided to demonstrate our investigation results.

1.8. Thesis Structure

The thesis is organized in 6 chapters. The **first chapter**, which is this chapter, covers the introduction to IoT devices and the requirement to conduct digital forensics investigations on IoT devices and their ecosystem. The background research on the security requirements of user data protection for client-side access methods is also included in this chapter. Finally, research statement, research questions, and scope of the research are also included. The **second chapter** presents a literature review of previous works and driving points towards the research. The **third chapter** presents the research methodologies and followed procedures to conduct the investigation. **Chapter four** presents the survey study results and the analysis of the results. **Chapter five** presents a discussion of the results and implications of the results to IoT forensic investigation and privacy of the users. A brief overview of privacy protection methods are also included in this chapter. The cloud data acquisition tools for the selected devices are also included in this section. The **sixth chapter** presents the conclusion and possible extension areas of the research. Finally, references and abstracts both in English and Korean language are also included.

CHAPTER 2. BACKGROUND RESEARCH

Before developing the research, design and executing the investigations, we conducted a literature review on the specific areas to assess the overall threats, already happened data breaches, and existing studies to determine the research directions. In this chapter, we presented the selected literature reviews on general IoT ecosystem securities and IoT forensic investigations.

2.1. Smart home IoT Devices Security and Privacy Issues

2.1.1. General IoT Security Issues

Smart home IoT devices are devices connected to a different network to facilitate the daily operation of users by collecting and communicating the data with the users through the Internet. Most of the information collected by IoT devices is related to personal information, such as date of birth, location, budgets, daily activities, and health records. As a result, the need to secure them is not a matter of choice but rather a critical step that needs to be addressed throughout the life cycle of the devices [34], [35]. However, as recent attack vectors and researches indicate the security solutions implemented in IoT lack comprehensiveness to protect the security of the devices and the privacy of users.

Sicari et al. in [36] presented a survey on the undergoing researches and existing solutions for IoT security and privacy protection techniques and providing research directions to be addressed to ensure the security of IoT devices. From authentication, authorization, access control, privacy, trust, secure middleware, policy enforcement, and mobility security point of view, they identified that there is a lack of unified insight to guarantee the privacy of users, security requirement of IoT devices and their diversified underlying systems.

Airehrour et al. in [37] claimed that existing IoT routing protocols such as 6LowPAN and RPL have limitations of security. Lack of standardization in secure routing between IoT impacted the

level of security in the protocols. They also stressed the need to balance the network and power consumption of the nodes while addressing the security in IoT routing protocols.

Yang et al. in [38] stated that the safety of IoT devices is affected by individual manufacturers involved in implementation, the protocols and the security techniques in the devices. Based on their survey on security and privacy of IoT devices, they pointed out that, all IoT devices could be affected by certain types of attacks depending on the particularity of the case.

Ling et al. in [39] presented analysis on the end to end security of IoT devices from the device, the cloud and the controller (monitoring device) point of view based on ten functionalities that need to be secured properly. Then they demonstrated using a case study on Edimax IP camera, in which they discovered vulnerabilities and exploited them to gain access to the camera. The identified vulnerabilities enabled them to control IP cameras. For instance, the spoofing attack enabled them to obtain the user's password for the device. The other attack scenario included the physical access of the device, in which using the reset button on the camera, the attacker could control the camera after changing the password. Finally, they stressed the need to address IoT securities broadly, to avoid massive attacks such as Mirai Botnet.

Ammar et al. in [40] presented a survey on the security of 8 commercially available IoT frameworks. AWS IoT from Amazon, ARM Bed from ARM and other partners, Azure IoT Suite from Microsoft, Brillo/Weave from Google, Calvin from Ericsson, HomeKit from Apple, Kura from Eclipse, and SmartThings from Samsung are the surveyed frameworks. Although the companies followed the same design approaches towards data aggregation methods and communication security; the implemented securities and the underlying technological solutions are quite different. However, regardless of the securities implemented in each of the frameworks, the authors showed that some of the frameworks have security issues that may arise from the design and implementation of the solutions. For instance, some of the embedded devices depend

on the commercial of the shelf chips which do not address hardware security, while some embedded the keys to devices before deployment.

Siboni et al. in [41] proposed an IoT security testbed system, a system with a collection of open source vulnerability assessment and penetration tools in an integrated format. The integration is based on task flow in which one tool's output is fed to another tool as input to perform further analysis. One of the interesting modules of the system is the context-based simulation module used to automate the data generation for the test. For instance, there is a robot module that can produce motion for motion sensors testing. The system also has an advanced testing module which is based on the application of Machine Learning techniques. Finally, the reporting module produces the results in a document format to be exported. Based on the practical implementation of the system, they presented an analysis of selected IoT devices. Their analysis shows that all of the analyzed IoT devices are vulnerable to at least one already known vulnerability. The system is very interesting to start as a development and analysis but considers IoT devices from only devices level, not the ecosystem in general. For instance, the system does not consider the weak points in the client Apps or the cloud running as a backbone for the IoT that will endanger the security of the IoT and privacy of the user. Moreover, the system does not address the end sensors except the communication network between the sensors and their hub.

However, all of the above works focus on the attacker's perspective, which does not consider the data acquisition process for digital forensic investigation purposes. For instance, none of the above works demonstrates how the one-time authentication and authorization of cloud APIs affect the data acquisition process for digital forensic investigations. Moreover, the above studies did not address how IoT devices' monitoring Apps handle user data on smartphones.

2.1.2. IoT Companion Apps Data Security

Liu et al. in [42] presented a survey on 17 popular Android Apps data storage security techniques to determine how developers follow security recommendations to protect the user's private data.

Their result shows that most developers do not implement the required security. Even those who apply tend to implement it in the wrong way. Moreover, in some case, they do not properly categorize sensitive information due to lack of developers' awareness of privacy and security issues. Their research is mainly focused on data stored in shared storage. However, as recent attacks trends show, even data saved in the apps, private storage is not immune to attacks.

Jain et al. in [43] presented two SQLite database vulnerabilities in Android Apps by manually analyzing the apps on a rooted Android phone using OWASP threat modelling. The reason they chose the manual analysis is due to the difficulty of the analysis because of the difference in the structure of the app's database. They analyzed different messaging apps and presented the result based on their threat modelling. The two vulnerabilities they focused on are on storing sensitive data in plaintext and on the synchronization procedure of the apps. The authors performed risk analysis based on the two vulnerabilities on the different application as a demonstration using the OWASP threat modelling approach. Specifically, they analyzed the naver-line chat service app and demonstrated a spoofing attack by changing the attributes of the database. Finally, they stressed the need to encrypt sensitive data on SQLite databases and perform regular synch of the data to avoid the tampering of data which may, for instance, result in an identity spoofing attack.

2.1.3. IoT Backend Cloud APIs Security

Rodríguez et al. in [44] presented analysis on a set of data from Mobile Internet traffic traces from a telecom company to determine how developers follow REST theoretical principles and guidelines to design APIs used for mobiles. They developed a set of approaches and measurement criteria that enable to measure API maturity levels. They found out that the practical implementation and usage of most web service REST APIs are not consolidated and standardized based on the recommended theoretical REST architecture styles. But their analysis is limited to the architectural style of REST APIs and did not consider the implemented security of the APIs.

Petrillo et al. in [45] presented a case study on three cloud service providers to determine how much of the recommended best practices they followed during designing REST APIs. Google Cloud Platform, OpenStack and OCCI were the subjects of the study. For their research, they catalogued 73 best practices to be considered during API designs. According to their analysis results, from the 73 best practices, GoogleCloud followed 66% (48/73), OpenStack followed 62% (45/73), while OCCI 1.2 followed 56% (41/73). However, their focus was on the understandability and usability of the APIs, where the security part is given less attention and skipped from the analysis.

Chung et al. in [46] presented API analysis on Amazon Alexa and showed that forensically-relevant data could be obtained from Echo speaker and its ecosystems (the cloud and companion client applications). For Echo investigation, most of the data can be acquired from the cloud using the unofficial APIs and cookies discovered through the communication analysis between the user web interface and the Amazon cloud. However, they took advantage of the user web interface available to the users, which may not be the case to all smart home IoT devices. Moreover, their focus is on the extraction of data from the cloud using the APIs, not the security of the APIs.

Kanmani in [31] presented a survey on the limitation and benefits of different techniques used by REST with the OAuth authorization protocol. The author was focused on how the OAuth protocol is used to authenticate REST based services exploiting the provided advantages such as service addressable, interface consistency and resource caching of the REST protocol. The survey of the paper is limited to providing analysis of the technology.

In contrast, in this thesis, we focus on the analysis of the user data protection techniques in the companion apps, communication securities, and APIs security, specifically those used between IoT and backend cloud service providers. More specifically, the analysis focuses on the implemented storage securities, channel security methods, authorization and authentication

methods used for selected IoT devices smartphone companion Apps' APIs. Moreover, the communication security between the IoT device and the backend cloud.

2.2. Forensic Investigation of IoT Ecosystem

As stated in the introduction section, IoT devices collect user data that can be useful for digital investigation purposes in investigating crimes that occurred in smart homes. Some IoT devices had already proved their usefulness in [47], [48].

Rahman et al. in [49] created smart home scenarios and analyzed the forensic relevance of the data collected by Sen.se Mother and its Cookies (motion sensors). In their scenario, they showed that data collected from different Cookies could be applied to different investigation cases. However, their research was limited to the analysis of the data by accessing the cloud data through the user web interface provided. The only security they have to consider was getting the username and password from the user. Therefore, their work did not address the extraction of the APIs, and how the user data can be obtained in case of the username and password are not available.

Chung et al. in [46] showed that forensically-relevant data could be obtained from Amazon Echo speaker and its ecosystems (the cloud and companion client applications). For the Echo investigation, most of the data can be acquired from the cloud using the unofficial APIs and cookies discovered through the communication analysis between the user web interface and the Amazon cloud. On their work, they also included a tool for IoT forensic approach (cloud-based IoT Forensic Toolkit) to indicate the process of acquiring forensic relevant data from Alexa and its ecosystem based on the extracted APIs and cookies. Also, they included artefacts from the companion App installed on smartphones. However, their research took advantage of the user web interface available to the users, which may not be the case to all smart home IoT devices.

Regarding the forensic investigation of gateway devices such as routers deployed for IoT devices and home networks, authors in [50] presented work on live forensics analysis using APIs

to investigate network attacks on MikroTik RB750 RouterOS. But, their forensic examination of the router was limited to a specific device.

Kebande and Ray in [51] proposed a generic Digital Forensic Investigation Framework (DFIF-IoT) for IoT devices based on three processes, Proactive, IoT forensics and Reactive process. Each process has its own mini processes. The proactive processes address the digital forensics readiness process to make the environment forensically prepared. The IoT forensics process addresses the actual forensic activities using already accepted methods across three domains, cloud forensics, network forensics and device level forensics. The third process, the reactive process addresses the required processes to investigate IoT devices after an incident, is identified.

Akatyev and James in [52] proposed a model on how to approach smart home IoT devices digital forensics investigation based on threat assessments on intelligent smart home IoT deployments. After conducting a threat assessment on the assumed smart home environment, they pointed out investigation focus areas (devices) based on the type of the threat.

Kebande et al. in [53] proposed a Digital Forensic Readiness Framework (DFR-IoT) for IoT devices to facilitate the digital forensic investigation process in IoT devices ecosystem if an incident is detected. The framework proposes three processes, the Proactive Process which addresses the required activities before incident detection; the IoT Communication Mechanism addresses the activities for IoT automated communication and the Reactive Process which focuses on activities after an incident has been detected.

2.3. IoT Forensic Challenges

Authors in [10], [11], [54], [55] pointed out challenges to general digital forensic investigations due to both technical and legal aspects of the area. From technical challenges, encryption, cloud computing and data hiding techniques lead the challenges. James and Jang also stressed the challenges faced in conducting cloud digital forensic investigations due to the difference between

jurisdictions of hosting and investigating authorities. They also addressed the technical challenges due to the deployment architecture of the cloud itself. Unfortunately, as IoT forensics relies on those technologies, most if not all of the challenges stated will apply to IoT forensic investigation.

Oriwoh and Sant in [56] proposed a comprehensive 1-2-3 zone and Next-Best Triage (NBT) IoT forensics model due to the unique nature of the IoT devices investigation. The dispersity and location of the devices and evidence, number and interconnection complexity of the devices, the amount of data, lack of clear cut between network borders, and the cloud dependency of the IoT for data storage are some of the challenges. In their model, they proposed a divided approach based on the network setup of the devices. However, their model is focused on the high-level challenges leaving the security challenges faced in each zone.

Rughani in [9] divided IoT architecture into 3 (cloud, network, and endpoint) and focused on the challenges faced during the data acquisition process from the endpoints. The visibility of the IoT devices, mapping the complete network due to the distributed nature of sensors, keeping the integrity of the evidence from sensors, deciding the relevance of the data to image a specific device and the mobility of the devices are the challenges stated by the author. Though this focused on the endpoints, the author did not consider the security on the devices.

2.4. Summary

In this chapter, we presented the literature reviews on the specific areas to assess the overall threats, already happened data breaches, existing researches, and to determine the research directions. From the study, we understood that IoT developers are not considering the cybersecurity issues in a holistic view and lack addressing it comprehensively from the design and development stage, which reduces the challenges to IoT forensic investigators.

CHAPTER 3. RESEARCH METHODOLOGY

A research methodology is a framework that guides how the research is going to be carried out, the methods and procedures to be followed, and the subject of the investigations. This thesis is a technical analysis based on research works that are lean to the qualitative research method. In qualitative research, analysis is done based on assumptions and reasoning on the gathered data combined of texts, images, audios, and other information. For the research to address the specified statement and answer the questions listed in the research questions section, we collected data by performing a technical investigation on the case study subjects. Case study research methodology is a scientific research method developed to suit scientific researches that require designing experiments and investigating selected subjects to formulate a theory for the broader set [57].

In the following subsections, we presented the research procedures, the selected study devices, the network setup, and assumptions for the research.

3.1. Followed Research Process

Since the thesis is based on the technical investigation of the devices, the first activity we did was conducting a literature review of the subject area. After the literature review for the general contexts and the specific case study devices, we dive into the technical investigations of each case study device. As shown in figure 5, we approached the research from 4 dimensions. Each of the four dimensions is described in detail with respective procedures. Since the study includes a demonstration of the findings on the case study devices, the investigation results led to two activities. The first one is developing a simple cloud data acquisition tool for the selected IoT devices. The second and final stage of the process is writing the thesis.

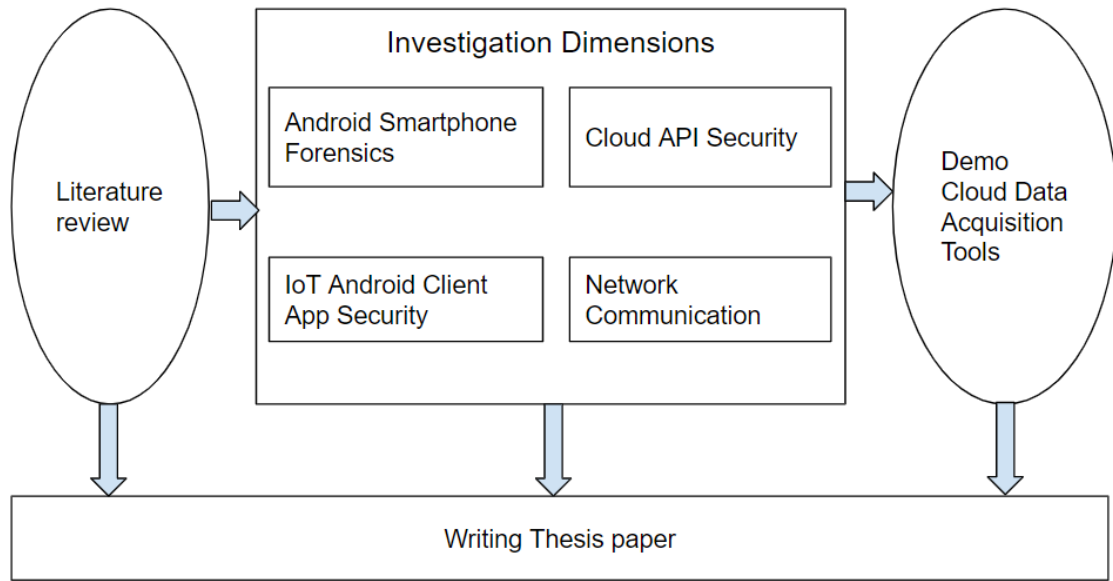


Figure 4: Followed research process. A first literature review was done before investigating the IoT devices. Both activities are used to develop the thesis. Finally, the sample demo tool is developed.

3.2. Followed Research Procedures

In this subsection, we presented an overview of the investigation dimensions we followed along with the high-level procedures followed and tools used to conduct the research.

3.2.1. IoT Android Companion (Client) Apps Investigation

Most of IoT devices come with companion mobile applications, which are used to set up and manage the device. Moreover, these apps are also used to access data stored and processed in the cloud or in the devices. Since APIs are the transporters of these data from the cloud to the app cache, analyzing the apps directory in the smartphones reveals information such as the cookies, authentication tokens and in some cases used APIs. Most of the time this information is saved either in the XML format in the shared preference directory or in the SQLite database format in the database directory of the apps.

In some cases, analyzing the cache files may also reveal the requested APIs. However, if data protection techniques are implemented to protect user data, access to this digital evidence is

hindered by securities. In this research, we conducted an extensive analysis of the data protection techniques implemented in the apps to protect user data while it is at rest and in transit.

We approached the analysis from three different sectors. The first part addresses the security techniques implemented in the app to protect user data saved on the smartphone storage. To achieve this analysis, we conducted a combination of app security and app forensics analysis. We performed static and dynamic analysis on the app using the Android reverse engineering approach to identify the security methods implemented. For the forensics analysis, we conducted live forensics on the app installed on Samsung Galaxy Note II with Android version 4.4 (KitKat). The smartphone is rooted for the purpose of this research. Moreover, we analyzed the app from network investigation perspectives using traffic analysis through Wireshark and Man-in-the-Middle attack approaches.

3.2.2. Companion Apps Security Analysis using Reverse Engineering

Reverse engineering and hooking are the two widely used techniques to analyze applications; understand the way they work and assess security of application through app analysis using static (code analysis) and dynamic analysis methods. The static analysis, decompiling and analyzing the apk file reveals what kind of shared preference files, databases, tables, and what kind of security mechanisms the app uses and how the app constructs the APIs. To analyze the source code, we used a combination of adb, apktools, MobSF and java decompilers. Moreover, the dynamic analysis helped us to understand the behaviour of the apps during operations. Analyzing the real-time activity of the app using dynamic analysis tools, we were able to identify the created files, called security methods, database operations, and cloud request operations. We used tools such as Inspeckage with the Xposed framework to perform the dynamic analysis.

The general procedures we followed to reverse engineer and analyze Android applications for the research are:

A. Static Analysis

1. Download the app from the smartphone using adb operations or download the app from the Google Play service. For our research, we downloaded the installed apps from the smartphone. This enabled us to control the version of the app we analyzed and run on the smartphone.
2. Decompile the app to java source codes
3. Study and analyze the source code

B. Dynamic analysis

1. Install the app on the smartphone
2. Root the smartphone using Android rooting tools such as Superuser.apk
3. Install dynamic analysis (hooking) tools Xposed framework and Inspeckage modules on the smartphone
4. Using the app within the dynamic analysis tools to generate data and analyze the app

Instead of performing, each analysis in a distinctively independent fashion, we combined the static and dynamic analysis together to investigate the companion apps. That means, for instance after performing static analysis for the database security, we immediately followed the dynamic analysis for the database.

For the combined investigation, we used the following procedures and commands as stated below with a brief description of each step.

Step 1: Connect the rooted smartphone with the installed Companion app to the research computer using a cable with USB type.

Step 2: Find the app name and path using adb operations using terminal - to get the path and name of the packages we followed two approaches. The first is browsing /data/app directory using the Linux “cd” (change directory) and “ls” (listing) commands and manually looking for the app’s name. The second one is using a short and efficient way to list installed packages (shell pm list packages -f -3). To achieve the first option rooting of the device is required.

Option 1:

```
#adb devices
#adb shell
#su
#cd /data/app/
#ls
```

Listing 1: adb commands to access smartphone

Option 2 - simply use

```
#adb shell pm list packages -f -3
```

Listing 2: adb command to list installed applications

Output Example:

```
package:/data/app/com.naver.clova.apk=com.naver.clova
```

Listing 3: Example of listing package name

Step 3: Download the app using the adb pull command

The generic command to pull apks from a smartphone is:

```
#adb pull /data/app/package name destination path
```

Listing 4: adb pull command to download installed apps

For example: `rootPath\Analysis\Clova>adb pull /data/app/com.naver.clova.apk .` In the above command, the “.” indicates the current directory as a destination path. The *Output is:* `/data/app/com.naver.clova.apk: 1 file pulled. 4.4 MB/s (23656683 bytes in 5.119s).`

Step 4: decompile the apk file into java source code files. Since apk files are compressed Java class files, unzipping the apk file will dump the apk into “dex” files (java source codes compiled to android binary format) and other resource files compressed together. A detailed description of apk file creation and structure are out of the scope of this research.

For static analysis, we used two methods with different tools. The first method was manual analysis using a combination of tools such as WinRAR, dex to java converter [58] and java decompiler [59], [60]. The steps and commands are stated as follows.

Step 4.1: Rename the file by changing the apk extension to zip extension.

Com.naver.clova.apk into com.naver.clova.zip

Step 4.2: Unzip the renamed file

Figure 6 shows the unzipped apk file with the dex files and other resource files used in the app. These are the compiled java source code files. Converting these files to jar files is required to open and analyze the class files as a java source code. To convert DEX files to jar file, we used the open source dex2jar converter tool on a terminal.

| | | |
|-----------------------------------|--------------------|-----------------|
| assets | 4/10/2019 10:38 PM | File folder |
| fabric | 4/10/2019 10:38 PM | File folder |
| kotlin | 4/10/2019 10:38 PM | File folder |
| kotlinx | 4/10/2019 10:38 PM | File folder |
| lib | 4/10/2019 10:38 PM | File folder |
| META-INF | 4/10/2019 10:38 PM | File folder |
| okhttp3 | 4/10/2019 10:38 PM | File folder |
| org | 4/10/2019 10:38 PM | File folder |
| res | 4/10/2019 10:38 PM | File folder |
| skcom | 4/10/2019 10:38 PM | File folder |
| AndroidManifest.xml | | XML Document |
| androidsupportmultidexversion.txt | | Text Document |
| classes.dex | | DEX File |
| classes2.dex | | DEX File |
| firebase-analytics.properties | | PROPERTIES File |

Figure 5: Example of an unzipped apk file showing the class files in dex format and other resources

Step 4.3: convert the dex files to jar files

```
dex2jar classes.dex
```

Listing 5: dex to jar converting command

Step 4.4: Open the converted jar files using either jd-gui or jadx-gui tools. jd-gui has the capability to link classes and methods, however, it is not capable to decompile the full source code. On the other hand, jadx-gui decompiles better than jd-gui; however, it has no capability of linking the classes.

In addition to the manual decompiling process, we also used an automated tool called Mobile Security Framework (MobSF). It is an all in Android, iOS and Windows application security analysis tool [61]. The advantage of this automated tool is, the capability to parse and report security issues in the apps after performing the decompiling process. The tool also provides the option to download the decompiled jar files for further analysis. Moreover, a browser can be used

to view and analyze the java source codes. However, like the jadx-gui tool, it has no linking capability to follow classes and methods in different files.

Step 5: Analyze the source code. In static code analysis, the goal is to understand how the application works and determine what kind of files created and what kind of user information are saved. Manually searching for keywords and hard-coded scripts and class files are the required activities in the static analysis. In this research, since the goal is to determine how the apps save files and what security techniques are used, we analyzed the source code related to SQLite database, shared preference files specific to the user data, internal and external storage operations.

As stated above, after we performed static analysis for each data storage, we followed the dynamic analysis for that specific data storage. For the dynamic analysis, we used the Inspeckage module on the Xposed framework. Inspeckage is a dynamic analysis tool developed to hook methods in the apps to be analyzed [62], [63].

Step 1: Start the Inspeckage module on the smartphone

Step 2: Access the Inspeckage module with a browser on the research computer using the IP address of the smartphone and port 8008. The URL looks like <http://192.168.166.31:8008> for specific our research setup. In this case, both the smartphone and the research computer have to be within the same LAN (Local Network Area).

Step 3: Start the app to be analyzed within the Inspeckage and perform the usual activities supported by the app, such as accessing user data from the cloud.

Step 4: Browse the required files and activities on the started browser on the computer.

3.2.3. Companion Apps Live Forensics Analysis

For our forensic investigation, we approached live analysis on the smartphones instead of imaging phone. This saved time and storage, which in turn, enabled us to perform the analysis multiple times efficiently. We used tools such as ADB, sqlite3, ES file Explorer to directly access the files and perform the analysis. Connecting the rooted phone to the computer and using the adb shell provided the connection while the sqlite3 command provided SQLite database operation

without extracting the database files. On the other hand, we used adb pull and ES file explorer to extract files from the smartphone. In those cases, we used the SQLite browser and text editors to open and analyze the extracted files.

1. The general steps to follow at this stage of the research, specific to the Android version of the app:
2. Install the app on the smartphone to set up the IoT devices access the data
3. Root the smartphone using Android rooting tools
4. Investigate the phone image using analysis tools.
5. Live forensics on the client Android Apps.

For the live forensic analysis, we used two approaches depending on the convenience of data presence for this research. The first approach is using the “sqlite3” and “cat” command on the smartphone, without downloading the data to the computer. For this method, the following steps are followed:

Step 1: Connect the rooted smartphone with the app using a USB cable

Step 2: Use the adb shell to get command line access to the smartphone

Step 3: Change the working directory to the specific app’s storage section

Step 4: Use appropriate commands for the particular files to be analyzed - sqlite3 commands for database analysis and “cat” for text files such as XML files in shared preference storage.

3.2.4. Network Investigation

Network investigation helps to understand the communication protocols and implemented securities, identify communication APIs (official and unofficial), sensitive information (like credentials or session) and exchanged user data during transit, etc. In network investigation, both live and offline network analysis can be achieved. For this research, we used tools such as Wireshark to understand the network flows, protocols and security methods used to protect the communication between the apps and the IoT device and the cloud. Besides, for the companion apps communication investigation, we used Man-in-the-Middle (MITM) attack approach and proxy tools to intercept the communication traffic and analyze the requested APIs, tokens,

credentials and other user information. For our analysis, the MITM attack method revealed unofficial APIs used between the apps and the respective Clouds along with the API authentication methods and credentials.

Moreover, we used DevTools in browsers to watch and analyze network traffic between web browsers and the cloud. This also helped us to analyze the APIs, the request and response headers, and evaluate the responses of the APIs. In general, during network investigation, we analyzed the network traffic data through both live or offline based on the situation. The network investigation included the following general procedures:

- A. Network traffic analysis between web browsers or web apps and the cloud
 - 1. Using browsers network monitoring tool (DevTools) - watch and analyze the network communication, the request and response headers, and determine/evaluate the responses of the APIs and their security.
 - 2. Using man-in-the-middle attack (MITM) - intercept the communication traffic and analyze it either using live traffic analysis (directly during the communication) or offline mode (capture the traffic and analyze it using other tools).
 - 3. Using Wireshark - for live or offline network analysis by mirroring the traffic without intercepting the traffic.
- B. Network traffic analysis between IoT devices and the cloud
 - 1. Using the Wireshark tool - intercept the communication traffic and analyze it either using live traffic analysis (directly during the communication) or offline mode.

3.2.5. Cloud API Security Analysis and Demo

In order to test the cloud API security, we used the python request library on python version 3.6 environment [64]. After, constructing the APIs, the required header and parameter, either get or post requests are performed to the cloud on behalf of the client apps.

3.3. Demonstration Tool Development

We used a Python script on the Python version 3.6 environment to develop the demonstration for the tool. The tool includes basic functionality such as user interface to select where to save a file and submit the required credentials.

3.4. Required Research Materials and Tools

To conduct our research, the following tools are used at different stages of our study. The tools and systems are acquired as research support from LIFS laboratory and open source.

Table 3: Used research materials and tools

| ID | Name | Source | Relevance |
|--------------------------|---|-------------|---|
| Devices | | | |
| 1 | Samsung smartphone (Galaxy Note II) | LIFS Lab | For monitoring the devices and app forensics |
| 2 | Smart Home IoT devices | LIFS Lab | See Table 1 above |
| Research SW Tools | | | |
| 3 | Burp Suite v1.7.36 and Sandroproxy v1.5.117 | Open Source | Network Analysis |
| 4 | Wireshark v3.0.1 | Open Source | Network Analysis |
| 5 | MobSF v.11 beta | Open Source | Android App Analysis |
| 6 | Other Android App reverse engineering and Analysis tool | Open Source | IoT Android apps reverse engineering and analysis |

3.5. List of IoT Case Studied Devices

South Korea is one of the leading top ten countries in ICT developments. The 2017 ITU index ranks the country in second place [65]. In addition to the availability of the Internet, it has strong ICT research and incubation facilities where new trends of technologies such as IoT devices are adapted and presented to the consumers at faster rates. After Amazon Alexa AI speaker device release in 2014, Korean Telecom, Internet and Electronic companies have been developing smart home IoT devices to dominate the local market. Companies such as Kakao, Naver, SK Telecom, KT, Samsung, and LG are the mainstream developers of AI speakers and sensor-based smart home automation devices.

For this research, the case study devices were selected based on their availability in the Korean market and the availability of the devices in the Legal Informatics and Forensics Science (LIFS) research laboratory. As a result, we selected two AI speaker devices from South Korean Companies Naver Clova and SKT Nugu. In addition, to have a broader context for the research, we also included two multipurpose sensors based IoT devices from outside of Korea, Sen.se Mother from France and Xiaomi smart home from China.

Table 4: Selected smart home IoT devices

| ID | IoT Device Name | Application Type | Companion App | Manufacturer | Country |
|----|-----------------------|---------------------|---------------|--------------|-------------|
| 1 | Sen.se Mother | Multipurpose Sensor | Pocket Mother | Sen.se | France |
| 2 | Naver Clova | AI Speaker | Naver Clova | Naver | South Korea |
| 3 | SKT Nugu | AI Speaker | Aladdin/Nugu | SKT Telecom | South Korea |
| 4 | Xiaomi Smart Home Kit | Multipurpose Sensor | Mi Home | Xiaomi | China |

3.6. IoT Devices Set up Network Design

The selected devices were set up and configured in the Legal Informatics and Forensics Science (LIFS) research laboratory. Since the laboratory is already set up for digital forensics research, we used the existing network equipment such as the gateway devices and integrated the case study IoT devices. Figure 4 shows the network design for the laboratory, including other devices connected to the network.

3.7. Assumptions

For this research, data collection will be conducted from scratch using the specified research methodologies after updating the IoT devices and their Androids apps to the latest version. However, for some of the devices to be analyzed, already existing research results (from LIFS research lab) and collected data will be used due to the challenges to regenerate the data. Among the problems, some of the devices are already disassembled, broken or run out of service.

Moreover, due to the difficulties in rooting and imaging process on the latest Android OS versions, the App forensics investigation will be conducted on a smartphone with Android OS version 4.4 and 6.0. Finally, for the research user credentials (username and password) to the IoT devices are assumed to be available whenever and wherever needed in the research process.

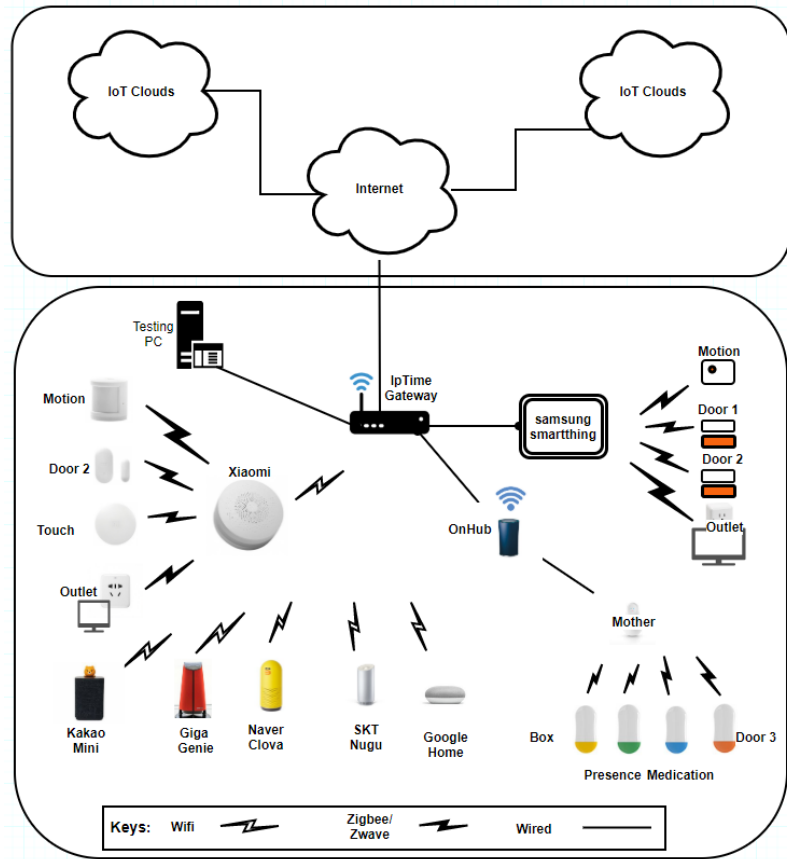


Figure 6: IoT Research network design and configuration

3.8. Summary

In this chapter, we presented the research methodology, the process and detailed procedure we followed in conducting our research. Moreover, we discussed the research tools, selected study devices, the network setup, and assumptions. For app analysis, we will use reverse engineering, live forensics and network investigation using MITM attacks. For the device communication security analysis, we will use traffic network analyzing tools such as Wireshark.

CHAPTER 4. CASE STUDIES

In this chapter, the investigation results of the studied case study devices are presented. Although the order of the studied devices has no values, we presented the analysis result for the AI speakers, first Nugu then Clova and next to the smart home automation devices.

4.1. SKT Nugu AI speaker

SKT Nugu is Artificial Intelligence (AI) enabled voice assistance from the Korean telecom giant SK Telecom. Nugu was first released in 2016. It has two types: Nugu and Nugu Mini - smaller version. Nugu's wake words include "Aria", "Tinkerbell", "Crystal" and "Rebecca". The supported functionalities include Music streaming, Weather, Calendar, Alarm, Reminder, Navigation, number of online shopping, and banking operations. It has SKT Nugu (Aladdin) companion App used to set up, manage the device and access the user's cloud data [66].

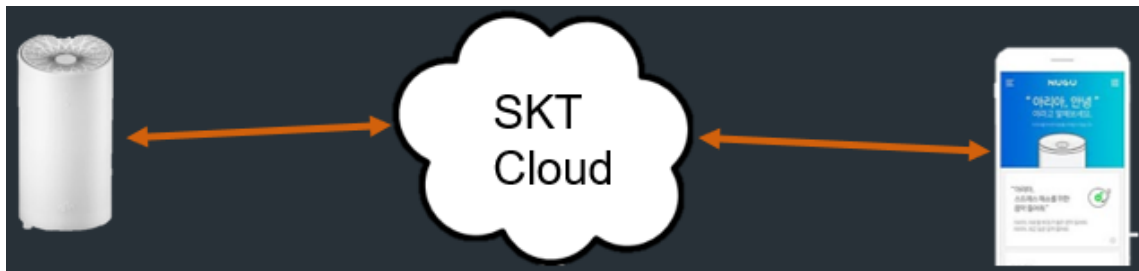


Figure 7: SKT Nugu AI speaker operation mode

As SKT did not provide a Web interface for the device management, SKT Nugu App is the only available application used to set up the device and access user data collected by the device. At the time of conducting this research, the latest version of the app is 2.3.0.

To set up the Nugu speaker, SKT account information required; specifically, TID subscription is required. To sign up for TID, SKT provides the option to use existing accounts such as Google, Naver, Facebook or Kakao and requires birthday information. Once the TID is created, using the Nugu app user can register the device and start configuring it, basically setting the WIFI network for the device. After the setup is complete, using one of the wake words, voice commands can be

issued to the device. Then the app is used to access the ordered services. In addition to accessing data from the cloud, the app can also be used to issue commands such as setting alarms, reminders and other functionalities using texts, instead of voice commands as in the case of the speaker.

As stated above, Nugu App is used to access user information and other ordered services. As a result, these user data should be protected using data protection security techniques. These data protection techniques should depend on the type of data and the location of data within the ecosystem. For, recommended data protection techniques, see section 4.1.

In the following section, the analysis of the data security of the app is presented. As defined in the scope of the research, the analysis focuses on the security techniques implemented to protect user data while at rest and in transit. Apart from the app, the study also addresses the communication security implemented between the speaker and the SKT cloud.

4.1.1. SKT Nugu (Aladdin) App Analysis

Based on the designed research methodology, we approached the analysis from three different sectors. The first part addresses the security techniques implemented in the app to protect user data saved on the smartphone storage. To achieve this, we conducted a combination of app security and app forensics analysis. We performed static and dynamic analysis on the app using the Android reverse engineering approach to identify the security methods implemented and determine what impact they will bring for digital forensic investigation. For the forensics analysis, we conducted live forensics on the app installed on Samsung Galaxy Note 2 with Android version 4.4 (KitKat). Moreover, we analyzed the app from network investigation perspectives using traffic analysis through Wireshark and Man-in-the-Middle attack approach. In the following subsection, we presented the detailed activities and findings of the study.

4.1.1.1. Nugu App Data Storage Security Analysis

For Nugu App reverse engineering, we used static and dynamic code analysis to understand the logic behind the app and how the app behaves at run time. For our study, the first activity we

performed on the app was to perform static code analysis after extracting the app using adb pull from the smartphone. After static analysis, we immediately followed the dynamic analysis to determine how the files are created. Finally, at the third stage, we used live forensics approaches using a combination of the Linux commands and adb operations. For database analysis, we also used the sqlite3 commands. For some of the files, we also exported them to the local computer using the adb pull operation and used tools such as SQLite DB browser and Notepad ++ editor.

Based on the stated procedures in the methodology section, we downloaded the app from the smartphone and first used the manual method to reverse the app. The first step was to rename the file and unzip the file to get the DEX files and then convert them to jar files.

- *rootPath/Analysis\Nugu>adb pull /data/app/com.skt.aladdin-5.apk .*
- *In the above command the “.” indicates the current directory as a destination path.*
- */data/app/com.skt.aladdin-5.apk: 1 file pulled. 4.4 MB/s (23656683 bytes in 5.119s)*
- *Renamed the file by changing the .apk extension to .zip extension.*
- *Com.skt.aladdin-5.apk into com.skt.aladdin-1.zip*
- *Unzipped the renamed file*



Figure 8: Nugu Unzipped apk, showing the class files in dex format and other resources compiled to form the app

From figure 8, we can see that there are two DEX files. These are the compiled java source code files. Now converting these files to jar files is required to open and analyze the class files as a java source code. To convert DEX files to jar file, we used the open source dex2jar converter tool on a terminal. Convert the dex files to jar files:

```
dex2jar classes.dex
dex2jar classes2.dex
```

After converting the DEX files to jar file, we used jd-gui to open analyze the source codes. We also used the jadx-gui for some of the files that were not decompiled due to the limitation of gd-

gui. Since jadx-gui has the ability to convert from apk to java source code, only opening the apk file is enough. The limitation is, it has no capability of linking the classes and methods as jd-gui.

In addition to the manual decompiling process, we also used an automated tool called Mobile Security Framework (MobSF). As stated in the methodology section, the advantage of this automated tool is, the capability to parse and report security issues in the apps after performing the decompiling process. The tool also provides the option to download the decompiled jar files for further analysis. Moreover, the browser can be used to view and analyze the java source codes. However, like the jadx-gui tool, it has no linking capability to follow classes and methods in different files.

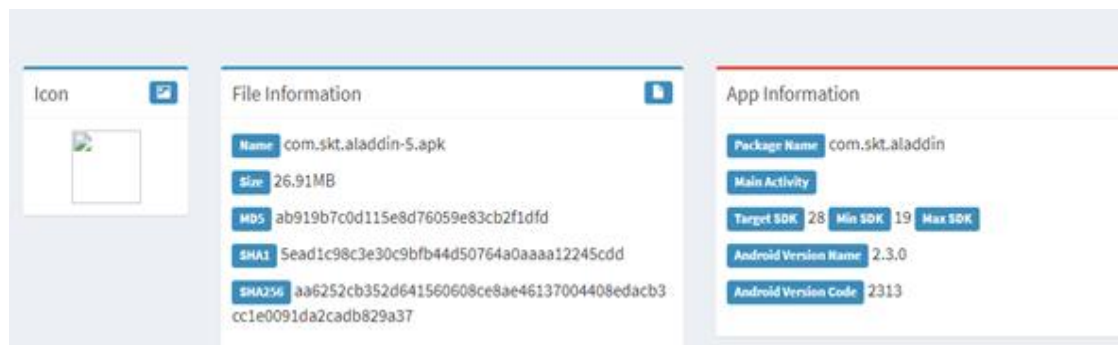


Figure 9: Nugu App information after decompiled using MobSF tool showing detail information about the app including the version and hashes

The following section presents the findings in the Nugu apk static analysis for each of the storage options. For Nugu App code analysis, the first thing we did was to figure out how the app creates the database and what security is implemented to protect data in the SQLite database.

From the source code, we were able to understand what database and tables are created. Figure 10 shows code snippets from the app used to create the SQLite database and its tables. The database name is _SSO.db, and it has three tables, TokenTank, LocalTokenTank, and packages. TokenTank is used to store the session id and tokens used for the app, and the LocalTokenTank has a copy of the TokenTank. On the other hand, the packages table used to store information about the app itself.

```

        private a g()
        {
            this.d = new a(this.e, "_SSO.db", null, 20160130);
            c = this.d.getWritableDatabase();
            return this;
        }

        public void onCreate(SQLiteDatabase paramSQLiteDatabase)
        {
            c.a("_SSO_MIG_", DB Create");
            paramSQLiteDatabase.execSQL("CREATE TABLE TokenTank (_ID INTEGER PRIMARY KEY);");
            paramSQLiteDatabase.execSQL("CREATE TABLE LocalTokenTank (_ID INTEGER PRIMARY KEY);");
            paramSQLiteDatabase.execSQL("CREATE TABLE Packages (_ID INTEGER PRIMARY KEY);");
        }

        try {
            String[] split = strArr[i3].split(":");
            ContentValues contentValues = new ContentValues();
            contentValues.put("userID", i2.a(split[0].trim());
            contentValues.put("sessionID", i2.a(split[1].trim());
            contentValues.put("token", i2.a(split[2].trim());
            contentValues.put("realYn", split[3].trim());
            contentValues.put("createDate", split[4]);
            contentValues.put("encryptedType", Integer.valueOf(1001));
            if (c.replace("TokenTank", null, contentValues) > -1) {
                i4++;
            }
        }
    }
}

```

Figure 10: Nugu App data creation code snippets

From the source code, we were able to understand what information the app saves in the database. Besides, to the static analysis on the source code, we also did dynamic analysis using tools such as Inspeckage. Figure 11: shows the Inspeckage analysis result.

```

6546 R/W Dir: /data/data/com.skt.aladdin/databases File: google_app_measurement_local.db
6545 R/W Dir: /data/data/com.skt.aladdin/databases File: _SSO.db
6544 R/W Dir: /data/data/com.skt.aladdin/databases File: _SSO.db
6543 R/W Dir: /data/data/com.skt.aladdin/databases File: _SSO.db
6542 R/W Dir: /data/data/com.skt.aladdin/databases File: _SSO.db
6541 R/W Dir: /data/data/com.skt.aladdin/databases File: http_auth.db
6540 R/W Dir: /data/data/com.skt.aladdin/databases File: http_auth.db
6539 R/W Dir: /data/data/com.skt.aladdin File: databases

```

Figure 11: Nugu app database dynamic analysis result from Inspeckage

From the above analysis, it is quite imperative to understand what information the app is saving and what kind of tables it creates. However, there are no data security considerations, except the database is created in the private storage, since the default MODE_PRIVATE attribute is applied.

In addition to the app analysis, we also did live forensics analysis to understand how the app stores the data in the databases. To perform, the analysis we used the sqlite3 command tool. After connecting the smartphone to the research computer, we get access to the device through the adb and shell commands. Once we navigated to the apps private storage /data/data/com.skt.aladdin/databases/, we used the sqlite3 tool to load the schema of the database and investigate the tables and their respective data. The following listings show the commands executed and the results as shown on the terminal.

```
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE android_metadata (locale TEXT);
INSERT INTO "android_metadata" VALUES('en_US');
CREATE TABLE TokenTank (_ID INTEGER PRIMARY KEY AUTOINCREMENT, userID
VARCHAR(200) NOT NULL UNIQUE, sessionID VARCHAR(100) NOT NULL, token
VARCHAR(300) NOT NULL, realYn VARCHAR(5) NOT NULL, encryptedType INTEGER
DEFAULT 0, createDate INTEGER);
INSERT INTO "TokenTank" VALUES(1,'9Q3Ah1d6SWU234J3IN3Ji1-
jF3sdfDkF13_3DmUvd2s','AuRo_afcAA9Iw6gRVd-XikvQmcG_2ZuWqKEy1-
IDVUvwQkTFn5WqMKKsyiGvAzYV','re-P0DRasGslT2pomZMxUC78E8pt353c-
PcFd9frah4AP2Z55RIjGrJYsMh3ylbiDqKG-
3phabAjJHpM9QHv0S2ub9Kl4rtkdeG5KpMIJ9hDZsL_y1nFT3VqYzITSIU7AXnltLzevKp
_toB9-BMbsw','Y',1001,1536752453000);
CREATE TABLE LocalTokenTank (_ID INTEGER PRIMARY KEY AUTOINCREMENT,
userID VARCHAR(200) NOT NULL UNIQUE, sessionID VARCHAR(100) NOT NULL,
token VARCHAR(300) NOT NULL, realYn VARCHAR(5) NOT NULL, encryptedType
INTEGER DEFAULT 0, createDate INTEGER);
```

Listing 6: dump command returns the current state of the tables in the database

The result from the live forensic analysis shows that the SSO database and the tables created have the current values for the information specified in the source code. More importantly, as the investigations show, there is no encryption applied to the data.

Next, to the database, the other data storage we investigated was shared preference. In the source code analysis, we learned that the app creates an XML file to save user login related information in a key-value format. Figure 12 is a code snip from the shared preference manager class.

```

public final class NuguPreferenceManager implements a {
    static final /* synthetic */ KProperty[] a = new KProperty[] { Reflection.mutableProperty1(new MutablePropertyReference1Imp
    public static final NuguPreferenceManager b;
    private static final SharedPreferences c;
    private static Map<String, SharedPreferences> d = new LinkedHashMap();
    private static final StringPreference e = new StringPreference("LastGetTIDLoginID", true, null, null, 12, null);
    private static final BooleanPreference f = new BooleanPreference("AutoLogin", true, false, null, 12, null);
    private static final StringPreference g = new StringPreference("LastGetToken", true, null, null, 12, null);
    private static final StringPreference h = new StringPreference("LastGetID", true, null, null, 12, null);
    private static final StringPreference i = new StringPreference("LastGetDeviceEncrypt", true, null, null, 12, null);
    private static final StringPreference j;

```

Figure 12: SKT Nugu shared preference manager code snippet

From the code, we can understand that login information is saved in the file. Next, we analyzed the same file using the dynamic analysis tool, Inspeckage to determine how the app writes the information. From the analysis, we understood that the app saves user login credentials such as TID, token and device ID as plain text in two files, com.skt.nugu.xml, and com.skt.nugu.some_number.xml.

```

448 GET[com.skt.nugu.824608292.xml] String(selectedDeviceId , ALDEPCKHWZ4E73D1F229)
447 GET[com.google.android.gms.analytics.prefs.xml] Long(monitored:count , 1)
446 GET[com.google.android.gms.analytics.prefs.xml] Long(monitored:start , 1548861261425)
445 GET[com.skt.nugu.xml] String(LastGetTIDLoginID , .....~1)
444 GET[com.skt.nugu.xml] String(LastGetID , ALDEZM1ZB0L33EF872B5)
443 GET[com.skt.nugu.xml] String(LastGetToken , A320E598E20B418786E74797FF0A934B)

```

Figure 13: SKT Nugu shared preference dynamic analysis Inspeckage output

Moreover, after analyzing the app using reverse engineering and live forensics analysis, we extracted the files from the smartphone and analyzed them offline using text editors. The first file com.skt.nugu.xml saves most of the login information while the second file com.skt.nugu.some_number.xml saves the device type and device ID. The only value that was encrypted in the com.skt.nugu.xml file is LastGetdeviceEncrypt. Figure 14 shows one of the files using text editors. During the forensic analysis, we found that there is another XML file in the shared preference called com.sktelecom.pref.xml. After opening the file, we learned that the values are encrypted. However, from the keys, it is easy to determine what information is being saved in the file.

```

<string name="LastGetToken">193E382F7AFF43D18DA60762A6E50D61</string>

<string name="LastGetID">ALDEZM1ZB0L33EF872B5</string>

<string name="LastGetDeviceEncrypt">dk39jwoA300Csld0</string>

<boolean name="AutoLogin" value="false" />

<string name="LastGetTIDLoginID">[REDACTED]</string>

```

Figure 14: com.skt.nugu.xml file

The other data sources in the app storage were internal and external memory storage. From the code analysis, we learned that the app saves data to the storage without any security features. Analyzing the AndroidManifest file indicates that the app writes to external storage. Moreover, using the MobSF tool, we learned that the app saves user data in external storage without any security considerations. From the forensic analysis, we learned that the app saves cache data in the internal storage of the app. On the other hand, during our analysis, we could not find user data saved on the external storage under the app's name. Figure 15 shows the security analysis report from the MobSF tool.

| | | | |
|---|-----------|-------------------------------------|--|
| android.permission.WRITE_EXTERNAL_STORAGE | dangerous | read/modify/delete SD card contents | Allows an application to write to the SD card. |
| android.permission.READ_EXTERNAL_STORAGE | dangerous | read SD card contents | Allows an application to read from SD Card. |

Figure 15: Permissions in the SKT Nugu Androidmanifest file

4.1.1.2. Nugu App Network Investigation

The other security point of view we analyzed the app was, the implemented network security between the app and the SKT cloud. To perform the investigation, we used two approaches specified in the research methodology section. The first approach was capturing the network traffic using the Wireshark analyzer. From the analyzer, we understood that the app uses TLSv1.0 to encrypt the communication channel.

| | | | |
|-----------------|-----------------|-------|----------------------|
| 192.168.137.29 | api.sktnugu.com | TLSv1 | 491 Application Data |
| api.sktnugu.com | 192.168.137.29 | TLSv1 | 936 Application Data |
| 192.168.137.29 | nog.sktnugu.com | TLSv1 | 518 Client Hello |
| nog.sktnugu.com | 192.168.137.29 | TLSv1 | 3014 Server Hello |

Figure 16: Communication security between SKT Nugu App and SKT cloud

The second approach was utilizing the MITM attack. First, from the source code analysis using the MobSF tool, we learned that the app is vulnerable to MITM attack due error in handling third-party certificates. Figure 17 shows a code snippet from MobSF analysis tools. Then, to conduct the attack, we used the Burp Suite proxy tool. From the attack, we managed to intercept user-related information such as tokens, TID and device ID. More importantly, we managed to extract the APIs used to request user data from the cloud.

```

public void onReceivedSslError(WebView webView, SslErrorHandler sslErrorHandler, SslError sslError) {
    if (webView instanceof SSOWebView) {
        webView = (SSOWebView)webView;
        if (webView.c != null) {
            c.a("remove handler runnable");
            webView.c.removeCallbacks(webView.d);
        }
        if (webView.a != null) {
            webView.a.setVisibility(4);
        }
        if (d.c) {
            sslErrorHandler.proceed();
            return;
        }
        webView.b.a(d.a.f, null);
        webView.b = null;
        sslErrorHandler.cancel();
        return;
    }
    super.onReceivedSslError(webView, sslErrorHandler, sslError);
}

```

Figure 17: SKT Nugu app sslError handling method code snippet

From the network analysis, we learned that during the login process, the app detects the proxy setting using the certificate information. Once the login process is passed without the proxy setting, the subsequent requests can be captured using the MITM attack.

Using this method, we were able to capture the APIs requests and the access token along with other user and smartphone device information. Figure 18 depicts the information captured using the MITM. The user's ID and TID were used to identify the user during the data requests using the APIs. For example, APIs such as device information and alarm look like the following

<https://api.sktnugu.com/v1/setting/devices/v2>

and

https://api.sktnugu.com/v1/setting/menu/services/serviceTypeCodes/SVC_ALARM. With the

auth-token and other user information, requests can be done to fetch user data from the cloud.

| Name | Value |
|--------------------------------|----------------------------------|
| GET | /v1/setting/common/app HTTP/1.1 |
| Content-Type | application/json; charset=utf-8 |
| Auth-Token | A320E598E20B418786E74797FF0A934B |
| User-Id | ALDEZM1ZB0L33EF872B5 |
| Os-Type-Code | MBL_AND |
| Os-Version | 4.4.2 |
| App-Version | 2.3.0 |
| Target-Device-Id | ALDEPCKHWZ4E73D1F229 |
| Target-Device-User-External-Id | |
| Target-Device-Type-Code | DVC_SPK |
| T-ID | [REDACTED] |
| Application-Type | NUGU_APP |
| Phone-Model-Name | SHV-E250L |
| appPhase | prd |
| Host | api.sktnugu.com |
| Connection | close |
| Accept-Encoding | gzip, deflate |
| User-Agent | okhttp/3.10.0 |

Figure 18: Intercepted Nugu Android App information

Auth-Token=2BFB91EC08D046199385C7EC290546CC

In SKT Nugu, for some of the APIs which are used to access another service, additional identification tokens are also used. For instance, for Melon Music Service request, Melon ID is required to access user data. Therefore, the user has to register independently for the service and use that ID to access the service. Later, that ID is included in the API request to access the service and acquire user data from the Melon service cloud.

4.1.2. Nugu AI Speaker Device Network Investigation

In this research, we investigated the implemented communication security method in SKT Nugu AI Speaker by intercepting the traffic between the IoT devices and the backend cloud using the Wireshark traffic analyzer. Similar to the App, SKT Nugu uses TLSv1.0 between the speaker device and the SKT cloud. Moreover, it pushes firmware updates to the device as a plaintext. A checksum is the only security applied to verify the integrity of the firmware. From the captured firmware, it is possible to get the update package information including the version. This kind

of vulnerability may lead to control the device by replacing the firmware with backdoored firmware. Moreover, one of getting the device's firmware is by intercepting the update from the network, during the update process called the sniffing Over-The-Air (OTA) process [67]. [68] demonstrated Xiaomi smart home gateway hack using the unencrypted firmware update process.

4.1.3. Results Analysis

From our investigation of SKT Nugu AI Speaker and the companion security, we learned that there are some security vulnerabilities which can be used for the benefit of a digital forensic investigation. From the apps, lack of data storage security enabled us to access the tokens, TIDs and device IDS that can be used to login to the cloud and acquire user data through the APIs extracted using the MITM attack. Specifically, the analysis of the shared preference files revealed all the login credentials for the API authorization to acquire user data from SKT cloud. However, the data on the SQLite database was not valuable in our data acquisition process. Similarly, from the vulnerability in handling SSL certificate error, we were able to intercept the login credentials and API requests to the cloud. More importantly, intercepting the traffic using the MITM attack, enabled us to extract the APIs and understand the request parameters and header information for the APIs. Using those API, we developed a cloud data acquisition tool to acquire user data from SKT cloud automatically. On the other hand, the network investigation using the Wireshark tool, revealed the communication security used, which is TLSv1.0. At the time of writing this thesis research, TLSv1.0 was at the verge of obsolescence due to the vulnerability in the protocol design. During this research, although we did not try to attack the device using the vulnerability in the device firmware updating process, researchers in Northern University College of Computer and Information Science demonstrated that through firmware update vulnerability they were able to take over Xiaomi smart home device by replacing the firmware by modified firmware [68].

4.2. Clova AI speaker

Naver Clova is Artificial Intelligence (AI) enabled voice assistance from the Korean search engine company Naver in collaboration with a company called Line. Naver Clova comes in two kinds Brown, and Yellow colored with wake words “Sally” and “Clova” respectively. The supported functionalities include Music streaming, Weather, Calendar, Alarm, Reminder, navigation, and many online shopping options. It has a Naver Clova companion App to manage and access user’s cloud data. The app also supports voice commands from the smartphone [69].



Figure 19: Naver Clova AI speaker operation mode

Since Naver did not provide a Web interface for device management, the Naver Clova app is the only available application used to set up the device and access user data collected by the device. At the time of conducting this research, the latest version is 2.13.0.

To set up the speaker, Naver account information required. Once the account is created, using the app users can register the device and start configuring it, basically setting the wifi network for the speaker. After the setup is complete, using the wake words, voice commands can be issued to the device. Then the app is used to access the ordered services. In addition to accessing data from the cloud, the app can also be used to issue commands such as setting alarms, reminders and other functionalities using voice commands in the same way to the speaker.

As stated above, the Naver Clova App is used to access user sensitive user information and other ordered services. As a result, these user data should be protected using the recommended security techniques.

In the following section, we presented the security analysis of the app using the stated methodologies. As defined in the scope of the research, the analysis focuses on the security techniques implemented to protect user data while at rest and in transit. Besides, the study also addresses the communication security implemented between the speaker and Naver cloud.

4.2.1. Naver Clova App Analysis

Based on the designed research methodology, we approached the analysis from three different sectors. The first part addresses the security techniques implemented in the app to protect user data saved on the smartphone storage. To achieve this, we conducted live forensics on the app installed on Samsung Galaxy Note 2 with Android version 4.4 (KitKat). The smartphone is rooted for the research purpose, in the same way, we used for Nugu App.

Moreover, we performed static and dynamic analysis on the app using Android reverse engineering approach. Finally, we analyzed the app from network investigation perspectives using traffic analysis through Wireshark and Man-in-the-Middle attack approach. In the following subsection, we presented detailed activities and findings for the study.

4.2.1.1. Naver Clova App Data Storage Analysis

For our analysis, the first activity we performed on the app was to perform static code analysis after extracting the app using adb pull from the smartphone. After static analysis, we immediately followed the dynamic analysis to determine how the files are created. Finally, at the third stage, we used live forensics approaches using a combination of the Linux commands and adb operations on the smartphone. For database analysis, we also used the sqlite3 commands. For some of the files, we also exported them to the local computer using the adb pull operation and used tools such as SQLite DB browser and Notepad ++ editor.

Using the manual decompiling process, we extracted the dex files from the apk file and converted them to jar files to analyze using java decompiling tools. Naver Clova app has three dex files. These are the compiled java source code files. Now converting these files to jar files is

required to open and analyze the class files as a java source code. To convert DEX files to jar file we used the open source dex2jar converter tool.

In addition to the manual decompiling process, we also used an automated tool called Mobile Security Framework (MobSF). The advantage of this automated tool is, the capability to parse and report security issues in the apps after performing the decompiling process. Figure 20 shows the app information provided by MobSF.

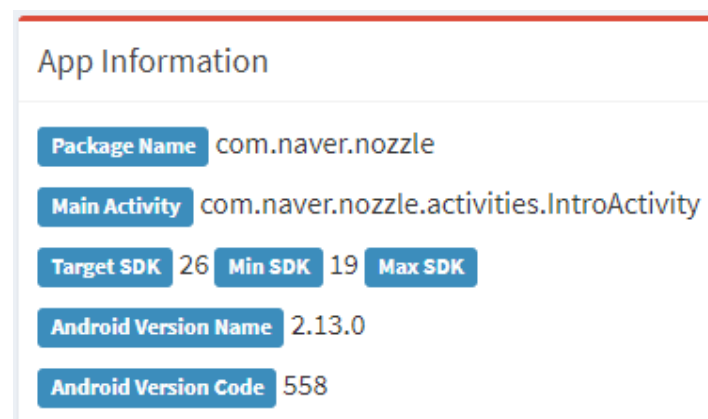


Figure 20: Naver Clova App information after decompiled using MobSF tool showing detail information about the app including the version

Similar to the Nugu App case, since the goal is to determine how the apps save files and what security techniques are used, we analyzed the source code related to SQLite database, shared preference, internal and external storage operations.

The following section presents the findings in the Clova apk static analysis for each of the storage options. For Nugu App code analysis, the first thing we did was to figure out how the app creates the database and what security is implemented to protect user data in the SQLite databases.

From the source code, we were able to understand what database and tables Naver Clova app creates. Figure 21 shows the code snippet from the app used to create the SQLite database and its tables. The database name is line_preference.db and two tables named notification_pref and notification_image. From the analysis, we learned that notification_pref is used to store

notification messages in JSON format while the notification_image table saves image files probably used in the notification.

```
public NotificationPrefDBHelper(Context paramContext)
{
    super(paramContext, "linenotice_pref.db", null, 2);
}

private void createTable(SQLiteDatabase paramSQLiteDatabase)
{
    paramSQLiteDatabase.execSQL("CREATE TABLE notification_pref (_id INTEGER PRIMARY KEY, noti_id LONG, noti_json_data STRING, noti_read_timestamp LONG);");
    paramSQLiteDatabase.execSQL("CREATE TABLE notification_image (image_url STRING PRIMARY KEY, noti_id LONG, image_data BLOB);");
}

private void dropTable(SQLiteDatabase paramSQLiteDatabase)
{
    paramSQLiteDatabase.execSQL("DROP TABLE IF EXISTS notification_pref");
    paramSQLiteDatabase.execSQL("DROP TABLE IF EXISTS notification_image");
}
```

Figure 21: Naver Clova app database creation code snippet

As stated above, from the source code, we were able to understand what information the app saves in the database. Besides, to the static analysis on the source code, we also tried to do dynamic analysis using Inspeckage. However, during this research, the app keeps crashing during our trial on our research phone, which was rooted and had dynamic analysis tools such as Xposed. Later during our network investigation, we learned that it sends a crash report as a plain text. From the crash result, we understood that the app has the capability to detect rooted phones.

From the above analysis, it is quite imperative to understand what information the app is saving and what kind of tables it creates. However, there are no data security considerations, except the database is created in the private storage and the default MODE_PRIVATE attribute is applied. To verify this, we used the MobSF tool to analyze the applied securities. However, the tool complained that the app does not use encryption methods to secure the database.

In addition to the static analysis, we also did live forensics analysis to understand how the app stores the data in the databases. To perform, the analysis we used the sqlite3 command tool. After connecting the smartphone to the research computer, we get access to the device through the adb shell command. Once we navigated to the apps private storage /data/data/com.naver.nozzle/databases/, we used the sqlite3 tool to load the schema of the

database and investigate the tables and their respective data. The following listings show the commands executed and the results as shown on the terminal.

```

root@t0ltelgt:/data/data/com.naver.nozzle/databases # sqlite3 linenotice_pref.db
SQLite version 3.7.6.3-Titanium
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE android_metadata (locale TEXT);
INSERT INTO "android_metadata" VALUES('en_US');
CREATE TABLE notification_pref (_id INTEGER PRIMARY KEY, noti_id
LONG, noti_json_data STRING, noti_read_timestamp LONG);
INSERT INTO notification_pref
VALUES(1,1115043,{ "body":"","weight":0,"linkUrl":"","interval":0,"bannerBtn2Text":"","r
epeat":true,"type":"page","btnType":0,"close":1548946740000,"id":1115043,"open":1546268
400000,"revision":1437772,"title":"유인나 2 차
(리얼 반영)","startupOnly":true,"marketAppLink":"","bannerBtnType":0,"bannerBtn1Url":"","
,"bannerBtn2Url":"","countOnType":"","status":"OPENED","contentUrl":"https://lanimg-
beta.line-
apps.com/lan/zipstaging/pageEvent/Clova/android/ko/popup_20181224_Android_3Ufe3
929556886287270393/popup_20181224_Android/popup_20181224.html","format":1,"bann
erDescription":"","bannerTitle":"","view":"","bannerBtn1Text":"","immediately":false}',0);

```

Listing 7: Live Analysis of linenotice_pref.db file using the sqlite3 command

The result from the live forensic analysis shows that the line_preference database and the tables created have the current values for the information specified in the source code. More importantly, as the investigations show, there are no applied encryptions to the data.

We also conducted an offline analysis of the database. After extracting the linenotice_pref.db file using the adb pull operation, we analyzed it using the open source DB Browser for SQLite tool. Figure 23: shows the result with the current data on the table. During this research, we could not find image files saved in the table as specified in the source code. This might be due to notifications are without picture files.

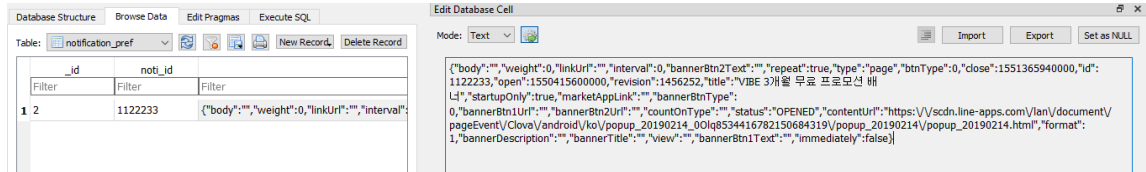


Figure 22: Naver Clova app linenotice_pref.db file

Next, to the database, the other data source we investigated was shared preference. In the source code analysis, we learned that the app creates multiple XML files to save user and app related information in a key-value format. Among them, we suspected that two of them (clova.xml and clovatoken.xml) are used to save user data required by the app. Figure 23 and 24: shows a code snippet from the shared preference manager class in the app. From the codes, we can see that the XML files are used for storing token values along with the client ID and Client Secret. Note, client ID and Client secret is used to authorize and authenticate the app by Naver cloud server.

```
static String m493c(@NonNull ClovaEnvironment clovaEnvironment, @NonNull CicNetworkInfo cicNetworkInfo, @Non
    Builder buildUpon = Uri.parse(clovaEnvironment.getValue(Key.authHostUri)).buildUpon();
    buildUpon.appendPath("token");
    buildUpon.appendQueryParameter("grant_type", "refresh_token");
    buildUpon.appendQueryParameter("refresh_token", str);
    buildUpon.appendQueryParameter(ParamConst.PARAM_MODEL_ID, clovaEnvironment.getValue(Key.modelId));
    buildUpon.appendQueryParameter(ParamConst.PARAM_DEVICE_ID, clovaEnvironment.getValue(Key.deviceId));
    buildUpon.appendQueryParameter("client_id", clovaEnvironment.getValue(Key.clientId));
    buildUpon.appendQueryParameter("client_secret", clovaEnvironment.getValue(Key.clientSecret));
    return buildUpon.build().toString();
}
```

Figure 23: Naver Clova app shared preference file editor code snippet

More importantly, from the code analysis, we identified that the Naver Clova app uses a security module to protect data in the shared preference files. The code snippet in figure 24 shows the SecuredSharedPreferences class with three different cipher algorithms used for XML files security.

```

public class SecuredSharedPreferences {
    /* renamed from: a */
    private final Cipher f14399a = Cipher.getInstance("AES/CBC/PKCS5Padding");
    /* renamed from: b */
    private final Cipher f14400b = Cipher.getInstance("AES/CBC/PKCS5Padding");
    /* renamed from: c */
    private final Cipher f14401c = Cipher.getInstance("AES/ECB/PKCS5Padding");
    /* renamed from: d */
    private final SharedPreferences f14402d;
}

```

Figure 24: Naver Clova app secured shared preference code snippet

Next, we analyzed the same file using the Inspeckage dynamic analysis tool to determine how the app writes user information in the XML files. However, as stated in database analysis, since the app keeps crashing when we try to run it within the dynamic analysis environment, we could not get the dynamic analysis results. On the other hand, using the MobSF tool, we were able to analyze the app dynamically. From the tool, we determined that the following XML files were accessed during the app's operation.

Moreover, after analyzing the app using reverse engineering, we extracted the files from the smartphone and analyzed them offline using text editors. The first file clova.xml saves most of the user and device information while the second file clovatoken.xml stores the tokens used for the app. Figure 25 and 26 show the results using text editors. Besides, NaverOAuthLoginPreferenceData.xml file has Naver app login information, including client id and client secret in an encrypted format. After opening the file, we learned that the keys and values in clova.xml are encrypted as specified in the source code. As a result, it is difficult to determine what most of the information saved in the file. However, since the keys are not encrypted for some part of the file, we were able to determine the information saved in the clova.xml file.

For the clovatoken.xml file saves token information used by Clova. In the file, one of the keys specifies the token as a Bearer token; however, the actual value of the token is not the same as the one captured using network analysis.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="68nbBmKz35WriNoyDOfw==">KL+cBH2CTscUSLibAzKvZj9YVh40TrBf8+14E/mra1ih+NtL4KvZYHnXTx2zHzZCH2PegykcH+f0xiWf1hfMKXR3NM0Q413KsPz
halVyk38862wJKVg4g00oJiAZ2bdpCDIPKekP9nm0pTvQCvYAA==</string>
  <string name="o73ZQbxKzbmTST2K+zkJAw==">DXryqdn4XdhYi2NkNcHjBw==</string>
  <string name="gnXsesVosSR2eFlkVCTKuQ==">mL1ktrh37duP7AwoK3tPet+XGNKMXANB1Fi64D1BkAqmJiulTXGS2MF7h+AaQX1Wcmtqbb0xFeqgtV7QPpfmrE9wPF9qGCRUinbc
7WMDub5BHiVgpQIEbH1X1h1qpbpZfX0AqWRMhyFbkQ+T5Wx256T++PxtXb1IQXCgU0TYPsc7E9+EZJYY3vascqCVB06sa/VVRHkPPzPBmXXDn5E4pUw77rjAkprmlKJqG1euTkTBDrne1EUQ
Bx+/F5hbFgPuAuKdKSDDi3Txb8mIrf+AKHcJut4dYEvtUjcd8uJRPFXN20aBrZEIM8wpDqs4HGE6x3OMJ/gr/4cj4NBH5MdqjxyfLx6j17cAyWbzYgLCUaic+C/1Jg35FuCPstWQ3gpnTPFu
y+f7y90WL3A86LEOM5BstgPF+UIGcYVvRPBHVZZJFd110xBEkft8Z68RKLNLnQWviANx1/X3Kc36nGgeZkc+erBa1WAKBQwskmTABNHTtRY1eouTw7yjayRQP/Z7iAFG0tFaaDcVyClXV0BC
iE2NzptaJEp6TihG5a3wrRo852toom00TzJ9io8YjVM4Lxma4Ut4PqL8hAyK9Y/ZSUs9UC842C+99t5FuFb+V8rFszek57gS18MEAj578KA5IQ35pf4DiInitZ204L09o3SBRUE53G5P4XF
vcTT4RN3upc15oH9erzigXIOgvJQxcOgC0MbieLk2YDi22QPjRLsK0718oK8+0feDqYil6MQ0jW4Twr5CKfjG6qneS8WCQ8LyhEdzkaS04/T9usrJhAo5V1NDYlnovHzz8Vw+IalnAJede9vG
Jj+H5kQW917seLYFP8voenmk/Cf1F2sZbJSRSnq6Rhr1qP15YD9z0FayqEQqHyD/sB6V1FTTQ7i/W+9gpkY0AoeNYcHfn8t/8LBzIXibjKvAQ4y+oBe0uRiW6Gx3kvNnfUS+FExxCMdmtaBd
hfEpuYqoupg==</string>
  <string name="80o3SeDj/f1FHp55mC+zxA==">J33Z6GNqMaak1w3h4Dkj+Q==</string>
  <string name="/A2AUv3Q1q+077KCJCz65Ngc15US/NaIONM02JedxCk=">osrSnbOJ0t50nPQwB4oSEQ==</string>
  <string name="2hTnay/Uy21QDIQdpKLNg==">gfCp5fvdBcbsG2VImgGQUw==</string>
  <boolean name="is_show_main_activity_badge" value="true" />
  <boolean name="last_input_mode" value="false" />
  <boolean name="is_set_user_id_to_nelo" value="true" />
  <int name="lastShownBadgeCount" value="3" />
  <boolean name="FCM_token_updated" value="false" />
  <boolean name="adult_agreement" value="false" />
  <string name="8wN9sITodoLX5aybamPnG04ygnIghoVdN1fdaivJ4KjyyXC/Nh2G0zOq6D2K5iuis9rbpZwx0TvWjgdkSQJcNugnEjIpJnHBba3hLg/4==">i7ivjCw/2IiuLGib/
NaSwg==</string>
  <boolean name="is_newbie" value="false" />
  <string name="gQ056P1/10LKYQH4sFhJLw==">uxx16RxDWJ1zWUyeOA0aLvtOH465B3G4RaYBX3bWxNQediTMLpLYj0YuPEeCa81bv510jkLo9uTm2eDfbUPHGxNGhibeRBqHZMjF
YjW80FAKSwrjUDHbvbKjShD/hEFvmp8oN6UmbjWvU5iWtP/LqX+QG3rNPps220XqYmQJeJatQdDbWgegIXftspOee3sKdmdTS6VZrjwath72ikQ==</string>
  <boolean name="is_show_user_setting_activity_badge" value="true" />
</map>
```

Figure 25: Naver Clova clova.xml file with partially encrypted keys and values

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="tokenType">Bearer</string>
  <null name="accessToken" />
  <string name="expiresIn">12960000</string>
  <string name="refreshTokenV2">VApnMiEDxgOzPel1AXQxlmOjzt2zFsXSobctXRf0W+WFT23JPgIkih/FosoB3jNiICZwjbQ+T5jI
HgybZU34sXuEwM+Dz09010dk+fUpGj4=
</string>
  <string name="accessTokenV2">KMHW1dbso4cOGgwZHOgg2UFTDSyiqtyLqKoxS/ZZ+lE+ZSJ8ue8VSjX7Fpv/U7EqNs1TXZM0vb51
3vM59WMTA28BK2EWQb+GQPTMD1riM8s=
</string>
  <null name="refreshToken" />
  <string name="expiredAt">1551520786689</string>
</map>
```

Figure 26: Naver Clova app clovatoken.xml file with current token values

In addition to the XML files, from the app analysis, we also found the base of the APIs in the source code. Figure 27 shows the base of the APIs and the URL to the authentication server. Later we extracted all the APIs using the Man-in-the-Middle attack, see the Naver Clova network investigation between the app and the cloud section.


```

localic.g = PreferenceManager.getDefaultSharedPreferences((Context)this).getString("hostUrl");
final String[] arrayOfString1 = new String[7];
arrayOfString1[0] = "https://dev-cic.clova.ai/";
arrayOfString1[1] = "https://beta-cic.clova.ai/";
arrayOfString1[2] = "https://dev-pub-cic.clova.ai/";
arrayOfString1[3] = "https://qa-pub-cic.clova.ai/";
arrayOfString1[4] = "https://prestige-cic.clova.ai/";
arrayOfString1[5] = "https://stage-cic.clova.ai/";
arrayOfString1[6] = "https://prod-ni-cic.clova.ai/";
final String[] arrayOfString2 = new String[8];

```

Figure 27: Naver Clova APIs base URL in the extracted apk file

The other data source we analyzed were, the internal and external memory. From the code analysis, we learned that the app saves data without any security features applied. Analyzing the AndroidManifest file indicates that the app writes to external storage. Moreover, using the MobSF tool, we learned that the app saves user data without any security considerations. From the forensic analysis, we learned that the app saves cache data in the internal storage of the app. Similar to the Nugu app case, during our analysis, we did not find user data saved on the external storage under the app's name. Figure 28 shows the security analysis report from the MobSF tool.

| | | | |
|---|-----------|-------------------------------------|--|
| android.permission.READ_EXTERNAL_STORAGE | dangerous | read SD card contents | Allows an application to read from SD Card. |
| android.permission.WRITE_EXTERNAL_STORAGE | dangerous | read/modify/delete SD card contents | Allows an application to write to the SD card. |

Figure 28: Naver Clova app permissions in the Android manifest file

4.2.1.2. Clova App Network Investigation

The other security point of view we analyzed for the Clova app was the implemented network security between the app and the Naver cloud. To perform the investigation, we used two approaches specified in the above research methodology section. The first approach was capturing the network traffic using the Wireshark analyzer. From the analysis, we understood that the app uses TLSv1.2 to encrypt the communication channel.

| | | | |
|----------------------|----------------------|---------|----------------------|
| 192.168.137.148 | prod-ni-cic.clova.ai | TLSv1.2 | 327 Application Data |
| 192.168.137.148 | prod-ni-cic.clova.ai | TLSv1.2 | 119 Application Data |
| prod-ni-cic.clova.ai | 192.168.137.148 | TLSv1.2 | 256 Application Data |
| prod-ni-cic.clova.ai | 192.168.137.148 | TLSv1.2 | 224 Application Data |

Figure 29: Communication security between Naver Clova App and Naver cloud

On the other hand, during our network investigation using Wireshark, we learned that Naver Clova sends crash analytics data without any security. The data includes the main class, the root status of the phone, the network state and the IP address assigned to the phone.

The second approach to the network investigation was using the MITM attack. From the source code analysis using the manual and automated analysis, we could not learn how the app handles third-party certificates, in different web activities except for the Facebook web activity call. For the Facebook web activity call, in case of `sslError`, the app is forced to cancel requests using the `sslError.cancel()` function.

Even though we could not get hints from reverse engineering of the app, we carried out a MITM attack to the app. To conduct the attack, we used the free version of Burp Suite proxy tool. From the attack, we learned that during the login process, Naver Clova detects proxy settings. Therefore, we have to first login without proxy settings and try the attack. Then, we figured out that, once the login process is passed, the app does not check for third-party certificates. As a result, subsequent requests were captured using the MITM attack. Using this method, we were able to extract the communication APIs and the bearer token (shown in figure 30) used for authorization of the API requests. For example, the API to acquire alarm information looks <https://prod-ni-cic.clova.ai/internal/v1/api-gw/alerts/>. Using this API with the authorization token, we were able to obtain user data from the cloud.

```
User-Agent: ClovaApp/Android/2.13.0 (Android 4.4.  
Authorization: Bearer VcaRU596QZe7qxmaCP8rfw  
Content-Type: application/json; charset=UTF-8  
Content-Length: 135  
Host: prod-ni-cic.clova.ai  
Connection: close  
Accept-Encoding: gzip, deflate
```

Figure 30: Naver Clova Bearer Token intercepted using MITM attack

Authorization = Bearer VcaRU596QZe7qxmaCP8rfw

We also learned that the API could be used to acquire alarm, reminder and alert information from the cloud by setting the requested message type in the body of the API request as shown below. In all requests, the same bearer token is used for authorization of the APIs.

```
{ "method": "GET", "path": "/api/pims/alerts/setting-  
list/", "query": { "type": "ALARM", "deviceIdList": "422063af-8ea2-3e79-a84a-  
a5cd50143fa4" } }
```

During our research, apart from the network interception using the MITM attack, we could not find the bearer token anywhere else. In the app analysis, in the shared_preference folder clovatoken.xml file saves access token that says bearer token with no value and a couple of refresh tokens, however, using those tokens to request data returned Unauthorized Error (401). Moreover, the length of the bearer token captured in the network and the one that was found in the clovatoken.xml file is different. The bearer token captured from the network is shorter than bearer token saved in the file.

4.2.2. Clova AI Speaker Device Network Investigation

In this research, we investigated the implemented communication security method in Naver Clova AI Speaker by intercepting the traffic between the device and the backend cloud using the Wireshark traffic analyzer. Similar to the App, Naver Clova uses TLSv1.2 between the speaker device and the Naver cloud.

| | | | |
|-------------------------|-------------------------|---------|--|
| 192.168.137.232 | nelo2-col.navercorp.com | TLSv1.2 | 583 Client Hello |
| nelo2-col.navercorp.com | 192.168.137.232 | TCP | 118 443 → 50542 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=1050015813 TSecr=172320 |
| nelo2-col.navercorp.com | 192.168.137.232 | TLSv1.2 | 3014 Server Hello |
| nelo2-col.navercorp.com | 192.168.137.232 | TCP | 3014 443 → 50542 [ACK] Seq=1449 Ack=518 Win=30080 Len=1448 TSval=1050015815 TSecr=172320 [TC |
| nelo2-col.navercorp.com | 192.168.137.232 | TLSv1.2 | 954 Certificate, Server Key Exchange, Server Hello Done |
| 192.168.137.232 | nelo2-col.navercorp.com | TCP | 66 50542 → 443 [ACK] Seq=518 Ack=1449 Win=90624 Len=0 TSval=172322 TSecr=1050015815 |
| 192.168.137.232 | nelo2-col.navercorp.com | TCP | 66 50542 → 443 [ACK] Seq=518 Ack=2897 Win=93440 Len=0 TSval=172322 TSecr=1050015815 |
| 192.168.137.232 | nelo2-col.navercorp.com | TCP | 66 50542 → 443 [ACK] Seq=518 Ack=3315 Win=96512 Len=0 TSval=172322 TSecr=1050015815 |
| 192.168.137.232 | clova-ota-auth.clova.ai | TCP | 66 49574 → 443 [ACK] Seq=851 Ack=7140 Win=105216 Len=0 TSval=172324 TSecr=3022793064 |
| prod-ni-cic.clova.ai | 192.168.137.232 | TCP | 118 443 → 46084 [ACK] Seq=11646 Ack=61977 Win=1432 Len=0 TSval=1310644257 TSecr=172318 |
| 192.168.137.232 | nelo2-col.navercorp.com | TLSv1.2 | 192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message |
| nelo2-col.navercorp.com | 192.168.137.232 | TLSv1.2 | 666 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message |
| 192.168.137.232 | nelo2-col.navercorp.com | TLSv1.2 | 1170 Application Data, Application Data |
| nelo2-col.navercorp.com | 192.168.137.232 | TLSv1.2 | 1114 Application Data |
| 192.168.137.232 | nelo2-col.navercorp.com | TLSv1.2 | 97 Encrypted Alert |
| nelo2-col.navercorp.com | 192.168.137.232 | TCP | 118 443 → 50542 [FIN, ACK] Seq=4887 Ack=1779 Win=32256 Len=0 TSval=1050015896 TSecr=172328 |

Figure 31: Encrypted network traffic between Naver Clova AI speaker device and cloud

4.2.3. Results Analysis

From our investigation of Naver Clova AI Speaker and the companion security, we learned that there are some security vulnerabilities which can be used for the benefit of a digital forensic investigation. From the apps, lack of data storage security enabled us to access the data from the database, which can be considered as user history log. The analysis of the shared preference files revealed that the app uses encryption to secure information saved on the clova.xml file. However, from one of the clovatoken.xml files, we were able to identify token information, but the tokens were not useful to acquire user data from the cloud. Similarly, from the vulnerability in handling SSL certificate error, we were able to intercept the bearer token and API requests to the cloud. More importantly, intercepting the traffic using the MITM, enabled us to extract the APIs used to acquire user data from the Naver cloud. Using those API, we developed a cloud data acquisition tool to acquire user data from the cloud automatically. On the other hand, the network investigation on the app using the Wireshark tool revealed the communication security used is TLSv1.2. However, we also managed to capture the crash reports in plain texts with valuable information, such as user's device status information and network information. Finally, the device also uses TLSv1.2 to encrypt traffic to the cloud.

4.3. Xiaomi Smart Home Kit

Xiaomi Smart Home Kit is a product of Xiaomi Company based in China. The smart home kit contains one Home automation unit (Gateway - called Lumi gateway), smart wireless switch (to control connected devices), door or window sensor (sense door or windows opening and closing activities) and smart plug (automate power operation for non-IoT devices). The gateway is the central hub that connects the sensors using ZigBee protocol and Wireless Internet to the Xiaomi cloud. Mi Home smartphone app is used to configure and monitor the Kit [70].

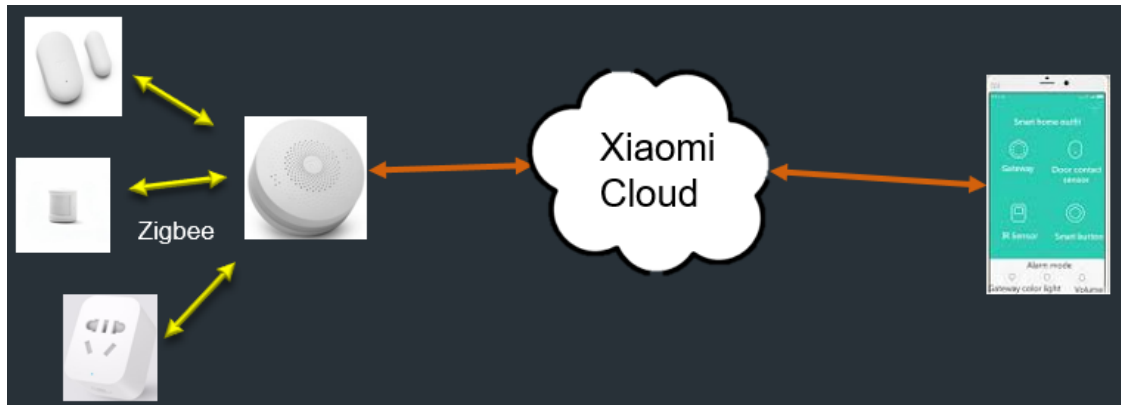


Figure 32: Xiaomi Smart Home Kit operation mode

Xiaomi did not provide a Web interface for the device and user data management; the Mi Home app is the only available application used to set up the device and access user data collected by the device. At the time of conducting this research, the latest version is 5.5.10; however, to run the app on Android KitKat, we used 5.1.30.

To set up the Xiaomi gateway device and the sensors, account information required. Once the account is created, users can register the device and start configuring it using the app, basically setting WIFI network for the gateway, identification and adding sensors to the gateway. Then the app is used to manage the devices and access log data from the sensors through the cloud.

As stated above, Mi Home App is used to access user sensitive user information and manage the devices. As a result, these user data and the devices should be developed with security techniques to protect attack on user data and the devices.

In the following section, we presented the security analysis of the app using the stated methodologies. As defined in the scope of the research, the analysis focuses on the security techniques implemented to protect user data while at rest and in transit. Apart from the app, the research also addresses the communication security implemented between the gateway and the Xiaomi cloud.

4.3.1. Mi Home App Analysis

Based on the designed research methodology, we approached the analysis from three different sectors as we did for the AI Speakers. The first part addresses the security techniques implemented in the app to protect user data saved on the smartphone storage. First, we performed static and dynamic analysis on the app using Android reverse engineering approach. Then, we conducted live forensics on the app installed on Samsung Galaxy Note 2 with Android version 4.4 (KitKat). The smartphone is rooted for the research purpose, in the same way, we used for AI Speakers Apps. Finally, we analyzed the app from network investigation perspectives using traffic analysis through Wireshark and Man-in-the-Middle attack approach. In the following section, detailed activities and findings of the analysis are presented.

4.3.1.1. Mi Home App Data Storage Analysis

For our analysis, the first activity we performed on the app was to perform static code analysis after extracting the app using adb pull from the smartphone. After static analysis, we immediately followed the dynamic analysis to determine how the files are accessed. Finally, at the third stage, we used live forensics approaches using a combination of the Linux commands and adb operations on the smartphone. For database analysis, we also used the sqlite3 commands. Also, for some of the files, we exported them to the local computer using the adb pull operation and used tools such as SQLite DB browser and Notepad ++ editor.

First using the manual app reversing process, we extracted the dex files from the apk file and converted them to jar files to analyze using java decompiling tools. After, reversing we found that, the app has lots of native and third-party libraries integrated the app. Moreover, there are seven dex files. From Figure 33, we can see that there are seven DEX files. These are the compiled java source code files. Now converting these files to jar files is required to open and analyze the class files as a java source code. To convert DEX files to jar file, we used the open source dex2jar converter tool.















| | | |
|--|-------------------|---------------------|
|  classes.dex | 4/17/2019 2:27 PM | DEX File |
|  classes2.dex | 4/17/2019 2:27 PM | DEX File |
|  classes3.dex | 4/17/2019 2:27 PM | DEX File |
|  classes4.dex | 4/17/2019 2:27 PM | DEX File |
|  classes5.dex | 4/17/2019 2:27 PM | DEX File |
|  classes6.dex | 4/17/2019 2:27 PM | DEX File |
|  classes7.dex | 4/17/2019 2:27 PM | DEX File |
|  classes2-dex2jar.jar | 4/17/2019 2:42 PM | Executable Jar File |
|  classes3-dex2jar.jar | 4/17/2019 2:42 PM | Executable Jar File |
|  classes4-dex2jar.jar | 4/17/2019 2:44 PM | Executable Jar File |
|  classes5-dex2jar.jar | 4/17/2019 2:44 PM | Executable Jar File |
|  classes6-dex2jar.jar | 4/17/2019 2:45 PM | Executable Jar File |
|  classes7-dex2jar.jar | 4/17/2019 2:45 PM | Executable Jar File |
|  classes-dex2jar.jar | 4/17/2019 2:41 PM | Executable Jar File |

Figure 33: Mi Home app dex to jar converted files

In addition to the manual decompiling process, we also used an automated tool called Mobile Security Framework (MobSF). The advantage of this automated tool is, the capability to parse and report security issues in the apps after performing the decompiling process. The tool also provides the option to download the decompiled jar files for further analysis. Moreover, the browser can be used to view and analyze the java source codes. However, like the jadx-gui tool, it has no linking capability to follow classes and methods in different files.

Similar to the AI speakers Apps case, since the goal is to determine how the apps save files and what security techniques are used, we analyzed the source code related to SQLite database, shared preference, internal and external storage operations.

The following section presents the findings in the app static analysis for each of the storage options. For the code analysis, the first thing we did was to figure out how the app creates the database and what security is implemented to protect user data in the SQLite databases.

From the source code, we were able to understand what database and tables the app creates. Figure 34: shows the code snippet from the app used to create the SQLite database and its tables. One of the databases created is called miio.db and has four tables.

```

public void onCreate(SQLiteDatabase sQLiteDatabase, ConnectionSource connectionSource){
    try{
        createTableIfNotExists(connectionSource, MessageRecord.class);
        createTableIfNotExists(connectionSource, ShareUserRecord.class);
        createTableIfNotExists(connectionSource, FamilyRecord.class);
        createTableIfNotExists(connectionSource, GatewayLogRecord.class);
    } catch (Throwable e){
        throw new RuntimeException(e);
    }
}

public class MessageRecord
{
    public static final String FIELD_DEVICENAME = "deviceName";
    public static final String FIELD_ID = "id";
    public static final String FIELD_MESSAGE_TYPE = "messageType";
    public static final String FIELD_MSG_ID = "msgId";
    public static final String FIELD_RECEIVE_TIME = "receiveTime";
    public static final String FIELD_RESULT = "result";
    public static final String FIELD_SEND_USER_ID = "senderUserId";
    public static final String FIELD_USER_ID = "userId";
}

```

Figure 34: Mi Home app database creation code snippet

From the source code, we were able to understand what information the app saves in the database. Message records shared user records, family records and gateway log records are saved in the database as individual tables. For instance, the message record class, we can understand that the app saves deviceName, message type, receive time and senderuserId. The other database file in the app's database is logdb. The geofencing database is used to store location information for Xiaomi push service. In all of the above databases, there is no security consideration for data saved in the tables, except the scope of the databases is MODE_PRIVATE.

Besides, to the static analysis on the source code, we also tried the dynamic analysis using Inspeckage. Figure 35 shows Inspeckage analysis result. However, since the app keeps crashing before the sign in process is done. As a result, we could not proceed with dynamic analysis.

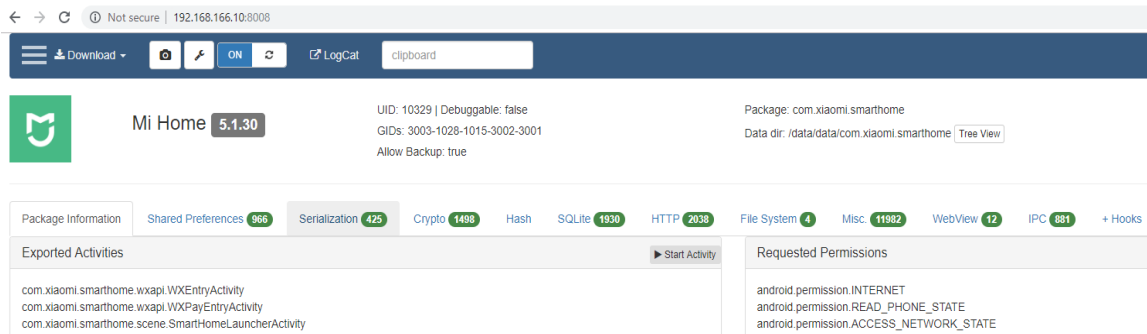


Figure 35: Mi Home app Inspeckage output showing the app name with intercepted activities

From the above analysis, it is quite imperative to understand what information the app is saving and what kind of tables it creates. However, there are no data security considerations, except the databases, are created in the private storage with the default `MODE_PRIVATE` attribute applied. To verify this, we used the MobSF tool to analyze the applied securities. The tool complained that the app does not use encryption methods to secure the databases. On the analysis, we learned that there are many other database libraries. However, since we were interested in the way the smart home app is handling the databases, we selected only four of them. Among the selected four, shown in figure 36, the device record class is more specific to the smart home device. As a result, we opened the class file and analyzed it. The result shows that the database saves smartphone information, including the location and owner's name.

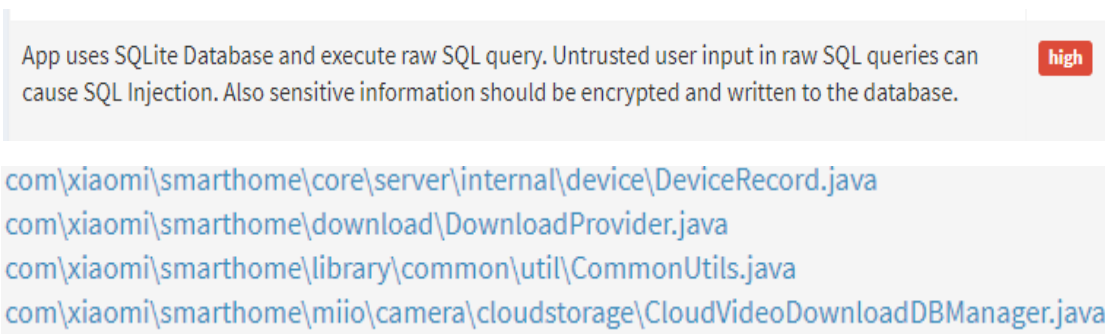


Figure 36: Mi Home app databases handler class files from MobSF security analysis report

In addition to the app analysis, we also did live forensics analysis to understand how the app stores the data in the databases. To perform the analysis, we used the `sqlite3` command tool. After connecting the smartphone to the research computer, we get access to the device through the `adb`

and shell commands. Once we navigated to the apps private storage /data/data/com.xiaomi.smarthome/databases/, we used the sqlite3 tool to load the schema of the database and investigate the tables and their corresponding data. Figure 37 shows the commands executed and the results as shown on the terminal.

```
127|shell@trelteskt:/data/data/com.xiaomi.smarthome/databases # ls
geofencing.db
geofencing.db-journal
hmdb
hmdb-journal
logdb.db
logdb.db-journal
mio.db
mio.db-journal
mio2.db
mio2.db-journal
mistat.db
mistat.db-journal
phone_num3.db
phone_num3.db-journal
shell@trelteskt:/data/data/com.xiaomi.smarthome/databases #
```

Figure 37: Mi Home app database files, showing the commands and list of the databases in Xiaomi app

From the live analysis, we learned that there are six databases. Out of the six database files, mio.db and mio2.db are structurally similar except mio2.db has the current data while the former is empty.

```
root@t0ltelgt:/data/data/com.xiaomi.smarthome/databases # sqlite3 mio2.db
SQLite version 3.7.6.3-Titanium
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .dump
INSERT INTO "devicerecord"
VALUES('90:9F:33:DB:10:DE',1,0,"",NULL,'80943756',NULL,'{"isSetPincode":0,"fw_version":"1.4.1_149","needVerifyCode":0,"isPasswordEncrypt":0,"mcu_version":"0143","is_factory":false}',13,1,0.0,'192.168.166.44',1,0.0,'78:11:DC:E1:6D:0B',NULL,'lumi.gateway.v3','Mi Control Hub',NULL,NULL,"",16,0,NULL,0,-44,1,'neo_house6',"1815102308',NULL);
INSERT INTO "devicerecord" VALUES('90:9F:33:DB:10:DE',1,0,'Close','%s
Close',[1555598167'],'lumi.158d0002164347',{'event.close':"{"timestamp":1555598167,\"value\":[]}",\"event.iam":{"timestamp":1525182169,\"value\":[]}",\"event.no_close":{"timestamp":1553875303,\"value\":[60]}",\"event.open":{"timestamp":1555598166,\"value\":[]}",\"prop.close\":\"1\", \"prop.fw_ver\":\"10\", \"prop.lqi\":\"97\", \"prop.no_close\":\"0\", \"prop.open\":\"0\", \"prop.status\":\"close\"}',{\"isSetPincode\":0,\"fw_version\":\"10\", \"needVerifyCode\":0,\"isPasswor
```

```
dEncrypt":0,"is_factory":false}',14,1,0.0,"2,0.0,"NULL,'lumi.sensor_magnet.v2','Door',NUL
L,NULL,'80943756','lumi.gateway.v3',16,3,NULL,0,0,0,'neo_house6',"','1815102308',NULL;
```

Listing 8: Live Analysis of *miio.db* file using the *sqlite3* command

The result from the live forensic analysis shows that, the *miio2.db* database and the created table have the current values for the information specified in the source code. More importantly, as the investigations show, there is no encryption applied to the data.

We also conducted an offline analysis of the database. After extracting the *miio2.db* file using the *adb* pull operation, we analyzed it using the open source DB Browser for SQLite tool. Figure 38 shows the result with the current data on the table.

| | bssid | version | userId | desc | descNew | descTimeJString | did | eventInfo | extraInfo | |
|---|-----------------|---------|------------|------------------|-----------------|-----------------|-----------------|-------------------|------------------|--------|
| | Filter | Fil... | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 90:9F:33:D5:... | NULL | 1815102308 | Alarming Off ... | | NULL | 80943756 | NULL | {"needVerifyC... | |
| 2 | 90:9F:33:D5:... | NULL | 1815102308 | Open | %s Open | [1540827763] | lumi.158d000... | {"prop.close":... | {"needVerifyC... | |
| 3 | 90:9F:33:D5:... | NULL | 1815102308 | Motion detected | %s Motion de... | [1539489538] | lumi.158d000... | {"event.iam":... | {"needVerifyC... | |
| 4 | 90:9F:33:D5:... | NULL | 1815102308 | Single click | %s Single click | [1540830585] | lumi.158d000... | {"prop.lq":"1... | {"needVerifyC... | |

Figure 38: Mi Home app *miio2.db* file analysis result using SQLite DB browser

In addition to *miio2.db*, we also analyzed the other database files to find out the implemented security. The *geofencing.db* is used to save the location of the smartphone. The *logdb.db* file saves the log file for the phone. The *mistat.db* stores the statistic information for app usage. The *phone_number3.db* file saves the user phone numbers. However, no encryption method is applied to each of them.

Next, to the databases, the other data source we investigated was shared preference. In the source code analysis, we learned that the app creates multiple XML files to save user and app related information in a key-value format. Among them, we suspected that some of them (for instance, `com.xiaomi.smarthome_preferences.xml` and `com.xiaomi.smarthome.globaldynamicsetting.xml`) are directly related to the smart home app. Figure 39: shows code snip `com.xiaomi.smarthome_preferences.xml` file. The app uses the file to

store, the server, WLAN information, port number, and used protocol. However, the access security of the file is dependent on the devices Build SDK version. As shown in figure 39 below, the file is created in MODE_PRIVATE mode if the SDK versions are below 4.

```
private void setDefaultValues()
{
    int i = Integer.parseInt(Build.VERSION.SDK);
    int j = 0;
    if (i >= 11) {
        i = 4;
    } else {
        i = 0;
    }
    settings = getSharedPreferences("com.xiaomi.smarthome_preferences", i);
    i = j;
    Object localObject1;
    while (i < 1)
    {
        if (i != 0)
        {
            localObject1 = new StringBuilder();
            ((StringBuilder)localObject1).append("");
            ((StringBuilder)localObject1).append(i);
            localObject1 = ((StringBuilder)localObject1).toString();
        }
    }
}
```

Figure 39: Mi Home app shared preference manager code snippet

The other file we analyzed in the shared preference was the deviceId.xml file. This file is created with an explicit setting of the MODE_PRIVATE. It is used to save device ID, and it saves the value in a hashed format. For the analyzed files, we learned that other than the MODE_PRIVATE, the app does not apply any encryption algorithms to protect information saved on the storage.

Next, we tried to analyze the same file using the dynamic analysis tool (Inspeckage) to determine how the app writes user information in the XML files. However, as stated above, since the app keeps crashing on the Android KitKat version, we could not proceed with this analysis.

The other data storages used by the app are internal and external memory. Analyzing the AndroidManifest file indicates that the app writes to external storage. Moreover, using the MobSF tool, we learned that the app saves user data without any security considerations. Figure 40 shows the security analysis report from the MobSF tool. During our analysis, we find log data saved on the external storage under the app's name.

| | | | |
|---|-----------|-------------------------------------|--|
| android.permission.WRITE_EXTERNAL_STORAGE | dangerous | read/modify/delete SD card contents | Allows an application to write to the SD card. |
| android.permission.READ_EXTERNAL_STORAGE | dangerous | read SD card contents | Allows an application to read from SD Card. |

Figure 40: Mi Home app external storage permissions in the Androidmanifest file

4.3.1.2. Mi Home App Network Investigation

The other security point of view we analyzed the app was, the implemented network security between the app and the Xiaomi cloud. To perform the investigation, we used two approaches specified in the above research methodology section. The first approach was capturing network traffic using Wireshark. From the analysis, we understood that the app uses TLSv1.2 to encrypt the communication channel.

| | | | |
|-----------------|-----------------|---------|----------------------|
| 47.74.174.229 | 192.168.137.228 | TLSv1.2 | 800 Application Data |
| 47.74.174.229 | 192.168.137.228 | TLSv1.2 | 564 Application Data |
| 192.168.137.228 | 47.74.174.229 | TLSv1.2 | 146 Application Data |
| 192.168.137.228 | 47.74.174.229 | TLSv1.2 | 318 Application Data |

Figure 41: Communication security between Mi Home App and Xiaomi cloud

On the other hand, the Mi Home app sends device status information without any protection. The data includes the smartphone device version and the installed Mi Home app version.

The second approach was using the MITM attack. From the source code analysis using the manual and automated analysis, we could not learn how the app handles third-party certificates, in different web activities except for Facebook web activity call. For the Facebook web activity call, in case of SSL error, the app is forced to cancel requests using the `sslError.cancel()` function.

Then, to conduct the attack, since the app could not run on the Android KitKat version, we used SandroProxy tool a Samsung Galaxy with Android version 6. Using this method, we were able to extract some of the communication APIs and the bearer token used for authorization of the API requests. However, the APIs are limited to the account and stats information of the app. Since the app detects the Sandro proxy, accessing user data was not possible. Therefore, this method did not reveal the APIs used to fetch data. Thus, to reveal the APIs, we used manual code injection.

The manual code injection is done through reverse engineering the app, and manually modifying the code by injecting a piece of code that logs and forward the results to the terminal. After inserting the code, compiling and signing the apk file using self-generated Certificate is required in the process to generate the apk file. Next, to installing the modified apk file and

performing the app supported functionalities, we were able to capture the API calls and the access tokens the APIs used. ADB logcat tool was used to capture the requests.

```
Request$Builder-url: https://us.api.io.mi.com/app/scene/list
NetRequest method?: POST
NetRequest path?: /scene/list
Request$Builder-header: Content-Type
Request$Builder-header: application/x-www-form-urlencoded
Request$Builder-header: Content-Length
Request$Builder-header: 197
Request$Builder-header: Host
Request$Builder-header: us.api.io.mi.com
Request$Builder-header: Connection
Request$Builder-header: Keep-Alive
Request$Builder-header: Accept-Encoding
Request$Builder-header: gzip
Request$Builder-header: Cookie
Request$Builder-header: userId=1815102308; serviceToken=508T2
Request$Builder-header: User-Agent
Request$Builder-header: okhttp/3.4.1
```

Figure 42: ADB logcat output for Xiaomi Mi Home App communication showing API and the header

As we can see from Figure 42, the service token is sent as a cookie with other information. The token value looks like:

```
serviceToken=508T2PH99O1SleqCiq7xQjt0Td1nRSbQK91tPUdh1deYEQn1+jO8HO
cOT3xkaTn6ODfxY2oSi/ezCtNgb/sdJQP1AR7zBG1Yw8n1uHfH4lApC/jvh2heiq+h56iW
iu+gPrd8XMO3ID9gTS2TPE6iOsfL+mvfL4IoEatSdiNmiak=
```

After constructing the header and the cookie values, we sent a request to the cloud using the extracted API; however, we got an error message that states the connection was unauthorized, with HTTP error code 401. From the error, we believe that the service token was used for a single session. After that subsequent request has to use new tokens to access data from the cloud. As a result, in our research, we could not access Xiaomi cloud data using the methods we used to access user data from other devices' cloud.

4.3.2. Xiaomi Lumi-gateway Network Investigation

In this research, we investigated the implemented communication security method in the gateway by intercepting the traffic between the IoT devices and the backend cloud using the

Wireshark traffic analyzer. During the setup, for the initial connections, Xiaomi Lumi gateway uses the TCP protocol; however, once the setup is done, all the communication between the device and the cloud is over UDP. The gateway uses proprietary security over UDP to exchange data with the cloud. In addition to cloud communication, the device continually sends broadcast information to the network over the UDP protocol.

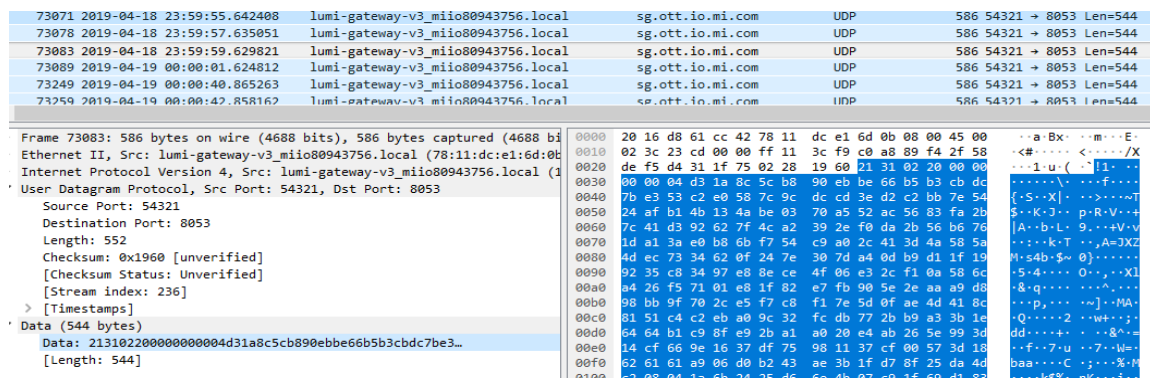


Figure 43: Network traffic between Xiaomi Lumi gateway and cloud showing the encrypted data over UDP

4.3.3. Results Analysis

From the Xiaomi Smart home Kit analysis, we learned that the Mi Home app uses extensive libraries and includes third-party libraries. As a result, in the apps data storage, lots of files and databases are created that are not necessarily required to the smart home app. Besides, in all of the related files, except the MODE_PRIVATE isolation security, there are no implemented crypto security techniques to protect user data. On the other hand, the app uses TLSv1.2 to protect communication between the app and the cloud. One of the security features on the app is the ability to detect MITM attack for response data. Using the Burp Suite tool, we were able to capture some request APIs; however, they do not fetch user data. Furthermore, after manually injecting log code into the source code and recompiled it, we used the adb logcat to recover more APIs and parameter values. However, since the APIs are authorized using one-time session tokens, we could not fetch user data from the cloud. After constructing the APIs and required header and parameter values, we used the Python request library to perform user data requests from the cloud.

However, the responses to our requests were Unauthorized request with HTTP status code 401. Besides, the gateway device uses the TCP protocol to establish an initial connection. Once the setup is done, the gateway uses a proprietary security protocol to protect user data sent over UDP. Also, the device sends of UDP broadcast to the network with a defined size of UDP packets.

4.4. Sen.se Mother

Sen.se Mother is a smart sensor used to measure movement, temperature and other environmental conditions [71]. The Mother collects data using its sensors (called Cookies) and notifies the state of the sensors to the users, and other IoT devices (currently, only Google's NEST) through IFTTT (If This Then That) cloud service. IFTTT is a cloud service that enables intercommunication between apps and devices [72].

Sen.se Mother contains two hardware devices, the Mother hub and motion sensors (Cookies). The Mother acts as a hub to connect the Cookies using radio frequencies of 868MHz in Europe and 915MHz in North America and Asia. The Mother is connected to the Internet using a standard RJ45-terminated ethernet cable. An Internet connection allows communication between the Cookies, Mother hub, cloud service, and the user. Cookies are multipurpose sensors [73]. They can be configured for applications, such as sensing object movements, measuring temperature, walking, presence, sleep monitoring, etc. Also, a single cookie can be set to serve at least two different purposes simultaneously depending on the context of the application. For example, a cookie monitoring door movement can also be configured to monitor the room temperature. Cookies can save data on the individual sensor for up to ten days. Sen.se Mother setup architecture is depicted in figure 44.

The Sen.se Mother hub and the Cookies can be monitored using a smartphone application called Sen.se Pocket Mother. The Sen.se Pocket Mother app is available for iOS, Android and Windows

phones. In addition to the smartphone app, web browsers can also be used to interact with Sen.se Mother cloud service.

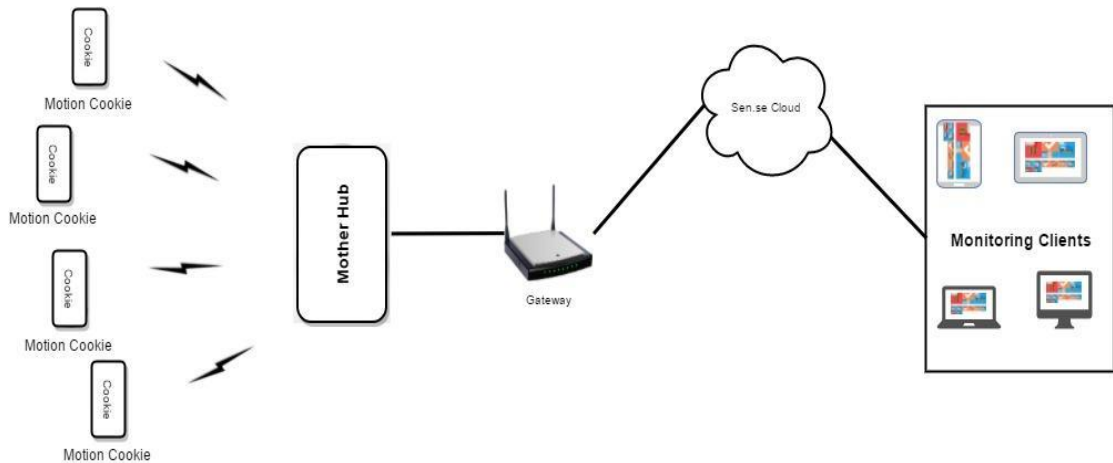


Figure 44: Sen.se Mother set up architecture with Sen.se cloud

For Sen.se Mother IoT device and ecosystem analysis, the research approach is different from the rest of the researched case study devices. The main reason, behind changing the analysis approach is, at the time of conducting this thesis research, the device is out of operation due to the company's lack of support. We believe that the company has run out of business. As a result, the device cannot connect to the cloud, and we were not also able to connect to the cloud from both the companion app and browser clients. As a result, for this research, we were forced to use the previously collected data and analysis results, which was conducted in 2017/2018.

Regarding the security of the device, during our research, we found a number of vulnerabilities on the companion app, the cloud and the network between the device. Moreover, during that period, we tried to report the findings to the company; however, the company did not respond at all. Apparently, the company run out of business and the cloud became unavailable. Therefore, for this research, we included the forensics analysis paper that was submitted to the Journal of Digital Investigation; however, got rejected due to the unavailability of the company.

In addition to the forensics analysis paper, we included the vulnerabilities we reported to the company and MITRE. Instead of including the whole documents in this thesis, we provided a

summary of the results, and the links to the complete google docs will be provided based on requests to the authors. Requests can be done to the following documents.

1. Sen.se Mother Forensic Analysis document
2. Sen.se Mother Vulnerability Assessment and PoC document

4.4.1. Pocket Mother App Analysis

Based on the designed research methodology, we approached the analysis from three different sectors as we did for the AI Speakers. The first part addresses the security techniques implemented in the app to protect user data saved on the smartphone storage. First, we performed static and dynamic analysis on the app using Android reverse engineering approach. Then, we conducted live forensics on the app installed on Samsung Galaxy Note 2 with Android version 4.4 (KitKat). The smartphone is rooted for the research purpose, in the same way, we used for AI Speakers Apps. Finally, we analyzed the app from network investigation perspectives using traffic analysis through Wireshark and Man-in-the-Middle attack approach. In the following section, detailed activities and findings of the analysis are presented.

4.4.1.1. Pocket Mother App Reverse Engineering

For our analysis, the first activity we performed on the app was to perform static code analysis after extracting the app using adb pull from the smartphone. After static analysis, we immediately followed the dynamic analysis to determine how the files are accessed.

First using the manual app reversing process, we extracted the dex files from the apk file and converted them to jar files to analyze using java decompiling tools. After, reversing, we found that the app has lots of native and third-party libraries integrated the app. Moreover, there are seven dex files. From figure 34, we can see that there are seven DEX files. These are the compiled java source code files. Now converting these files to jar files is required to open and analyze the class files as a java source code. To convert DEX files to a jar file, we used the open source dex2jar converter tool.

In addition to the manual decompiling process, we also used an automated tool called Mobile Security Framework (MobSF). The advantage of this automated tool is, the capability to parse and report security issues in the apps after performing the decompiling process. The tool also provides the option to download the decompiled jar files for further analysis. Moreover, the browser can be used to view and analyze the java source codes. However, like the jadx-gui tool, it has no linking capability to follow classes and methods in different files.

Similar to the AI speakers Apps case, since the goal is to determine how the apps save files and what security techniques are used, we analyzed the source code related to SQLite database, shared preference, internal and external storage operations.

The following section presents the findings in the app static analysis for each of the storage options. For the code analysis, the first thing we did was to figure out how the app creates the database and what security is implemented to protect user data in the SQLite databases.

From the source code, we understood that the app creates databases only for the google analytics purpose. Since there are no databases for user data, we did not proceed with the analysis of the databases. However, from the code analysis on the shared preference files, we figured out that the app saves passwords as plain text in a string format. Figure 45 shows the code snippet from the app static analysis. The code indicates that username and password are saved without any security.

```

public final class d {
    public static String a(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context).getString("PREF_LOGIN", "");
    }

    public static void a(Context context, Object obj) {
        Editor edit = PreferenceManager.getDefaultSharedPreferences(context).edit();
        edit.putString("PREF_BOARD", new Gson().toJson(obj));
        edit.commit();
    }

    public static void a(Context context, String str) {
        Editor edit = PreferenceManager.getDefaultSharedPreferences(context).edit();
        edit.putString("PREF_LOGIN", str);
        edit.commit();
    }

    public static void a(Context context, SenseUser senseUser) {
        Editor edit = PreferenceManager.getDefaultSharedPreferences(context).edit();
        edit.putString("PREF_USER", new Gson().toJson((Object) senseUser));
        edit.commit();
    }

    public static String b(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context).getString("PREF_BACKEND", 0) == 1 ? "https://mike.sen.se/pocketmother" : "https://pocketmother.sen.se/pocketmother";
    }

    public static void b(Context context, String str) {
        Editor edit = PreferenceManager.getDefaultSharedPreferences(context).edit();
        edit.putString("PREF_PASSWORD", str);
        edit.commit();
    }

    public static SenseUser c(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context) == null ? null : (SenseUser) new Gson().fromJson(PreferenceManager.getDefaultSharedPreferences(context).getString("PREF_USER", ""), SenseUser.class);
    }

    public static String d(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context).getString("PREF_BADGE_DATE", "");
    }

    public static SectionBoardAndBoard[] e(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context) == null ? null : (SectionBoardAndBoard[]) new Gson().fromJson(PreferenceManager.getDefaultSharedPreferences(context).getString("PREF_BOARD", ""), SectionBoardAndBoard[].class);
    }
}

```

Figure 45: Pocket Mother app shared preference file code snippet

From the source code, in addition to the user credentials, we were able to acquire embedded URLs. Using those embedded URLs, we sent requests to the cloud. The responses also showed us that the server was vulnerable to further attacks. On the other hand, the app did not use any external memory to write data.

4.4.1.2. Pocket Mother App Network and Cloud Investigation

The other security point of view we analyzed the app was, the implemented network security between the app and the Sen.se Mother cloud. To perform the investigation, we used two approaches specified in the above research methodology section. The first approach was capturing network traffic using Wireshark. From the analysis, we understood that the app uses TLSv1.2 to encrypt the communication channel.

| | | | |
|---------------|---------------|---------|----------------------|
| 192.168.1.97 | app-02.sen.se | TLSv1.2 | 389 Application Data |
| app-00.sen.se | 192.168.1.97 | TLSv1.2 | 64 Application Data |
| 192.168.1.97 | app-02.sen.se | TLSv1.2 | 396 Application Data |
| app-02.sen.se | 192.168.1.97 | TLSv1.2 | 988 Application Data |

Figure 46: Communication security between Pocket Mother app and Sen.se Mother cloud

The second approach was using the MITM attack. From the man-in-middle attack, shown in figure 47, we intercepted the plain user credentials, and the APIs requested to the cloud.

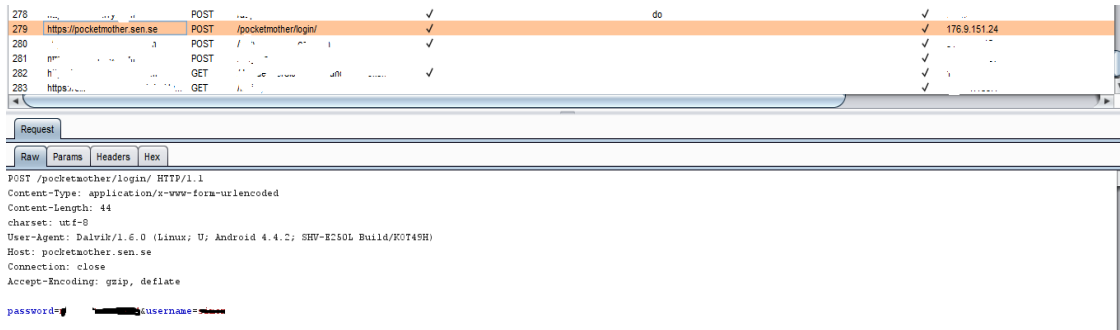


Figure 47: Pocket Mother app man-in-the-middle attack

4.4.2. Sen.Se Mother Network Investigation

The communication between the mother hub and the sen.se cloud is using the WebSocket protocol. The mother initiates a connection to that server. During the session initiation, it sends get request with its information and its MAC or Serial number as an authentication method. But the whole communication is in plain text. And after the session connection is finished cookies data between the mother hub and the Sen.se cloud is sent as a plain text in JSON format. The data contains the method (POST) and Cookie Serial, feed type (the configured application), the value of the current event to be reported or updated and the timestamp of the event off which the data is sent.

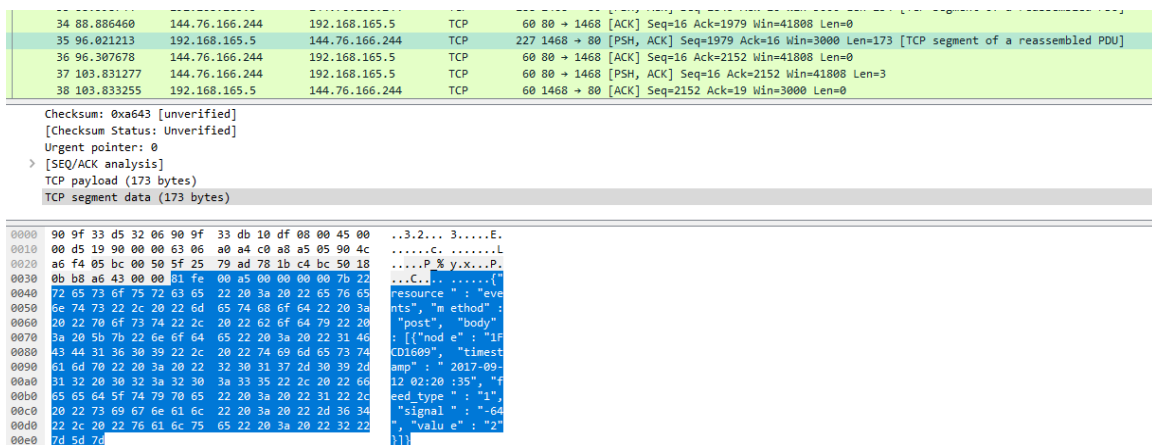


Figure 48: Network traffic analysis between mother and Sen.se cloud

4.4.3. Results Analysis

From the Sen.se Mother analysis, we learned that the Pocket Mother app saves user credentials as a plain text in the shared preference storage. Besides, in all of the related files, except the `MODE_PRIVATE` isolation security, there are no implemented crypto security techniques to protect user data on the app's storage. On the other hand, the app uses TLSv1.2 to protect communication between the app and the cloud. However, the app is vulnerable to a MITM attack. Using the Burp Suite tool, we were able to capture user credentials in plain text and some request APIs. Furthermore, using the DevTools on the browser, we were able to extract all the APIs and the cookies. However, some of the APIs use a one-time session in which we were not able to extract data from the cloud. On the other hand, some of the APIs do not require any credentials such as cookies to extract the data from the cloud. Similarly, the communication between the Mother hub and the cloud is not protected using security methods such as TLS. From this vulnerability, we were able to extract the plain text between the cloud and the mother hub.

4.5. Summary

In this chapter, we presented the analysis on four smart home IoT devices data protection techniques implemented in the Android type of their companion apps, communication security between the devices and the backend cloud, and between the companion apps and the backend cloud. We used Smartphone forensics; Android apps reverse engineering and network traffic interception techniques to conduct the research. Based on the analysis, we were able to acquire artefacts from smartphones and network investigations without significant security challenges. Moreover, using those artefacts, we acquired user data from the cloud for three of the devices.

CHAPTER 5. RESULTS DISCUSSION

In this research, we performed a security assessment on the smart home IoT ecosystem to identify the data protection mechanism implemented to protect user data in apps storage and API security methods. We also studied how the security methods affect data acquisition process for digital forensics investigation. In this chapter, we presented an analysis of the results and how those security assessments results are used to acquire data from the app or the cloud.

More importantly, the goal of the research was to assess how IoT developers consider cybersecurity for IoT devices and their ecosystem. For the research, the main drive was that IoT developers do not consider cybersecurity solutions comprehensively during the development life cycle of IoT devices and their ecosystem, which can be used for digital forensics investigation purpose in solving crimes that directly or indirectly involve smart home IoT device. This lack of comprehensiveness of security consideration has a double impact on IoT users. From the hacker's perspective, the impact could be devastating in affecting the overall way of life of the users. On the other hand, from a law enforcement point of view, those security weaknesses can be used to support hypothesis either in favor of the owner or against the owner depending on the cases.

To address the research, we formulated a series of questions that are directed towards addressing the general goal of the research, which is identifying how data protection security techniques implemented in IoT devices affect the digital forensics investigation process. The questions are stated in the thesis statement section stated in the introduction section of this paper. As a review, we stated the questions in this section too and presented the discussion of the results to answer the questions.

The thesis aimed to identify the current state of the selected smart home IoT devices applications security implementation from digital forensic investigations point of view. To address this, the

research questions stated in section 1.3 were formulated. In the following section, we presented an analysis of the results to answer the stated questions.

5.1. Companion Apps Data Storage Security

Companion apps are interfaces to IoT devices device and user data management. User information used for registering the device and cached cloud data is stored on these companion apps. As a result, this information should be protected from malicious operations. Using the default private access mode and implementing data encryption technologies are recommended to protect these data.

From our investigation of SKT Nugu AI Speaker and the companion security, we learned that there is a number of security vulnerabilities that can be used for the benefit of a digital forensic investigation. From the apps, lack of data storage security enabled us to access the tokens, TIDs and device IDS that can be used to login to the cloud and acquire user data through the APIs extracted using the MITM attack. Specifically, the analysis of the shared preference files revealed all the login credentials for the API authorization to acquire user data from SKT cloud. However, the data on the SQLite database was not valuable in our data acquisition process.

Similarly, Clova AI Speaker and its companion app data security, we learned that there is a number of security vulnerabilities that can be used for the benefit of a digital forensic investigation. From the apps, lack of data storage security enabled us to access the data from the database, which can be considered as user history log. On the other hand, the analysis of the shared preference files revealed that the app uses encryption to secure information saved on the clova.xml file. However, from the other file (clovatoken.xml), even though we were able to identify the token information, it was not useful to acquire user data from the cloud.

From the Xiaomi Smart home Kit analysis, we learned that the Mi Home app uses extensive libraries and includes third-party libraries. As a result, in the apps data storage, a number of

databases are created that are not necessarily required to the smart home app. Besides, in all of the related files, except the MODE_PRIVATE isolation security, there are no implemented encryption-based securities to protect user data. As a result, from the databases, we were able to acquire user data and log history for the sensors.

On the other hand, from Sen.se Mother, we were able to acquire the plain text of the username and password used for registering the device in the shared preference storage. In addition to the login credentials, we were able to recover the cloud APIs used between the Pocket Mother app and the Sen.se cloud. Figure 5 presents a summary of the implemented storage security methods to protect user data in each companion apps.

Table 5: Summary of the companion apps data storage security implementation

| Companion App name | App Version | IoT Device name | Implemented Security | | | |
|--------------------|-------------|-------------------|----------------------------------|---------------------------------------|----------------------------------|---------------------|
| | | | SQLite DBs | Shared Prefs | Internal | External |
| Pocket Mother | 1.1.0 | Sen.se Mother | Default private No encryption | Default private No encryption | Default private No encryption | No external storage |
| Nozzle Clova | 2.13.0 | Naver Clova | Default private No encryption | Default private Partial encryption | Default private No encryption | No Security |
| Nugu Aladdin | 2.3.0 | SKT Nugu | Default private No encryption | Default private No encryption | Default private No encryption | No security |
| Mi Home | 5.5.0 | Xiaomi Smart Home | Default private No encryption | Default private No encryption | Default private No encryption | No security |

5.2. Communication between Companion Apps and the Cloud

The companion Apps are client-side applications available for users to set up, configure and manage IoT devices, and access data stored on the cloud. RESTful APIs are used to communicate between the companion apps and the cloud. Since the apps and the cloud exchange user data, the confidentiality and integrity of the data should be guaranteed. Security protocols such as SSL/TLS are recommended to protect user data. TLS version 1.2 and the latest release version 1.3 are the recommended security protocols. However, IoT applications developers are still hesitating to consider it seriously. In this section, we presented our survey on the selected IoT devices to identify the implemented communication security. We used the Wireshark network analyzer to

capture and analyze the traffic between each cloud and the companion smartphone apps. Table 6 presents a summary of the results and identified the main security weakness for each app.

Table 6: Summary of communication security between IoT companion Apps and cloud

| ID | Companion App name | App Version | IoT Device name | Implemented Security | Remarks |
|-----------|---------------------------|--------------------|------------------------|-----------------------------|---|
| 1 | Pocket Mother | 1.1.0 | Sen.se Mother | TLSv1.2 | Vulnerable to MITM attack due to third-party certificate acceptance |
| 2 | Nozzle | 2.13.0 | Naver Clova | TLSv1.2 | Vulnerable to MITM attack due to third-party certificate acceptance |
| 3 | Nugu/Aladdin | 2.3.0 | SKT Nugu | TLSv1.0 | Vulnerable to MITM attack due to third-party certificate acceptance and other SSL attacks |
| 4 | Mi Home | 5.5.0 | Xiaomi Smart Home Kit | TLSv1.2 | Partially Vulnerable to MITM attack due to third-party certificate acceptance |

During our investigation of the communication security between the Apps and the cloud for the above case studied devices, we identified that all use SSL/TLS protocol to protect the confidentiality and integrity of the data. However, SKT Nugu app uses TLSv1.0 which is on the verge of depreciating and vulnerable old SSL attacks [74], [75]. Moreover, in all of the above implementations, there is a vulnerability that can be exploited to undermine security. The sslError handling mechanism creates vulnerability in allowing communication to proceed using third-party certificates if not handled to properly [76]. In the case of Pocket Mother, Nugu and Clova due to the certificate error handling mechanism, the apps are vulnerable to MITM attacks. Using Burp Suite proxy tool, we were able to intercept the APIs, the tokens, cookies and other user information. In the case of specific to Pocket Mother, we were able to obtain also the username and password used for registering the device. For Mi Home, we intercepted some APIs using Sandroproxy, however, the APIs were not used to fetch user data on the cloud. On the other hand, we used manual code injection to extract some APIs used to fetch user data, however, due to the one-time session tokens used for authorizing the APIs, we were not able to acquire user data.

Moreover, from the investigation, we learned that the Naver Clova app sends crash analytics data without any security. The data includes the main class, the root status of the phone, the network state and the IP address assigned to the phone.

5.3. Communication between IoT Devices/Hub and the Cloud

IoT devices depend on a number of protocols to communicate with backend Clouds. Some use proprietary while some use open protocols. It is common to find protocols such as CoAP [77], MQTT [78], 6LowPAN [79] and other protocols such as HTTP in multipurpose sensor based IoT devices. However, in all cases, the underlying requirement is the confidentiality and integrity of user data during communication with the cloud.

In this research, we investigated the implemented communication security methods in the selected IoT devices by intercepting the traffic between the IoT devices and the backend cloud using the Wireshark traffic analyzer. Similar to the App, SKT Nugu uses TLSv1.0 between the speaker device and the SKT cloud. Moreover, it pushes firmware updates to the device as a plaintext. A checksum is the only security applied to verify the integrity of the firmware. From the captured unencrypted firmware, we were able to get the update package information, including the version. This kind of vulnerability may lead to control the device by replacing the firmware with backdoored firmware. [68] demonstrated Xiaomi smart home gateway hack using the unencrypted firmware update process.

Table 7: Summary of communication security between IoT devices and the cloud

| ID | Device/hub name | Type of Application | Transport Security | Remarks |
|----|-----------------------|---------------------|-------------------------------|-----------------------------|
| 1 | Sen.se Mother | Multipurpose Sensor | None | |
| 2 | Naver Clova | AI Speaker | TLSv1.2 | |
| 3 | SKT Nugu | AI Speaker | TLSv1.0 | Unencrypted firmware update |
| 4 | Xiaomi Smart Home Kit | Multipurpose Sensor | Proprietary security over UDP | |

5.4. IoT Cloud APIs Security Investigation

Most of the APIs we analyzed are unofficial APIs extracted through the App reversing and network analysis using a Man-In-The-Middle attack on rooted Android phones. However, since the approach and implemented securities are different for each of the analyzed devices, in this

section, first, we presented their analysis separately and then a table that summarizes the results. The analysis was focused on the authentication and authorization methods used for the APIs.

5.4.1. Naver Clova

During our research, we used the vulnerability in the Naver Clova app to extract APIs and other artefacts from the app. The app is vulnerable to the Man-in-the-Middle attack due to the flaw in the SSL certificate handling mechanism which lets third-party certificates to be used. For our research, we used a Burp Suite proxy tool to intercept the traffic between the App and the cloud. During the login process, Naver Clova detects proxy settings; therefore, we have to first login without proxy settings. However, once the login process is passed, the subsequent requests can be captured using the MITM attack. Using this method, we were able to extract the communication APIs and the bearer token used for authorization of the API requests. The API can be used to acquire alarm, reminder and alert information from the cloud by setting the requested message type in the body of the API request as shown below. The bearer token is used for authorization of the APIs.

In our research, apart from the network interception, we could not find the bearer token anywhere else. In the app analysis, in the shared_preference folder clova.xml file saves access token that says bearer token and a refresh token; however, using those tokens to request data returns Unauthorized Error (401). Moreover, the length of the bearer token captured in the network and the one that was found in the clova.xml file is different.

5.4.2. SKT Nugu

Like Naver Clova, SKT Nugu Android App is also vulnerable to Man-in-the-Middle attack using proxy tools such as Burp Suite. Similar to the Naver Clova app, for the login process, the app detects a proxy setting using the certificate information. Once the login process is passed without the proxy setting, the subsequent requests can be captured using the MITM attack.

Using this method, we were able to capture the APIs requests and the access token along with another user and device information. In SKT Nugu, for some of the APIs which are used to access another service, additional identification tokens are also used. For instance, for Melon Music Service request, Melon ID is required to access user data. Therefore, the user has to register independently for the service and use that ID to access the service. Later, that ID is included in the API request to access the service and acquire user data from the Melon service cloud.

5.4.3. Xiaomi Smart Home Kit

For Xiaomi smart home kit, the API extraction approach was different as the app has the capability of detecting the proxy applications. However, through the reverse engineering the android app and manual code injection, we were able to capture the API calls and the access tokens for the APIs. We used adb logcat tool to capture the requests.

The service token is sent as a cookie with other information. After constructing the header and the cookie values, we sent a request to the cloud using the extracted API; however, we got an authorized error message, with the HTTP error code 401. That means the service token is used for a one-time session. After that subsequent request has to use new tokens to access data from the cloud, as a result, in our research, we could not access Xiaomi cloud data using the methods we used to access user data from other devices' cloud.

5.4.4. Sen.se Mother

For Sen.se Mother analysis, since the manufacturer is not supporting the device at the time of writing this paper, we used old data that was generated and collected in 2017 in our laboratory. Although through the forensic analysis of the Pocket Mother app, we were able to extract the APIs, primarily, we used network analysis using DevTools integrated into the browser to extract the APIs and associated security tokens. The advantage of using the network analysis is that it gives the request and response headers and parameters in a structured way.

In Sen.se Mother, authorization tokens are used for some of the applications supported by the sensors (like health and secret). However, the tokens are used only for one session. That means, re-requesting the APIs with the authorization tokens results in an unauthorized error response. The tokens were sent along with the cookies.

On the other hand, for applications such as door activity sensing and presence, the APIs did not require any security. As a result, we were able to download JSON data from the Sen.se cloud without any authorization tokens. Moreover, using the APIs we were able to change and delete the configuration of the Cookies without security requirement. For these and other security issues we discovered from the Companion app and the device communication, we tried to contact the company. However, the company did not respond at all. Currently, Sen.se Mother is out of the market and access to the cloud is also not possible. We think that the company run out of business and unfortunately there was no alert or notification about the state during that time and afterwards.

Table 8: Summary of the cloud API security Methods implemented for the selected IoT devices

| ID | App name | IoT Device | Type of Application | API Types | Implemented Security types | | |
|----|---------------|-------------------|---------------------|------------|----------------------------|----------------------------|--------------------|
| | | | | | Authentication | Authorization | Reusability |
| 1 | Pocket Mother | Sen.se Mother | Multi-purpose | Unofficial | Some use Some none | Some used the access token | Some used one-time |
| 2 | Nozzle | Clova | AI Speaker | Unofficial | None | Token | reusable |
| 3 | Aladdin | Nugu | AI Speaker | Unofficial | Token | Token | reusable |
| 4 | Mi Home | Xiaomi Smart home | Multi-purpose | Unofficial | Token | Token | One-time session |

5.5. Forensic Implications of the Securities

In IoT digital forensics investigation, the digital evidence acquisition process addresses a number of data sources in the IoT devices and their ecosystem. Companion apps on smartphones and computers, network traffic between each communication section, backend cloud, the hardware devices, and sensors are digital evidence sources in IoT devices.

However, digital investigators face different challenges in conducting a digital forensic investigation in IoT devices. In addition to the specific challenges faced due to the nature of the

IoT devices, traditional forensics challenges on smartphones, cloud, and network also apply to the IoT investigations. More importantly, data protection security solutions implemented in each of these sources are by far the most significant challenges to be faced by IoT forensic investigators. Encryption technologies applied to smartphones, smartphone app's data storage, API requests, and network communications are the main challenges faced in today's digital forensic investigation [11], [54]. That is also the same as the IoT investigations since most of the data acquisition process relies on traditional digital forensic techniques and tools. For instance, if database encryption is applied to IoT companion apps' SQLite databases, extracting digital artefacts will be difficult if not impossible at all. Similarly, if the key and values of shared preference files are encrypted or saved in a hashed format, there will be no way of understanding the information in the files.

Moreover, researches on different IoT devices show that most of the user data is acquired from the backend cloud. As pointed out by researchers such as [10], [11], the second main challenge to a digital forensic investigation is cloud forensics. In cloud forensics, acquiring data can be achieved using two methods. One is using subpoena from the court and requesting the cloud service provider to hand in data. This has lots of challenges, including Jurisdictions and collaborations requirements from the cloud service providers. The second method is using official or unofficial interfaces provided by the cloud service provider for apps and application developers. In order to acquire data through this method, using cloud APIs is a necessity. In APIs based data acquisition, user credentials, cookies, tokens or a combination of those credentials might be required to access the user data. However, as API securities get stronger, using the APIs to acquire data will be challenging. For instance, if one-time session tokens are deployed for APIs authorization and authentication, re-requesting data from the cloud will not be possible.

Likewise, communication securities, specifically encryption technologies applied to protect network traffic pose challenges to IoT forensics in addition to the network forensic investigations

challenges. Due to this reason, most of the time, the network investigation results are used for flow analysis in digital forensics investigations. In general, IoT network forensic investigation face number of challenges due to the traditional network forensic challenges such as acquisition, the integrity of the data, identifying the devices and seizure of the devices [80], [81].

Therefore, all the above challenges may apply to IoT investigations in the same way they apply to the traditional digital forensic investigation. This is where the weakness in implemented security solutions provides a gateway for IoT forensic investigators. In other words, the security vulnerabilities in IoT devices and their ecosystems provides a way for investigators in acquiring digital evidence from IoT ecosystem [82]. The results of this thesis analysis also demonstrated this claim. Based on the analysis results, we also developed a cloud data acquisition tool for three of the analyzed devices. A brief description of the tools is provided in section 5.7.

5.6. Privacy Implications of the Securities

On the other hand, the security vulnerabilities pose privacy and safety danger to the IoT users if exploited by criminals [52], [82], [83]. As stated in the introduction section, for instance, data collected by the door sensor can be used to infer when the owners leave and return to the house. These data can be used to plan attacks such as robbery to IoT owners. Moreover, IoT devices can be exploited to activate or be inactive in certain situations where they supposed to act accordingly. For instance, a fire smoke detector can be initiated when there is no fire or the other way it can be disarmed in case of fire, which could be devastating.

From our research, we were able to collect sensitive user information used for the operation of the IoT devices. For instance, in the case of SKT Nugu, we were able to acquire user's TID while from Naver Clova we obtained the user's birthdate, name and email information, which could have been used as a credential or registration information for other systems. Besides, in the case of Sen.se Mother, we were able to change the configuration of the sensor's application, such as

changing the door sensor application to presence application. This means that IoT security issues are not solely privacy issues but also safety issues [83].

Therefore, IoT developers should consider protecting not only user data, which affects the privacy of the user, but they should also protect the IoT devices themselves from cyber-attacks that could endanger the safety of the users. For data protection in android client apps, the IoT developers could follow the android security recommendations, that is applying data encryption at the apps level. Besides, avoiding storing or transmitting sensitive information is another option. On the other hand, if storing and processing sensitive information is mandatory to the operation of the IoT devices, using anonymizing techniques such as hashing the information before storing is also another recommended method.

From our studied IoT devices, Naver Clova applied partial security on one of the shared preference files, that is encrypting some of the key and value pair of the file. As a result, we were not able to interpret parts of the information stored on the file. This kind of security techniques helps to protect user data in the app's storages in addition to the protection provided by the Android OS. Besides, Sen.se Mother applied one-time session tokens for some of the APIs used to transport user data for sensor applications such as secret (application to protect precious treasures) and medication (sensor attached to medicine boxes) applications. Moreover, Xiaomi smart home kit applied one-time session tokens to the cloud APIs. Those security methods prevented us from accessing user data from the corresponding clouds.

On the other hand, Sen.se Mother developers did not apply security for some of the sensor applications, which enabled us to change the configuration of the sensors without any credentials. This could happen may be due to the lack of awareness on data and critical system classifications. It might also be a lack of standards and guidelines to protect user data. Therefore, IoT developers should also consider developing comprehensive data protection policies, standards and guidelines.

Regarding data transmission securities, end to end encryption technologies such as SSL/TLS is recommended to protect data in transit.

In general, privacy protection could reach different aspects, including the legislation of countries and data protection regulations in addition to the technical protection methods. Therefore, covering those aspects goes beyond the scope of this thesis. However, based on our analysis, we can say that IoT developers should apply the baseline security recommendations provided by security frameworks such as OWASP Security Guidance in [12] for general security considerations and specific guidelines for each component; for instance, such as android data security guidelines in [16] for companion apps. Also, comprehensive API security methods should be considered during API design and implementation. More importantly, they should design and implement IoT securities, starting from the design stage based on risk assessments. On the contrary, it should be noted that these privacy protection methods could bring challenges to IoT forensic investigators.

5.7. Cloud Data Acquisition Tools

The demonstration tool for cloud acquisition is A Graphical User Interface (GUI) based developed using Python programming language. It uses the python request library to request the APIs and fetch cloud data. The tool is developed for three different devices from the survey; however, since the Sen.se Mother is already out of business and the cloud is not accessible, we minimized the tool for two devices (SKT Nugu and Naver Clova).

To authorize the API requests, the tool requires the credentials from the user which are extracted using the above analysis methods. In the case of Naver Clova, token and device ID extracted from the MITM attack while for SKT Nugu, either from the MITM attack or the app forensics analysis. The tool provides the user with the option to select where to save the downloaded data.

However, as the intention of the tool is to demonstrate how to use the combination of the security weaknesses in the IoT ecosystem, it has limitations that need to be fixed for real applications. For instance, data integrity verification methods such as a hashing algorithm can be added to provide the hash values of the data during downloading. The links to the tool will be provided when this thesis is made publically available.

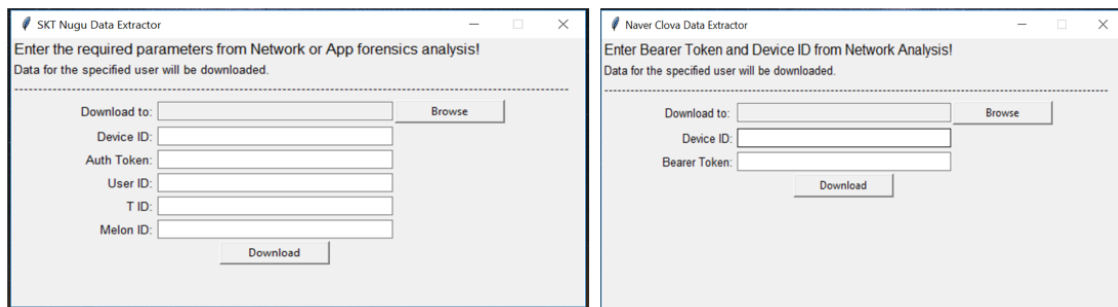


Figure 49: SKT Nugu and Naver Clova AI Speakers cloud Data Acquisition tools

5.8. Summary

In this chapter, we presented the discussion and summary of the results. From the apps data storage security, we showed that almost all of the apps do not use data encryption, except Naver Clova partially encrypt the clova.xml file. We also showed that 3 of the devices companion apps are vulnerable to a MITM attack. From our APIs security analysis, we showed that Sen.se Mother uses the one-time session for some the applications while Xiaomi uses for data access APIs.

On the other hand, both SKT Nugu and Naver Clova use authorization tokens that can be used for multiple requests. Forensics implication of the results is also presented in this chapter. Finally, a brief analysis of the privacy implications of that security weakness and an overview of the recommended protection methods are discussed in this chapter.

CHAPTER 6. CONCLUSION

The main goal of this research was to identify how IoT application developers secure user data and how those security techniques challenge the data acquisition process in the IoT ecosystem for digital forensic investigation purposes.

In IoT forensics investigation, the digital evidence acquisition process addresses several data sources in the IoT devices and their ecosystem. Companion apps on smartphones and computers, network traffic between each communication section, backend cloud, the hardware devices, and sensors are digital evidence sources in the IoT ecosystem.

However, digital investigators face different challenges in conducting a digital forensic investigation in IoT devices. In addition to the specific challenges faced due to the nature of the IoT devices, traditional forensics challenges on smartphones, cloud, and network also apply to the IoT investigations. More importantly, data protection security solutions implemented in each of these sources are by far the greatest challenges to be faced by IoT forensic investigators. Encryption technologies applied to smartphones, smartphone app's data storage, API security methods, and network communications securities are the main challenges faced in today's digital forensic investigation. However, as IoT developers do not comprehensively address security issues in the IoT ecosystem, the vulnerabilities will assist digital forensic investigators.

We approached the research by making a series of research questions specific to data storage security techniques and cloud data access interface security methods implemented in IoT client-side apps, specifically, in Android companion apps.

To answer the research questions, we analyzed data security techniques and cloud data acquisition possibilities for four smart home IoT devices – SKT Nugu, Naver Clova, Xiaomi smart home kit and Sen.se Mother. We investigated the android version of the IoT companion

apps, network investigation between the apps and clouds, between the devices and the clouds, and security of cloud APIs used in companion apps.

We used a combination of research processes and procedures to conduct our investigation. First, we analyzed the companion apps using the static and dynamic analysis by reverse engineering the apps using a variety of open source tools. Then we also performed live forensics analysis on the apps using adb tools. After that, we used the SSL implementation vulnerabilities in the apps to install third-party certificates and conduct Man-in-the-Middle attack using open source proxy tools. Finally, we used Wireshark to investigate the communication traffic by mirroring the traffic between the companion apps and the cloud and between the devices and the cloud.

From the apps data storage security, we showed that almost all of the companion apps we investigated do not use data encryption. They use the default `PRIVATE_MODE` setting to store the databases in the apps specific storage. As a result, as long as the databases can be extracted from the smartphone, all user data saved by the app can be extracted for forensics purposes. For instance, from the Mi Home app database, we extracted recorded devices log data for the door sensors. The records include timestamps for the door open and close activities. Besides, except Naver Clova, all the apps do not use data encryption methods in the shared preference. In the case of the Naver Clova app, some of the key and value pairs are encrypted using the AES algorithm but only in one of the files. That made it challenging in understanding the saved information. If IoT developers follow this kind of trend, which is expected as attack impacts increase, digital forensic investigators may lose valuable evidence from IoT devices. Except for that challenge, we were able to extract user credentials – such as username and password, cookies and tokens from files in shared preference. From some of the apps, we also extracted user information like email, location, and birthday information. In our investigation, we also showed that the apps have the capability to write data on the external storage, but without any security considerations. This enables anyone to read the data saved on the external storage that might be helpful for digital

forensic investigators. However, as IoT developers limit writing user data on external storage or apply data encryption, investigators will face the challenges.

In our investigation, we also showed that three of the devices are vulnerable to MITM attack between the companion apps and the cloud. The attack enabled us to extract cloud APIs and tokens used to authorize the APIs. In the case of Sen.se Mother, we were able to intercept the user's username and password in plain text. However, as IoT developers correctly implement TLS and limit third-party certificate acceptance options [84], digital investigators have to find other options to extract valuable artefacts, including the cloud APIs. For instance, in our investigation, we showed that the Mi Home app was not vulnerable to the MITM attack using the proxies. Thus, we were forced to reverse the app and manually inject code to extract the cloud APIs and header information.

On the other hand, from the network investigation between the devices and the cloud, we were able to intercept insightful information, though most of the user data is encrypted regardless of the SSL version used by the devices. For instance, SKT Nugu uses TLSv1.0, which is vulnerable to a number of attacks and on the verge of obsolescence, while the rest of the devices uses TLSv1.2. Besides, SKT Nugu pushes firmware update as a plain text, while Naver Clova app sends app crash analytic information and some user and device information as a plain text. In our research, we did not try to exploit SKT Nugu using the flaw in the firmware update process. However, researches such as [68] on Xiaomi smart home gateway demonstrate this possibility. Also, [67] presents IoT device hacking using flaws in the firmware update process. Such vulnerabilities may create chances for digital forensic investigators to acquire digital artefacts from IoT devices.

From our APIs security analysis, we showed that IoT developers do not implement API security tokens properly. This resulted in accessing user data from the cloud using tokens extracted from the apps network investigation and app storage forensics. From our researched devices, both SKT Nugu and Naver Clova use authorization tokens that can be used for multiple requests. On the

other hand, Sen.se Mother used a one-time session to authorize the APIs for some of the applications. However, Mother does not use any API authorization security for some of the applications that enabled us to change the configuration of the cookies beyond accessing user data.

On the contrary, Xiaomi uses a one-time session for user data access APIs, that limited our access to the user data saved on the cloud. Therefore, during our research, we could not access user data from Xiaomi smart home kit cloud. This shows that as IoT developers follow such securities, API based data acquisition from the cloud will be challenging, which forces digital forensic investigators to rely on the service providers to acquire user data from IoT clouds.

From our research, we can say that smart home IoT developers do not follow recommended security guidelines to comprehensively secure IoT ecosystem, that helps digital forensics investigators to acquire data from the IoT ecosystem. However, those security vulnerabilities also create opportunities for bad guys to undermine the privacy and safety of IoT users. The results from our studied devices, verify the existence of these double-sided problems in today's smart home IoT devices.

6.1. Future Works

This research focused primarily on the high-level analysis of data security methods used by smart home IoT devices and their ecosystems. The analysis addressed the storage security in the Android companion apps; and communication between the backend cloud and IoT devices and between the companion apps and the cloud. The research can be extended to address communication between the sensors and the devices. Moreover, for this research, we considered only 4 IoT devices; therefore, there is a possibility to address a large set of devices. On the other hand, in addition to the Android versions, there are also iOS and Windows versions of companion apps that need to be addressed. Finally, the research can also be extended to include the app security implemented to protect the apps from reverse engineering and dynamic analysis.

REFERENCES

- [1] A. Berisha-Shaqiri, "Impact of Information Technology and Internet in Businesses," *IMPACT OF INFORMATION TECHNOLOGY AND INTERNET IN BUSINESSES*, vol. 1, Mar. 2015.
- [2] "IoT in electronics is revolutionizing the way we live and work," *Internet of Things blog*, 01-Nov-2017. [Online]. Available: <https://www.ibm.com/blogs/internet-of-things/iot-connected-electronics/>. [Accessed: 04-Jun-2019].
- [3] M. Roberto, B. Abyi, and R. Domenico, *Towards a definition of the Internet of Things (IoT)*. 2017.
- [4] "What is the Internet of Things (IoT)? - Definition from Techopedia," *Techopedia.com*. [Online]. Available: <https://www.techopedia.com/definition/28247/internet-of-things-iot>. [Accessed: 30-Apr-2019].
- [5] S. M. A. Group *et al.*, "Internet of Things (IoT): A Literature Review," *Journal of Computer and Communications*, vol. 03, p. 164, 2015.
- [6] "Gartner," 2017. [Online]. Available: <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016>. [Accessed: 06-Jan-2019].
- [7] M. R. Alam, M. B. I. Reaz, and M. Mohd Ali, "A Review of Smart Homes – Past, Present, and Future," *IEEE Transactions on Systems, Man, and Cybernetics -Part C: Applications and Reviews*, vol. 42, pp. 1190–1203, Nov. 2012.
- [8] R. C. Hegarty, D. J. Lamb, and A. Attwood, "Digital Evidence Challenges in the Internet of Things," *Proceedings of the Tenth International Network Conference (INC 2014)*, pp. 163–172, 2014.
- [9] P. H. Rughani Ph D Assistant Professor, "IoT Evidence Acquisition – Issues and Challenges," vol. 10, no. 5, pp. 1285–1293, 2017.
- [10] J. I. James and Y. Jang, "Practical and Legal Challenges of Cloud Investigations," *The Journal of the Institute of Webcasting, Internet and Telecommunication*, vol. 14, no. 6, pp. 33–39, 2014.
- [11] D. Lillis, B. A. Becker, T. O'Sullivan, and M. Scanlon, "Current Challenges and Future Research Areas for Digital Forensic Investigation," in *CDFSL*, 2016, pp. 9–20.
- [12] "About The Open Web Application Security Project - OWASP." [Online]. Available: https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project. [Accessed: 17-Jan-2019].
- [13] H. Altuwaijri and S. Ghouzali, "Android data storage security: A review," *Journal of King Saud University - Computer and Information Sciences*, Jul. 2018.
- [14] "App Manifest Overview," *Android Developers*. [Online]. Available: <https://developer.android.com/guide/topics/manifest/manifest-intro>. [Accessed: 10-Apr-2019].
- [15] "Data and file storage overview," *Android Developers*. [Online]. Available: <https://developer.android.com/guide/topics/data/data-storage>. [Accessed: 09-Apr-2019].
- [16] "Security tips," *Android Developers*. [Online]. Available: <https://developer.android.com/training/articles/security-tips>. [Accessed: 10-Apr-2019].
- [17] "Full-Disk Encryption," *Android Open Source Project*. [Online]. Available: <https://source.android.com/security/encryption/full-disk>. [Accessed: 07-Jun-2019].
- [18] "Mobile Top 10 2016-M2-Insecure Data Storage - OWASP." [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage. [Accessed: 10-Apr-2019].

- [19] “Telegram – a new era of messaging,” *Telegram*. [Online]. Available: <https://telegram.org/>. [Accessed: 30-Apr-2019].
- [20] “WhatsApp,” *WhatsApp.com*. [Online]. Available: <https://www.whatsapp.com/>. [Accessed: 30-Apr-2019].
- [21] “KaKaoTalk,” *kakaocorp.com*. [Online]. Available: <http://www.kakaocorp.com/service/KakaoTalk>. [Accessed: 30-Apr-2019].
- [22] “Application Program Interfaces (APIs),” *Service Architecture*. [Online]. Available: https://www.service-architecture.com/articles/web-services/application_program_interfaces_apis.html. [Accessed: 02-Apr-2019].
- [23] “What is REST – Learn to create timeless RESTful APIs.” [Online]. Available: <https://restfulapi.net/>. [Accessed: 07-Jun-2019].
- [24] “SOAP Vs. REST: Difference of Web API Services.” [Online]. Available: <https://www.guru99.com/comparison-between-web-services.html>. [Accessed: 02-Apr-2019].
- [25] “Develop a secure API design in a cloud environment,” *SearchCloudSecurity*. [Online]. Available: <https://searchcloudsecurity.techtarget.com/tip/Develop-a-secure-API-design-in-a-cloud-environment>. [Accessed: 18-Jan-2019].
- [26] “Credential stuffing - OWASP.” [Online]. Available: https://www.owasp.org/index.php/Credential_stuffing. [Accessed: 07-Jun-2019].
- [27] “Fuzzing - OWASP.” [Online]. Available: <https://www.owasp.org/index.php/Fuzzing>. [Accessed: 07-Jun-2019].
- [28] “OWASP Top 10 #10: Unprotected APIs [Updated 2018],” *Infosec Resources*, 05-Oct-2018. [Online]. Available: <https://resources.infosecinstitute.com/owasp-top-10-10-unprotected-apis/>. [Accessed: 02-Apr-2019].
- [29] “REST API Security Essentials – REST API Tutorial.” [Online]. Available: <https://restfulapi.net/security-essentials/>. [Accessed: 31-Mar-2019].
- [30] M. Brian, *Web API Design - Crafting Interfaces that Developers Love*. .
- [31] PG Student Velammal Engineering College, Chennai-66 and K. V. Kanmani, “Survey on Restful Web Services Using Open Authorization (Oauth),” *IOSR Journal of Computer Engineering*, vol. 15, no. 4, pp. 53–56, 2013.
- [32] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, “Study on ZigBee technology,” in *2011 3rd International Conference on Electronics Computer Technology*, Kanyakumari, India, 2011, pp. 297–301.
- [33] “IoT technology stack - IoT devices, sensors, gateways and platforms,” *i-SCOOP*. [Online]. Available: <https://www.i-scoop.eu/internet-of-things-guide/iot-technology-stack-devices-gateways-platforms/>. [Accessed: 07-Jun-2019].
- [34] A. Guzman and G. Aditya, *IoT Penetration Testing - Cookbook*. 2017.
- [35] X. Li, “A Review of Motivations of Illegal Cyber Activities,” *Criminology & Social Integration*, vol. 25, no. 1, pp. 110–126, 2017.
- [36] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, “Security, privacy and trust in Internet of Things: The road ahead,” *Computer Networks*, vol. 76, pp. 146–164, Jan. 2015.
- [37] D. Airehrour, J. Gutierrez, and S. K. Ray, “Secure routing for internet of things: A survey,” *Journal of Network and Computer Applications*, vol. 66, pp. 198–213, May 2016.
- [38] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, “A Survey on Security and Privacy Issues in Internet-of-Things,” *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1250–1258, Oct. 2017.
- [39] Z. Ling *et al.*, “IoT Security: An End-to-End View and Case Study,” *arXiv:1805.05853 [cs]*, May 2018.

- [40] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A survey on the security of IoT frameworks," *Journal of Information Security and Appl*, vol. 38, pp. 8–27, Feb. 2018.
- [41] S. Siboni *et al.*, "Security Testbed for Internet-of-Things Devices," *IEEE Transactions on Reliability*, pp. 1–22, 2018.
- [42] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "An Empirical Study on Android for Saving Non-shared Data on Public Storage," *arXiv:1407.5410 [cs]*, Jul. 2014.
- [43] V. Jain, M. S. Gaur, V. Laxmi, and M. Mosbah, "Detection of SQLite Database Vulnerabilities in Android Apps," 2016, p. 11, 2016.
- [44] C. Rodríguez *et al.*, "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices," in *Web Engineering*, vol. 9671, A. Bozzon, P. Cudre-Maroux, and C. Pautasso, Eds. Cham: Springer International Publishing, 2016, pp. 21–39.
- [45] F. Petrillo, P. Merle, N. Moha, and Y.-G. Guéhéneuc, "Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study," in *Service-Oriented Computing*, vol. 9936, Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, Eds. Cham: Springer International Publishing, 2016, pp. 157–170.
- [46] H. Chung, J. Park, and S. Lee, "Digital forensic approaches for Amazon Alexa ecosystem," *Digital Investigation*, vol. 22, pp. S15–S25, Aug. 2017.
- [47] "Amazon Echo data could hold the key to murder trial," *Information Age*, 23-Feb-2017. .
- [48] C. Hauser, "In Connecticut Murder Case, a Fitbit Is a Silent Witness," *The New York Times*, 22-Dec-2017.
- [49] K. M. S. Rahman, M. Bishop, and A. Holt, "Internet of Things Mobility Forensics," 2016.
- [50] M. I. Mazdadi, I. Riadi, and A. Luthfi, "Live Forensics on RouterOS using API Services to Investigate Network Attacks," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 15, no. 2, pp. 406–410, 2017.
- [51] V. R. Kebande and I. Ray, "A generic digital forensic investigation framework for Internet of Things (IoT)," in *Proceedings - 2016 IEEE 4th International Conference on Future Internet of Things and Cloud, FiCloud 2016*, 2016, pp. 356–362.
- [52] N. Akatyev and J. I. James, "Evidence identification in IoT networks based on threat assessment," *Future Generation Computer Systems*, Nov. 2017.
- [53] V. R. Kebande, N. M. Karie, and H. S. Venter, "Adding Digital Forensic Readiness as a Security Component to the IoT Domain," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 1, p. 1, Feb. 2018.
- [54] scar, "An Introduction To Challenges In Digital Forensics," *Forensic Focus - Articles*, 29-Jun-2017. .
- [55] scar, "Current Challenges In Digital Forensics," *Forensic Focus - Articles*, 11-May-2016. .
- [56] E. Oriwoh and P. Sant, "The forensics edge management system: A concept and design," pp. 544–550, 2013.
- [57] Z. Zainal, "Case study as a research method," p. 6, 2007.
- [58] B. Pan, *Tools to work with android .dex and java .class files: pxb1988/dex2jar*. 2019.
- [59] J. Decompiler, *A standalone Java Decompiler GUI. Contribute to java-decompiler/jd-gui development by creating an account on GitHub*. 2019.
- [60] skylot, *Dex to Java decompiler. Contribute to skylot/jadx development by creating an account on GitHub*. 2019.
- [61] *Mobile Security Framework is an automated pen-testing framework for performing static analysis, dynamic analysis.. Mobile Security Framework*, 2019.
- [62] acpm, *Android Package Inspector: dynamic analysis with api hooks, start unexported activities and more. (Xposed Module) - ac-pm/Inspeckage*. 2019.

- [63] “Automated Analysis with Inspeckage,” *Infosec Resources*, 02-Aug-2016. [Online]. Available: <https://resources.infosecinstitute.com/android-hacking-and-security-part-24-automated-analysis-with-inspeckage/>. [Accessed: 30-Apr-2019].
- [64] “Python’s Requests Library (Guide) – Real Python.” [Online]. Available: <https://realpython.com/python-requests/>. [Accessed: 07-Jun-2019].
- [65] “ITU | 2017 Global ICT Development Index.” [Online]. Available: <http://www.itu.int/net4/itu-d/idi/2017/index.html>. [Accessed: 14-Apr-2019].
- [66] “SKT NUGU.” [Online]. Available: <https://www.nugu.co.kr>. [Accessed: 08-Apr-2019].
- [67] G. Aditya, *IoT Hackers Handbook: An Ultimate Guide to Hacking the Internet of Things and Learning IoT Security*. 2017.
- [68] D. Giese, “Having fun with IoT: Reverse Engineering and Hacking of Xiaomi IoT Devices,” p. 86.
- [69] “Naver Clova.” [Online]. Available: <https://clova.ai/ko>. [Accessed: 08-Apr-2019].
- [70] “Xiaomi Mi Smart Home Kit: full specifications, photo | XIAOMI-MI.com.” [Online]. Available: <https://xiaomi-mi.com/sockets-and-sensors/xiaomi-mi-smart-home-kit/>. [Accessed: 08-Apr-2019].
- [71] “Mother • Sen.se.” [Online]. Available: <https://sen.se/store/mother/>. [Accessed: 17-Feb-2018].
- [72] “IFTTT helps your apps and devices work together - IFTTT.” [Online]. Available: <https://ifttt.com/>. [Accessed: 19-Feb-2018].
- [73] “Motion Cookies – Support • Sen.se.” [Online]. Available: <https://sen.se/store/cookie/>. [Accessed: 24-Nov-2017].
- [74] “It’s Time to Disable TLS 1.0 (and All SSL Versions) If You Haven’t Already.” [Online]. Available: <https://www.globalsign.com/en/blog/disable-tls-10-and-all-ssl-versions/>. [Accessed: 01-Apr-2019].
- [75] “TLS 1.0 is no longer used to secure communications | PCI Compliance,” *comodo.com*. [Online]. Available: comodo.com. [Accessed: 01-Apr-2019].
- [76] Y. Liu, C. Zuo, Z. Zhang, S. Guo, and X. Xu, “An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid Web apps,” *World Wide Web*, vol. 21, no. 1, pp. 127–150, Jan. 2018.
- [77] “CoAP — Constrained Application Protocol | Overview.” [Online]. Available: <http://coap.technology/>. [Accessed: 08-Apr-2019].
- [78] “MQTT (<https://mqtt.org/>).” .
- [79] “6LoWPAN: Transmission of IPv6 Packets over IEEE 802.15.4 Networks.” [Online]. Available: <https://www.nsnam.org/docs/models/html/sixlowpan.html>. [Accessed: 08-Apr-2019].
- [80] J. Buric and D. Delija, “Challenges in network forensics,” in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 1382–1386.
- [81] S. Khan, A. Gani, A. W. A. Wahab, M. Shiraz, and I. Ahmad, “Network forensics: Review, taxonomy, and open challenges,” *Journal of Network and Computer Applications*, vol. 66, pp. 214–235, May 2016.
- [82] F. Servida and E. Casey, “IoT forensic challenges and opportunities for digital traces,” *Digital Investigation*, vol. 28, pp. S22–S29, Apr. 2019.
- [83] M. Schunter and A. Wespi, “Editorial: Special issue on IoT security and privacy,” *Computer Networks*, vol. 148, pp. 280–282, Jan. 2019.
- [84] “SslErrorHandler,” *Android Developers*. [Online]. Available: <https://developer.android.com/reference/android/webkit/SslErrorHandler>. [Accessed: 09-Jun-2019].

SMART HOME IoT FORENSICS

2019

Master's Degree

Birhanu, Addisu Afework

Department of International Studies

Advisors: Prof. Jang, Yunsik, Prof. Joshua, I. James

Smart home Internet of Things (IoT) are becoming the mainstream technologies that are being integrated into today society. Recent cyber-attacks and researches on these devices indicate smart home IoT developers do not design and implement data protection solutions comprehensively in the IoT ecosystem. These security weaknesses have different implications for user privacy, safety and digital forensic investigations. This thesis provides an analysis of data protection methods implemented in smart home IoT devices and how the weaknesses can be applied to digital forensic investigation purposes.

In this thesis, we included the analysis of four smart home IoT devices (Sen.se Mother, Naver Clova, SKT Nugu and Xiaomi Smart home) user data protection techniques to identify the vulnerabilities that can be exploited to acquire user data for digital forensic investigation purpose.

To achieve the goal, we analyzed data security techniques and cloud data acquisition possibilities for the selected smart home IoT devices. The investigation is conducted using a combination of forensic analysis of companion apps on smartphones, network investigation between the app and cloud, between the device and the cloud and security analysis of cloud APIs used between companion apps and the cloud.

From the apps data storage security, we showed that all of the apps do not consider data encryption. As a result, if the databases can be extracted from the smartphone, the stored data can

be extracted for forensics purposes. Similarly, except one of the devices' companion app, all the apps do not consider data encryption in the shared preference storage. On the other hand, we identified that some of the devices use one-time session tokens for cloud APIs authorization.

Based on the research, we were able to acquire artefacts from smartphones and network investigations without security challenges. Moreover, using those artefacts, we were able to acquire user data from the cloud for three of the devices. While using such kind of vulnerabilities helps digital forensics investigations to acquire user data from smart home IoT ecosystem, they also endanger users' privacy and safety if exploited by hackers.

Keywords: Smart Home; IoT; Internet of Things; IoT Security; Digital Forensic Investigation; Cloud API Security; App Data Security; Cloud Data; Privacy; Network Forensics

스마트 홈 IoT 포렌식

2019

석사학위논문

발하누 아디수 아페워크

국제학과

지도교수: 장윤식, Joshua I. James

본 논문은 스마트 홈 IoT 장치에 구현된 데이터 보호에 대한 분석과 이에 대한 취약점이 디지털 포렌식 수사에 어떻게 적용될 수 있는지에 대한 분석을 제공한다. 스마트 홈 IoT(Internet of Things)장치는 오늘날 사회에 통합되는 주류 기술이 되고 있다. 최근 장치에 대한 연구와 행해지는 사이버 공격은 개발자가 사이버 보안 솔루션을 종합적으로 설계하거나 구현하지 않았음을 보여준다. 본 논문은 스마트 홈 IoT 장치에서 구현된 데이터 보호 방법에 대한 분석과 디지털 포렌식 조사 목적에 약점이 어떻게 적용될 수 있는지를 제공한다.

본 논문에서는 디지털 포렌식 수사 목적으로 사용자 데이터를 획득하기 위해 악용될 수 있는 취약점을 식별하기 위한 4 가지 스마트 홈 IoT 장치(센스 마더, 네이버 클로바, SKT 누구, 샤오미 스마트 홈)보고 기술 분석을 포함한다.

목표를 달성하기 위해 4 개의 스마트 홈 IoT 장치에 대한 데이터 보안 기술 및 클라우드 데이터 획득 가능성에 대한 분석을 수행한다. 본 논문은 IoT 용 스마트폰 어플리케이션의 포렌식 분석, 앱과 클라우드 사이의 네트워크 조사, 스마트폰 기기와 클라우드 사이의

네트워크 조사, 그리고 스마트폰 어플리케이션과 클라우드 사이에서 사용되는 REST API의 보안 분석 내용을 다루었다.

논문에서는 대부분의 앱 데이터 저장소의 보안이 데이터 암호화를 고려하지 않는다는 것을 보여준다. 그 결과, 스마트폰에서 데이터베이스가 추출될 수 있는 한, 어떠한 데이터가 저장되어있던 포렌식 목적으로 추출이 가능하다는 의미로 볼 수 있다. 앱 데이터 저장소와 마찬가지로, 장치의 앱 중 하나를 제외한 다른 앱들은 공유 환경 설정 저장소에서 데이터 암호화를 고려하지 않는다. 반면, 일부 장치는 클라우드 API 승인을 위해 일회성 세션 토큰을 사용함을 확인했다.

연구 결과, 스마트폰 및 네트워크 조사에서 주요 보안 문제없이 아티팩트를 얻을 수 있었다. 뿐만 아니라, 이러한 아티팩트를 사용하여 세 장치의 클라우드에서 사용자 데이터를 획득할 수 있었다. 이와 같은 종류의 취약점을 사용하는 것은 디지털 포렌식 수사가 스마트 홈 IoT 환경에서 사용자 데이터를 획득하는데 도움을 주지만, 해커가 악용할 경우, 사용자의 프라이버시를 위협에 빠뜨릴 수도 있다.

주제어: 스마트 홈, IoT, 사물인터넷, IoT 보안, 디지털 포렌식 조사, 클라우드 API 보안, 앱 데이터 보안, 클라우드 데이터, 프라이버시, 네트워크 포렌식