IEICE TRANS. INF. & SYST., VOL.E105–D, NO.11 NOVEMBER 2022

LETTER *Special Section on Next-generation Security Applications and Practice*

Hiding Data in the Padding Area of Android Applications without Re-Packaging*

Geochang JEON[†a)], **Jeong Hyun YI**[†b)], ***and*** **Haehyun CHO**[†c)], ***Nonmembers***

SUMMARY Anonymous attackers have been targeting the Android ecosystem for performing severe malicious activities. Despite the complement of various vulnerabilities by security researchers, new vulnerabilities are continuously emerging. In this paper, we introduce a new type of vulnerability that can be exploited to hide data in an application file, bypassing the Android's signing policy. Specifically, we exploit padding areas that can be created by using the alignment option when applications are packaged. We present a proof-of-concept implementation for exploiting the vulnerability. Finally, we demonstrate the effectiveness of VeileDroid by using a synthetic application that hides data in the padding area and updates the data without re-signing and updating the application on an Android device.

key words: *Android application, APK file, data hiding*

1. Introduction

Despite the steady advancement of sophisticated security technologies on the 5G mobile ecosystem, new vulnerabilities are continuously emerging. An attacker usually exploits security flaws in the Android system such as permissions of a manifest file, third-party libraries, repackaging policy, and tampering an application for performing malicious actions.

In this work, we focus on the application file format called APK (Android Application Package) which is the same as the ZIP file format. Specifically, we demonstrate a vulnerability that an Android application can exploit to hide, use, update any type of data in the apk file without re-signing and re-packaging it. In the Android ecosystem, all applications must be digitally signed with a certificate for guaranteeing the authenticity and integrity of applications. However, we found that, if a developer aligns an application with a large value when it is packaged to an APK file, then padding areas, where we can store any data, are created at a certain offset of the APK file. Furthermore, the data stored in the padding area is not used while the APK is digitally signed.

We first show the vulnerability that can store any type of data in the padding areas and restore it on a device when an application executes. In addition, we can update the data without repacking the application. To demonstrate the effectiveness of VeileDroid, we conduct an experiment to bypass Android's signing policy by using a synthetic application that hides data in the padding areas and updates it without re-signing.

Manuscript received February 8, 2022.
Manuscript publicized June 13, 2022.

[†]The authors are with School of Software, Soongsil University, Seoul 06978, Korea.

*This work was supported in part by the Mid-Career Researcher program through the National Research Foundation of Korea (NRF) funded by the MSIT (Ministry of Science and ICT), and Future Planning (NRF2020R1A2C2014336).

a) E-mail: jkch0213@soongsil.ac.kr
b) E-mail: jhyi@ssu.ac.kr
c) E-mail: haehyun@ssu.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2022NGL0003

2. Exploiting the Padding Area

In this section, we present an exploitation technique, VeileDroid, that can inject any type of data in the padding areas in an APK file, and restore it on a device by collecting split data pieces.

2.1 Aligning an Application

Android provides an aligning option for APK files to efficiently access files in an application. By aligning files in an APK file, the Android system can access files in the APK by using offsets, which point to each file, defined in the header of the APK file. Android generally recommends 4 bytes to align files in an APK file, but developers can put values bigger than 4, even 65,536 bytes is available. If the alignment option is used, each file in an APK file starts with offsets that are multiples of the alignment value. Therefore, the maximum size of each padding area can be up to the alignment value.

2.2 Creating the Padding Areas

The padding area is an empty space generated by aligning files in an APK file. Figure 1 shows the location of

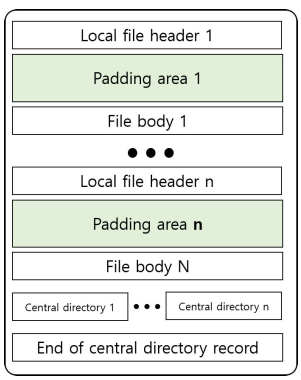

Fig. 1 The location of padding areas in an APK file. Padding areas are created at the end of each file header.

padding areas in an APK file packaged with an aligning option. Padding areas are created at the end of each local file header. However, we note that the padding areas are only created in specific files of which filename ends with `.png`, `.arsc`, `.tflite`, `.version`, and `.gz`.

2.3 Hiding Data and Restoring It

Because there is a limited size of a padding area, if data is larger than a padding area, we need to divide it into several pieces to store them in each padding area. However, each padding area has a different size, and thus, we first need to find the size of padding areas. Then, we divide data and store a piece of data into each padding area with additional information that contains the address of the next piece and the size of the data piece. Also, the information has a flag that tells whether the current piece is the last one. It is worth noting that we can update the data hidden in an APK file without re-packaging it, which bypasses the Android's signing policy. The restoration process is straightforward: We can collect each piece of the data from the padding areas, checking the information.

3. Evaluation

To show the effectiveness of our attack, we use a synthetic application that implements VeileDroid. we installed it on our Android device (Google Pixel 2XL). The application accesses the APK file of itself that existed in the installed directory and aligns it with a large value. It then sequentially updates a series of bytes from '0x00' to '0xAA' in padding areas with the information that identifies the address, the size, and flag of each piece described in Sect. 2.3. Finally, the application restores all pieces of data into a file. Also, the application updates the data and stores it into the padding areas. As a result of the experiment, we confirmed that the application can update the bytes in padding areas without any issue, bypassing the Android's signing policy.

The evaluation result implies that the authenticity and integrity of an application can be broken at any time. Therefore, there can be a security hole in the security of the Android ecosystem. We provide a malicious scenario that can happen if attackers exploit this vulnerability: Attackers register an application in an Android application market without any malicious code. Then they can *unofficially* update the application to perform malicious actions by using the padding area. In this way, attackers can bypass the vetting process of the Android application.

4. Related Work

Despite the extensive effort to improve the security of the Android ecosystem for a long time, attackers are finding new vulnerabilities from the application layer [1]–[4] to perform malicious actions such as runtime information gathering (RIG), code injection, and sensitive data leaking attacks [5], [6]. Taylor et al. [1] analyzed third-party libraries and found that lots of private information is leaked everyday by Ad libraries. Jin et al. [5] found a code injection attack, which inherits the fundamental cause of Cross-Site Scripting attack.

5. Conclusion and Future Directions

In this work, we introduced a new vulnerability that can bypass the Android's signing policy. By exploiting the vulnerability, we can hide any type of data including executable binaries in an application and use it. Also, we can update the data without re-signing the application.

For our future work, we will focus on investigating malicious applications and benign applications in the Android market to check whether there is any malware (or benign application) that use the padding area to hide data for any reason. Also, we will work on developing an effective and efficient analysis method to prevent the use of hidden data for guaranteeing the authenticity of applications that can be used in an market scale analysis. Lastly, another way that we can make use of the padding area is to design of a new obfuscation technique that dynamically instruments an application to conceal sensitive operations.

References

[1] V.F. Taylor, A.R. Beresford, and I. Martinovic, "Intra-library collusion: A potential privacy nightmare on smartphones," arXiv preprint arXiv:1708.03520, 2017.
[2] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, "BadBluetooth: Breaking android security mechanisms via malicious bluetooth peripherals," NDSS, 2019.
[3] M. Rangwala, P. Zhang, X. Zou, and F. Li, "A taxonomy of privilege escalation attacks in Android applications," International Journal of Security and Networks, vol.9, no.1, pp.40–55, 2014.
[4] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2," Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security, pp.1313–1328, 2017.
[5] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G.N. Peri, "Code injection attacks on HTML5-based mobile apps: Characterization, detection and mitigation," Proc. 2014 ACM SIGSAC Conference on Computer and Communications Security, pp.66–77, 2014.
[6] Y. Lee, S. Woo, J. Lee, Y. Song, H. Moon, and D.H. Lee, "Enhanced android app-repackaging attack on in-vehicle network," Wireless Communications and Mobile Computing, vol.2019, Article ID 5650245, 2019.