SECURITY ANALYSIS OF MOBILE AUTHENTICATION USING QR-CODES

Siwon Sung[1,2], Joonghwan Lee[1,2], Jinmok Kim[1,2], Jongho Mun[2] and Dongho Won[2]

[1]Samsung Electronics, Suwon, Korea
{siwon.sung, joonghwan.lee, jinmok.kim}@samsung.com
[2]College of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea
{jhmoon, dhwon}@security.re.kr

ABSTRACT

The QR-Code authentication system using mobile application is easily implemented in a mobile device with high recognition rate without short distance wireless communication support such as NFC. This system has been widely used for physical authentication system does not require a strong level of security. The system also can be implemented at a low cost. However, the system has a vulnerability of tampering or counterfeiting, because of the nature of the mobile application that should be installed on the user's smart device. In this paper we analyze the vulnerabilities about each type of architectures of the system and discuss the concerns about the implementation aspect to reduce these vulnerabilities.

KEYWORDS

Authentication, Security, QR-Code, Mobile

1. INTRODUCTION

The one-dimensional barcodes have been used in limited areas such as identifier of goods due to the limitation of information capacity. The two-dimensional barcode has been widely used in various fields because of great increase of the information capacity. Thus the user authentication system using the QR-Code becomes possible. Therefore the widely use of smart devices has promoted to spread the QR-Code authentication system. Users could generate the QR-Code image for authentication easily through mobile application on their smart devices. The system is mainly used in a low cost authentication system such as gate access control in buildings and unmanned book rental system in libraries. However, this system is vulnerable to attempts to disguise access to someone else's identity because of the nature of easily replicable QR-Code. In case of mobile device, it is very easy to duplicate through screen capture and be transferred to other user via instant messenger. Thus it is very difficult to eradicate these user violations because of the limitations by those characteristics. In order to overcome these vulnerabilities, various considerations such as security policy, authentication protocols, network security are required. In addition, the system needs also to consider reverse engineering because the vulnerable attributes of the mobile application. The reverse engineering can occur on the various components to configure the system. Particularly the mobile application running on the smart device platform is extremely vulnerable to reverse engineering. Even if an application use very secure authentication scheme, the security might not meet the security levels by lower implementation maturity. Whereas the system uses vulnerable methods, fine implementation recommendations help to prevent those vulnerabilities. In this paper, we suggest some recommendations in implementation points of view to prevent those threats. The remainder of the paper is organized as follows. In Section 2, related works are presented. We introduce the general architecture of the QR-Code authentication system using mobile application in Section 3. The attacker models are described in Section 4. The vulnerabilities in the software implementation point of the view are discussed in Section 5. The result of the forgery attack on real system is reported in Section 6. Then the countermeasure to reduce this software threats is given in Section 7. Finally, we conclude this paper in Section 8.

2. Related Work

QR-TANs is a transaction authentication technique based on QR-Code, allow the user to directly validate the content of a transaction within a trusted device [1]. *Liao et al.* [2] proposed a scheme based on two-dimensional barcode not only eliminates the usage of the password verification table, but also is a cost effective solution. *Lee et al.* [3] proposed an authentication system used Mobile OTP with the combination of QR-Code. *Oh et al.* [4] suggests a technology to authorize users three kinds of QR-Code. *Kao et al.* [5] try to develop a safe and efficient authentication way by using mobile device and implement it for access control system that enhances the security of physical access control systems. *2CAuth* is a two factor authentication scheme that enhances secure usage of application information and preserves usability, without sacrificing user's privacy [6]. *Lee et al.* designed secured *QR-Login* user verification protocol for smart devices that are ready to communicate with QR-Code and proposed a way to keep critical data safe when using the Internet [7]. *Kale et al.* proposed a anti phishing *single sign-on* authentication model using QR-Code [8]. *Malik et al.* introduced the idea of a *one-time password*, which makes unauthorized access difficult for unauthorized users [9].

3. Architecture of Authentication System

The QR-Code authentication system using mobile application (*BAS-MA*) consists of five participants: the users U, the mobile application *MA*, QR-Code generator *QG*, the service provider *SP* and authentication center *AC*. We summarize the notations and acronyms used in this paper in Table 1.

Table 1. Notations and Acronyms

U_i	User
C_i	User credentials for U_i
B_i	QR-Code for authentication issued to U_i
QG	QR-Code generator
SP	Service provider
AC	Authentication center
BAS-MA	QR-Code authentication system using mobile application
AC-AR	Authentication center based architecture

MA-AR	Mobile application center based architecture
A_{it}	Internal attacker
A_{ex}	External attacker

BAS-MA is classified into two kinds of architectures based on the location of the *QG* :

- Authentication Center based Architecture
- Mobile Application based Architecture

In the *AC-AR*, *QG* is located in *AC* that could be regarded as a secure server in the remote location. The basic configuration of *AC-AR* is shown in Figure 1. The procedure through which a U_i can get a valid access to *SP* is the following:

1. U_i is willing to use the *SP*. U_i get authentication through the user authentication scheme of the system.
2. *MA* requires B_i for authentication to *AC*.
3. *AC* retrieves user data from *User Data Storage* then calculates C_i. *AC* asks the *QG* a B_i carrying C_i.
4. *QG* transform the C_i to B_i without any modification on C_i and returns a B_i.
5. *AC* passes through the B_i to *MA* via secure channel.
6. *MA* displays the B_i on its screen and submits to *SP*.
7. *SP* decodes the B_i and extracts C_i. *SP* asks the *AC* the verification carrying the C_i.
8. *AC* verifies C_i and return with an authentication callback.

The B_i is generated in the *AC* on remote. The *MA* has just downloaded the image B_i from *AC* and displays B_i on its screen. The *MA* does not have any logic for generating B_i or processing C_i and is just provided with a bitmap image. If the *MA* is authorized to download the B_i, the C_i can be secured to the reversing attack because they are created and managed on *AC* in remote.

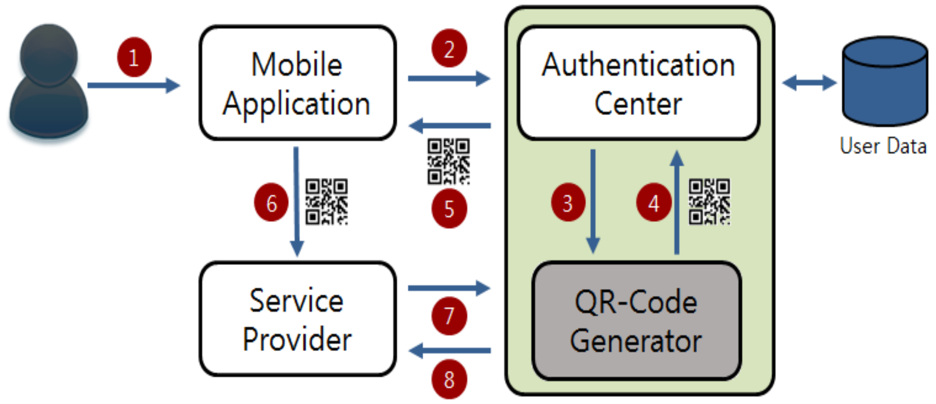

Figure 1. Basic procedure of Authentication Center based Architecture

Otherwise, the *QG* is associated with *MA* tightly in *MA-AR*. The *MA* includes *QG* itself or is able to interact with the external *QG* in the same device. Figure 2 shows the operation flows that the

MA is granted the permission for *SP*. The procedure on *MA-AR* is different to that of the *AC-AR*, yet it has the following differences:

1. U_i try to get a permission to access *SP*.
2. *MA* requests a C_i to generate B_i.
3. *AC* assemble C_i using the *User Data Storage* and return to *MA*.
4. *MA* askes *QG* existed in mobile device a B_i carrying C_i.
5. *QG* generate B_i containing C_i then pass it to *MA*.
6. *MA* gives the B_i to *SP*.
7. *SP* open the B_i and find C_i then *SP* request a verification to the *AC* with the C_i.
8. *AC* checks the equality of the C_i and response.

The *MA* is responsible to manage the C_i in secure and to generate the B_i itself. The *QG* is exposed to threat to get to be attempt software attack.

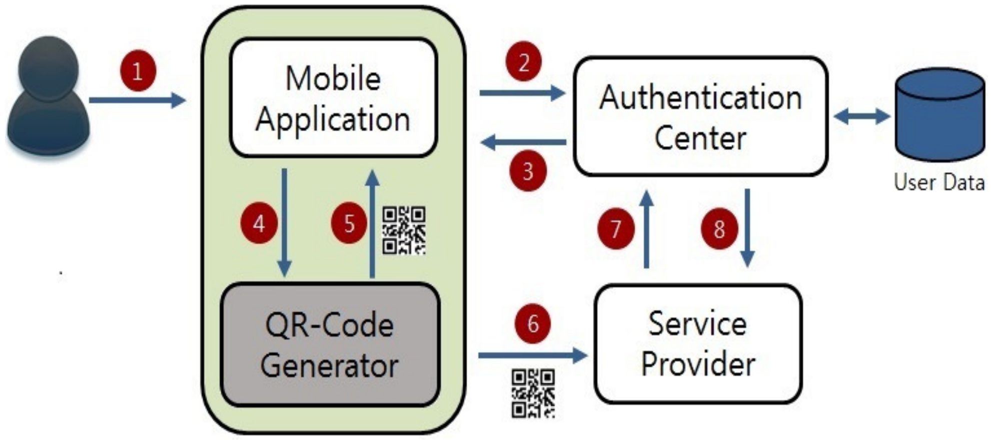

Figure 2. Basic procedure of Mobile Application based Architecture

4. ATTACK MOTELS

The attackers who threaten the *BAS-MA* can be divided into two groups:

4.1. The Internal Attacker

The internal attacker A_{it} is a member of the organization provided services by *SP* and has the correct permissions can access the *SP*. A_{it} may be a malicious user himself willing to access to *SP* by the identity of the other user U_j . The A_{it} also tries to let an unauthorized person to get the authentication to the system by sharing his identity. The A_{it} can try various attacks on the system because they know very well about the *BAS-MA*, and can examine the *MA* on their mobile devices. A_{it} is able to perform repeated incorrect attempts to authentication system with less doubt of the system administrator.

4.2. The External Attacker

Otherwise, the external attacker A_{ex} is not participated in the organization. A_{ex} is the attacker would like to acquire authentication to the system even if they don't have any right permissions. Depending on the mobile application distribution policy, the A_{ex} may have difficulties to obtain the installable packages or binary of the mobile applications. For that reason, A_{ex} distribute a malware to find vulnerabilities in order to steal someone's C_i and B_i.

5. VULNERABILITY ANALYSIS

5.1. QR-Code Cloning

In the *BAS-MA*, the B_i is the key object to achieve the authentication. However, the B_i could be cloned very easily on *MA* through screen capture. The A_{it} might cause the masquerade attack transferring their image to others via an instant messenger. The A_{ex} also try to get the copies of the B_i. In fact, it is impossible to completely prevent such cloning because of the nature of the QR-Code that should be exposed on the screen. The A_{ex} could even take a picture of the B_i using a separate camera.

5.2. Authentication Hijacking

The permanent authentication such as automatic login is usually enabled for the convenience use after once successful authentication. Even if the authentication protocol do not support long term session, the developer tries to implement virtual session by background authentication. In order to do it, the keeping of the user authentication cookies such as user's identity and password in the internal cache are required. These important data in the cache can be easily exposed by the software attack. The attackers are able to hijack the user authentication through the cached data.

5.3. Stored Data Exfiltration

The C_i is stored in the non-volatile storage area under *MA* in the *MA-AR*. The attackers can extract the values through software attack. If the device is rooted, the attack is easier to occur. Because of that, it is recommended that the value is not stored permanently or data should be stored encrypted. Once the encryption key is exposed, the protection using cryptography is useless. After that, the attackers are able to generate the B_i without big difficulty. The data in local storage is very vulnerable. In rooted device, the malware get the read permission under other application's control. The attacker also can access the file system on the device directly or inspect the runtime memory.

5.4. Internal Algorithm Disclosure

The developers usually assume that the compiled source code will be safe from the logic disclosure. However, the powerful tools such as decompiler and logic analyzer are able to disclose the internal logic. The decompiler is able to recover the source codes using the binaries. The logic analyzer can draw the flow graph to help the understanding the logic. It can expose which cryptographic algorithms are used and how to assemble the parameters to construct credentials. In case of the use of the original designed two-dimensional barcode formats for the security purpose, the algorithms should not be disclosed and the software modules for the algorithms should not to be exposed and extracted for other application. If it is possible, the attackers can produce forgery algorithms to perform same work.

5.5. Network Message Eavesdropping

The attackers have various methods to intercept the messages on the mobile device. The general purpose smart devices can be installed any application without any restrictions. The A_{it} could install a network packet monitoring tool on their mobile device to eavesdrop the messages between *MA* and the *AC*. While the mobile platform operates under some policies that restrict the packet monitoring permissions, the A_{it} can attempt to bypass the restriction by using rootkit. In rooted device, all application could obtain root permission.

5.6. IPC Message Eavesdropping

In the *MA-AR*, the *QG* can be located in separated module from *MA*. This is the case in order to use the common module to generate barcode provided by the platform. In this case, the *IPC* (*Inter Process Communication*) communication is generally used. If the *IPC* channel is not secure, messages C_i through *IPC* can be exposed to malware. For example, the *Broadcast* is popularly used in *Android* platform to pass the values to other application. However the implicit *broadcast message* could be broadcasted to all the application in the device not excluding malwares.

5.7. Communication Protocol Vulnerabilities

The secure protocols such as *SSL/TLS* prevent the leakage of the network packet. In this case, the attacker can manipulate the victim's network packet to pass through the malicious *HTTP Proxy* to incapacitate the secure communication. The *HTTP Proxy* can perform the *man-in-the-middle attack* to obtain the secrete key of the channel. *Callegati et al.* presented the *man-in-the middle attack* to the *HTTPS* through *ARP Spoofing* [10]. A_{ex} also can install rogue access point in the near place that the authentication communication usually happened. Because the authentication procedure has occurred in fixed places such as the gateway of the buildings or in front of the unmanned machines, A_{ex} could intercept the communications to proceed authentication through the rogue access point. *Marlinspike et al.* showed *SSL Strip Attack* that do a *man-in-the-middle attack* on *SSL* connections [11].

The Internal logic for communication in the software should be hidden as much as possible. The address of the destination server, *Restful API* names, the configuration of the parameters and the keyword for the message format are usually hard-coded inside mobile applications. The skilful attacker can reconstruct the protocol completely using a piece of this information.

6. Forgery Attack Results

We performed a forgery attack based on the discussed vulnerability in the previous section against the real world *BAS-MA*. The used attacker model was the internal attacker A_{it} and the architecture of the targeted system was *MA-AR*. The A_{it} registered his device to the *AC* through user authentication phase. Then the *MA* downloaded the C_i and generated B_i for displaying. The B_i had expired in 5 minutes to prevent replay attack. The *MA* was able to create a new B_i every 5 minutes without additional network communication with *AC*. This means that the targeted *MA* had stored not only user's credential but also the refresh logic in itself. The targeted *BAS-MA* also used its original two-dimensional barcode format (*Inter-Code*). The attack procedure is described as follows. The A_{it} intercepted the packets of the user authentication phase. The packets were not encrypted and the network channel also wasn't secure. A_{it} could find a plain XML document containing C_i in the intercepted packets. The C_i had the member identification number and organization code. The *AC* received the user's mobile phone number as parameter then returned the C_i. Anyone with a member's phone number can obtain member's credential data. The *QG* was included in the install package of the *MA*. The binary package of *MA* could be extracted from the targeted device without root permissions. The A_{it} could obtain the *QG* by uncompressing the package. The *QG* was a kind of shared object for the targeted device platform. A_{it} tried to decompile the binaries to understand the interface and the parameters of the *QG*. Thus the interface module was implemented in *Java*, the analysis didn't need high difficulty. The free decompile and analysis tools were available without commercial license. A_{it} could find the logic that how to assemble the U_i and timestamps to make the parameters for the *QG*. The shared object could be linked and executed with the forgery application on the same platform. Finally, A_{it} could produce the forgery application that received the victim's mobile number as parameter, then displayed the cloned B_i as *Inter-Code* format. The B_i could be accepted by the verification on the real world *BAS-MA* without any restrictions.

7. Recommendations For Secure Implementation

In Section 5, the various software vulnerabilities on the *BAS-MA* caused by immaturity implemented were discussed. The case study of the forgery attack through the software vulnerabilities on real system was reported in Section 6. Based on this, we suggest the countermeasures against the premature software implementations.

7.1. Interfered Screen Capturing

The screen capture function is usually enabled on common smart devices. It is recommended to block the function while the *MA* shows the B_i. The watermarking on the captured image is also good to give a warning to the A_{it}. It might be able to stop the harmful behaviour of the A_{it}.

7.2. Expiring QR-Code Available Period

The expiration of already issued B_i could cancel the replay attack. Some of salt value has to be mixed with the C_i before *QG* generates the B_i. *Lee et al.* [7] proposed a timestamps based authentication scheme suited for mobile device environment, in which users can be authenticated using a QR-Code. A secure authentication system proposed by *Shamal et al.* [12] that uses a two factor authentication by combining a password and a camera equipped mobile phone, where mobile phone is acting as a authentication token. If the nonce for expiring is hard-coded in the *MA*, it should be obfuscated against to be disclosed by malicious inspection.

7.3. Maintain Local Storage Cleanliness

The candidate data willing to be stored in local storage should be reviewed carefully. The local storage is always in danger to be examined by attackers. If it is possible, the empty of the local area is recommended. The session between *MA* and *AC* should be implemented based on secure protocol. The developer should not arbitrarily implement features beyond the protocol.

7.4. Deliberate Data Storing

The cryptography does not guarantee complete data protection. However, storing data without encryption is more dangerous. The encryption key supplied from trusted party over secure channel is better than included in the binary package. The white-box cryptography helps to hide the key inside the application [13]. If the decryption of the stored value is not required, one-way hash function could be a good choice. In other hand, hardware-backed approach such *as ARM TrustZone* [14] offers a high level security assurance. It provides a completely isolated memory area and data storages physically or logically. However, the application requires a close collaboration with hardware manufacturers to use this approach, and there are many restrictions on the implementation.

7.5. Obfuscation

The software obfuscation is a process to transform the source code into obfuscated code. The variable names are replaced by meaningless name and the execution flows become skewed without any logical error. It is a time consuming and laborious process to analyze the obfuscated binary. The obfuscation is the most effective and comprehensive defence against reversing attack. The obfuscation tool produces a obfuscated result with the source code or compiled binary. This can be fully integrated and automated build process. In fact, the developers do not need to worry to apply the obfuscation. It can be easily accomplished using the 3rd-party tools. In *Android*, the *ProGuard* is included as default in build system. Ensure good performance, it is good to use a commercial tool. The high performance obfuscation tools replace the strings, the constants and the hard-coded encryption keys by encrypted themselves. The obfuscation of control-flow had been proposed by *Chow et al.* [15]. In case that the *MA* includes the encryption key, the obfuscated use of key storage is the essential of the successful encryption.

7.6. Securing Messaging over IPC

When the mobile application structure uses the external *QG* through *IPC*, the *IPC message* carrying C_i to *QG* has to be attention not to be eavesdropped. According to the type platforms, the secure communication between processes might not be supported. The C_i should not be sent through a non-secure IPC. *Chin et al.* examine *Android* application interaction and identify security risks in application components. They provide a tool, *ComDroid*, that detects application communication vulnerabilities [16]. The *TaintDroid* is an extension to the *Android* mobile-phone platform that tracks the flow of privacy-sensitive data through third-party applications [17].

7.7. Detecting Rootkit and Malware

The attack on the rooted device is always more critical. The restricted policies are able to be cancelled easily. The applications are free to invade the protected area each other. It is recommended that let *MA* not launch on the rooted device. A rootkit is a set of malicious tools, in order to achieve the top level privilege of the system. *Kruegel et al*. presents a technique that exploits binary analysis to ascertain, at load time [18]. In case of the difficulty to implement to detect rootkit or malware itself, the use of 3rd-party solution could be alternative.

7.8. Detecting Rogue Access Point

The rogue AP is a wireless access point installed on a wired enterprise network without authorization from the network administrator. They are installed by a legitimate user who is unaware of its security implications or easily smuggled onto enterprise premises by an outsider. It also allows the attacker to conduct a *man-in-the-middle attack*. The white-list based detection approach [19] is appropriate to *BAS-MA*. The valid access point nearby *SP* could be listed up. It also can be an alternative to use the mobile network only such as *3G* or *LTE* instead of *WIFI*.

7.9. Restricting Allowed Device

The number of authentication enabled devices should be limited against the hijacking. Thus the B_i could be downloaded in only predefined mobile device. The device unique key is useful to implement the restrictions. The *IMEI* (*International Mobile Station Equipment Identity*) is a unique number to identity *3GPP* (*GSM, LTE*) mobile device. The *IMEI* could be a seed of the device unique key. The key is transmitted to the server along with the request for B_i. The server verifies the equality of the unique key from mobile device and the registered key in the server. Therefore, even if an attacker acquires a login cookie, it cannot be used in other devices. Of course the device unique key should not be stored in the local cookies.

7.10. Forged Verification

The executable is recommended to be signed by trusted certificate authority. This allows that the platform is able to verify whether the binary is forged. The application can verify the certification itself using platform API. The signature of the binary package could be accessed using the *PackageManager* in *Android*.

7.11. Limiting Distribution Path

The public on-line application store such as *Google Play Store* can be accessed without any limitation to download some mobile applications registered in the store. If the *MA* is registered on the public store, A_{ex} can download and install on their own devices in order to analyze the internal structures. The applications for internal users, such as *MA* are recommended to distribute on a private path. The system operators can send a URL containing the application downloadable link to the valid user via *SMS* or *E-mail*.

8. CONCLUSION

We discussed the vulnerabilities on the QR-Code authentication system caused by an immature software implementation. The main issue of the software security on the QR-Code authentication system is the protection of the logic and data around the QR-Code generator. These vulnerabilities can be overcome by a mature implementation. The improvisation implementation beyond the specification of the protocol is to be eradicated. The locally stored data should be cherry-picked and encrypted. The obfuscation can complete the robust software implementation. With these recommendations, the QR-Code authentication system would be a good solution for physical access system.

REFERENCES

[1] Starnberger, G., Froihofer, L., & Göschka, K. M., (2009) “QR-TAN: Secure mobile transaction authentication”, Availability, Reliability and Security, pp578-583.

[2] Liao, K. C., & Lee, W. H., (2010). “A novel user authentication scheme based on QR-code” Journal of Networks, No.5(8), pp937-941.

[3] Lee, Y. S., Kim, N. H., Lim, H., Jo, H., & Lee, H. J., (2010) “Online banking authentication system using mobile-OTP with QR-code”, Computer Sciences and Convergence Information Technology, pp644-648.

[4] Oh, D. S., Kim, B. H., & Lee, J. K., (2011) “A study on authentication system using QR code for mobile cloud computing environment”, Future Information Technology, pp500-507

[5] Kao, Y. W., Luo, G. H., Lin, H. T., Huang, Y. K., & Yuan, S. M., (2011) “Physical access control based on QR code” Cyber-enabled distributed computing and knowledge discovery pp285-288

[6] Harini, N., & Padmanabhan, T. R., (2013) "2CAuth: A new two factor authentication scheme using QR-code", International Journal of Engineering and Technology, No.5, pp1087-1094.

[7] Lee, Y., Kim, J., Jeon, W., & Won, D., (2012) "Design of a simple user authentication scheme using QR-code for mobile device", Information Technology Convergence, Secure and Trust Computing, and Data Management, pp241-247.

[8] Kale, V., Nakat, Y., Bhosale, S., Bandal, A., & Patole, R. G., (2015) "A Mobile Based Authentication Scheme Using QR Code for Bank Security"

[9] Malik, J., Girdhar, D., Dahiya, R., & Sainarayanan, G., (2014) "Multifactor Authentication Using a QR Code and a One-Time Password", Journal of Information Processing Systems, No.10.

[10] Callegati, F., Cerroni, W., & Ramilli, M., (2009) "Man-in-the-Middle Attack to the HTTPS Protocol", IEEE Security & Privacy, pp78-81.

[11] Marlinspike, Moxie, (2009) "New tricks for defeating SSL in practice", BlackHat DC.

[12] Shamal, S., Monika, K., & Neha, N., (2014) "Secure Authentication for Online Banking Using QR Code", IJETAE–International Journal for Emerging Technology and Advance Engineering

[13] Chow, S., Eisen, P., Johnson, H., & Van Oorschot, P. C., (2003) "White-box cryptography and an AES implementation", Selected Areas in Cryptography, pp250-270.

[14] Winter, J., (2008) "Trusted computing building blocks for embedded linux-based ARM trustzone platforms", Proceedings of the 3rd ACM workshop on Scalable trusted computing, pp21-30.

[15] Chow, S., Gu, Y., Johnson, H., & Zakharov, V. A., (2001) "An approach to the obfuscation of control-flow of sequential computer programs", Information Security, pp144-155.

[16] Chin, E., Felt, A. P., Greenwood, K., & Wagner, D., (2011) "Analyzing inter-application communication in Android", Proceedings of the 9th international conference on Mobile systems, applications, and services, pp239-252.

[17] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B. G., (2014) "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones", ACM Transactions on Computer Systems.

[18] Kruegel, C., Robertson, W., & Vigna, G., (2004), "Detecting kernel-level rootkits through binary analysis", Computer Security Applications Conference, pp91-100.

[19] Park, J., Park, M., & Jung, S., (2013). "A whitelist-based scheme for detecting and preventing unauthorized AP access using mobile device", The Journal of Korean Institute of Communications and Information Sciences, No.38, pp632-640.