Poster: Design of Backdoor on Android Devices

Junsung Cho, Geumhwan Cho, Sangwon Hyun and Hyoungshick Kim
Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea
Email: {js.cho, geumhwan, swhyun77, hyoung}@skku.edu

***Abstract*—This paper presents a practical design of backdoor to permanently bypass the screen lock mechanisms (e.g., 4-digit PIN) on Android devices.**

1. Our goal and assumptions

We aim to give insights in designing backdoor that can be used to provide persistent access to a victim's Android device by compromising the secret for user authentication while effectively hiding its presence from the victim.

We assume that a victim uses the PIN scheme to protect her smartphone. Moreover, the victim can often update her PIN secret. Under these conditions, the attacker's goal is to continuously spy on the victim's smartphone without revealing her spying activities. In practice, many likely attackers are such *insiders* rather than strangers in that the people who most want to intrude on a victim's privacy are likely to be in the victim's circle of acquaintances. We also assume that the attacker's backdoor is secretly installed on the victim's smartphone at the initial stage. Probably, an *insider* attacker can often have some chances to install her backdoor on a victim's smartphone in a stealthy manner by either physically accessing the device or performing a social engineering method (e.g., sending a gift app).

2. Design and implementation

To validate the feasibility of the proposed attack, we designed and implemented a proof-of-concept backdoor on Android. Our implementation consists of three components: *trigger application*, *Firebase* (https://firebase.google.com) and *backdoor application*. Here, trigger application and Firebase are controlled by an attacker.

The implementation of the remote triggering feature is important for hiding the backdoor from the device owner (i.e., the victim). We achieved this by using Firebase which supports the communication between the attacker and the backdoor through push notification services. The use of push notifications makes it difficult to detect the malicious traffic from Firebase because many normal applications are also using Firebase for push notification services.

We used two Android devices; the *rooted* Nexus 5 with Android 5.1 Lollipop which plays the role of the victim's device to run the backdoor application while Nexus 5X with Android 6.0 Marshmallow which plays the role of the attacker's device to run the trigger application. For other Android versions, our backdoor can also be adapted. We found that our backdoor performs well on under Android 5.X with just a slight modification of the code. As a result, 84.8% of all Android devices could be vulnerable to our attack in 2016 (https://developer.android.com/about/dashboards).

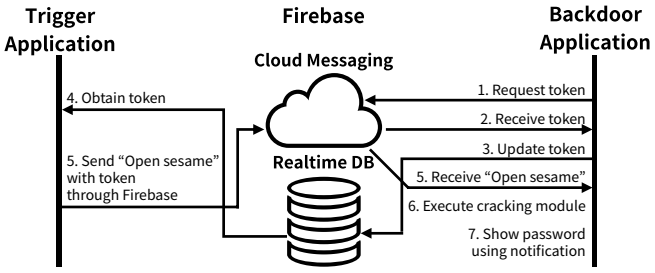

Figure 1. Overview of our backdoor implementation.

We present the overall attack procedure shown in Figure 1. When the backdoor application is installed on the victim's device, the application requests and receives a unique token from Firebase (Step 1 and 2), and updates the real time database of Firebase with the new token (Step 3). Then the attacker retrieves the token from the database of Firebase (Step 4). Whenever the attacker wants to unlock and access the victim's device, the attacker sends an "Open sesame" message attached with the retrieved victim's token to the backdoor application via Firebase (Step 5). Once receiving the message, the backdoor application is automatically triggered to execute the cracking module matching the PIN scheme and eventually finds out the victim's password (Step 6). Finally, the backdoor application informs the attacker of the found password by showing up a notification popup (Step 7).

To hide the backdoor application, we carefully designed it to provide the following properties in terms of resilience against detection. Firstly, we adopt the remote triggering mechanism, thus our backdoor is executed only when there is a request from the attacker while normally hiding its presence. Secondly, our backdoor only communicates with the push messaging server without directly communicating with the attacker. Therefore, network intrusion detection systems cannot distinguish the network traffic of our backdoor from that of other benign applications that are also using push notification services. Thirdly, our backdoor requires the only permissions that are needed to use Firebase. Hence, it is not easy to recognize the potential risk of our backdoor even for security conscious users because those permissions are

also required by normal applications for push notifications.

To crack the victims password, the backdoor application reads the hash value of the password from `password.key`. In particular, the corresponding salt should also be retrieved from `locksettings.db`. The hash values of PIN guesses are calculated, respectively, with the salt and those hash values are eventually tested against the hash value from `password.key`. To avoid unnecessary computational efforts, our cracking procedure first checks whether the password was updated.

3. Evaluation

When casual users encounter this backdoor installation on their Android devices, they cannot realize that the backdoor is installed and used because their unlock secrets are not changed and all the activities of the backdoor can be *temporarily* performed when a specific push notification is triggered by an attacker.

To validate our backdoor's resistance against malware detection, we tested whether commercial anti-virus scanners could successfully detect our backdoor implementation. We used VirusTotal (https://www.virustotal.com) and SandDroid (http://sanddroid.xjtu.edu.cn) that are popularly used for analyzing Android APK files.

VirusTotal provides 55 anti-virus scanners in total to detect various types of malicious applications. However, we found that all those scanners failed to detect our backdoor implementation.

SandDroid performs both static and dynamic analysis on a given Android application to measure its potential risk score between 0 and 100. In our experiment, SandDroid reported 24 as its risk score for our backdoor implementation. We note that this risk score is close to the mean for normal Android applications. The mean risk score of 30 randomly selected Google applications (e.g., YouTube and Gmail) from the Google Play was 26.47 with a standard deviation of 17.87. We also analyzed 30 malicious applications using SandDroid to compare their resulting scores with those of the normal applications. Unlike the normal applications, SandDroid reported 100 as the risk scores for all the tested malicious applications.

Significant changes on the Android application package (APK) file after being repackaged with our backdoor could be an important clue to detect the backdoor. To analyze this, we repackaged a normal application using Firebase with our backdoor implementation and compared the differences between before and after the repackaging in terms of the following aspects: required permissions, activities, services, receivers, providers, intent filters, contained files and file size. As a result, no difference was observed except the increases in the APK file size and the number of files contained in the APK file; specifically, the number of contained files has increased by 8 and the APK file size has increased by about 1.5 Mbytes due to the backdoor's codes and dictionary files.

To measure the execution time, we conducted the dictionary attacks. We used the 4-digit PIN dictionary [1].

TABLE 1. AVERAGE POWER CONSUMPTION (J) TAKEN TO RUN OUR BACKDOOR APPLICATION (μ: AVERAGE, σ: STANDARD DEVIATION).

	`Changed`	`Not changed`
μ	10.393J	0.050J
σ	1.944J	0.050J
`Ratio`	1.669%	0.008%

To avoid the selection bias, we randomly selected 5,000 password samples from real PIN datasets, respectively. We measured the execution time taken from our backdoor implementation. For 4-digit PIN dictionary attacks, 98.72% of the tested PIN samples was cracked within 5 seconds.

To see the impact of our backdoor on power usage, we measured the power consumption of our backdoor application using PowerTutor [2]. We configured the victim's device to include only our backdoor, PowerTutor and default system applications. We measured the power consumption of our backdoor in three different circumstances; `Idle` represents the case that our backdoor has not yet been triggered, `Changed` represents the case that the user changes the password every time, and `Not changed` represents the case that the user never changes the password. We note that in `Not changed` case our backdoor just pops up a notification of the password cracked before without the need of executing the password cracking procedures. We used the last password in our dictionaries as the victim's password to estimate the worst case execution time.

In Table 1, `Ratio` represents the influence of our attack on battery. To take this, we measured the total power usage of the device for an hour. The measurements in `Idle` case indicate that our backdoor application consumes no power before being triggered. Also, `Not changed` and `Changed` results show that our implementation has only a negligible effect on battery.

4. Conclusions

We presented a design of backdoor that makes an attacker consistently unlock a victim's Android device in a stealthy manner. We believe that our backdoor design can also be flexibly used to construct other types of malware (e.g., botnet) by evading existing defense mechanisms such as firewall and intrusion detection system.

Acknowledgment

This work was supported by Defense Acquisition Program Administration and Agency (UD060048AD).

References

[1] H. Kim and J. H. Huh, "PIN selection policies: Are they really effective?" *Computers & Security*, vol. 31, no. 4, pp. 484 – 496, 2012.

[2] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of Conference on Hardware/Software Codesign and System Synthesis*, 2010.