©ISTOCKPHOTO/PEOPLEIMAGES

Digital Object Identifier 10.1109/MCE.2018.2797618
Date of publication: 10 April 2018

User Credential Cloning Attacks in Android Applications

Exploiting automatic login on Android apps and mitigating strategies.

By Junsung Cho, Dayeon Kim, and Hyoungshick Kim

Automatic login is a commonly used feature of smartphones, because their small keyboards make it difficult to key in user credential information. However, this feature may pose a serious risk to smartphone users' privacy. The stored data for automatic login could be stolen by an attacker, resulting in identity theft. In this article, we demonstrate an execution of this attack in a systematic manner through two real-world Android application case studies by implementing a prototype. We also discuss five possible defense strategies to mitigate the risk of user credential data being stolen from the application files.

AUTOMATIC LOGIN VULNERABILITY

Automatic login is a feature for user authentication whereby user credentials are stored locally and used when verification is required. It is frequently provided as a common means to enhance convenience by avoiding the necessity of keying a user name and password into an application (e.g., a web page) each time it is accessed.

Automatic login is useful, but it can be harmful as well; the stored user credentials can be stolen or misused by unauthorized parties. For example, many websites offer the opportunity to remain logged in to a website via a browser-saved cookie that caches the user credential information. However, cookies can simply be stolen [1] and used by an attacker to impersonate someone. In the worst-case scenario, if the stored credential is used for a single sign-on authentication system (e.g., a Google account), then the attacker would have access to all the services it protects. Also, anyone having physical access to a system using an automatic login could unlock it without a victim's password even when it cannot be stolen.

In this article, we revisit this problem, focusing on a different domain of the Android platform and present a generic attack strategy called a *user credential cloning attack*, which might be a serious threat to Android applications that support the automatic login feature. This attack is an attempt by a malefactor to steal a victim's credential data from his or her Android app and log in to the victim's account using these stolen access rights via the attacker's Android application. In practice, people tend to prefer the automatic login feature in small-touchscreen smartphones in particular, as opposed to personal computer (PC) environments, because of its quickness and convenience [2], [3]. The majority of existing Android apps (e.g., Facebook, Twitter, and Skype) support the automatic login feature by default. We show how the user credential cloning attack can be effectively devised against such Android applications.

To show the feasibility of the proposed attack strategy, we implemented the user credential cloning attack against two Android applications, Starbucks and MOCI. The Starbucks application provides various services, such as paying for orders, earning loyalty rewards, and finding stores. MOCI is a well-known anonymous social networking application (https://www.moci.kr/) in South Korea. For Starbucks, we can see that a user's authentication credentials are stored in an Extensible Markup Language file called `Shared Preferences` (`com.starbucks.co_preferences.xml`) to manage an application's private primitive data in key-value pairs. For MOCI, a user's authentication credentials are stored in `SQLite Databases` (`moci.sqlite`). If an attacker obtains this kind of file from a victim's app and replaces his or her own file with the victim's, the attacker can log in to the victim's account via the automatic login feature.

A surprising large-scale survey carried out by Tencent revealed that about 80% of Chinese smartphone users had rooted their Android smartphones during 2014.

Our key contributions in this article can be summarized as follows:

- We present a generic procedure for user credential cloning attacks against Android applications that support the automatic login feature.
- We evaluate the feasibility of the proposed attack through two real-world case studies (Starbucks and MOCI) on the Android platform.
- We suggest five possible defense mechanisms to mitigate user credential cloning attacks.

THREAT MODEL

For our study, we assume that an attacker's malware for a user credential cloning attack is installed on the victim's smartphone at the initial stage (e.g., through social engineering [4], repackaging of apps [5], or the official app market [6]). In practice, a significant number of Android devices are often infected. For example, the DroidDream malware infected more than 260,000 devices within 48 h [7].

We also assume that the victim's smartphone is already rooted or will be rooted before performing the user credential cloning attack, because some storage spaces on Android (i.e., `Shared Preferences`) can be accessed only on rooted devices. While this may be a rather ambitious assumption, it appears that rooting is fairly common. In practice, many Android users have rooted their devices to update their operating system with a new version of Android or to remove unnecessary built-in apps. According to Google's official report in 2016 [8], about 5.6% of all Android devices were rooted by device owners or potentially harmful applications. In fact, Android devices can often be unintentionally rooted with a security bug [9]. Also, another study on Android rooting showed that approximately 7% of survey participants rooted their devices [10]. Finally, a surprising large-scale survey carried out by Tencent revealed that about 80% of Chinese smartphone users had rooted their Android smartphones during 2014 [11].

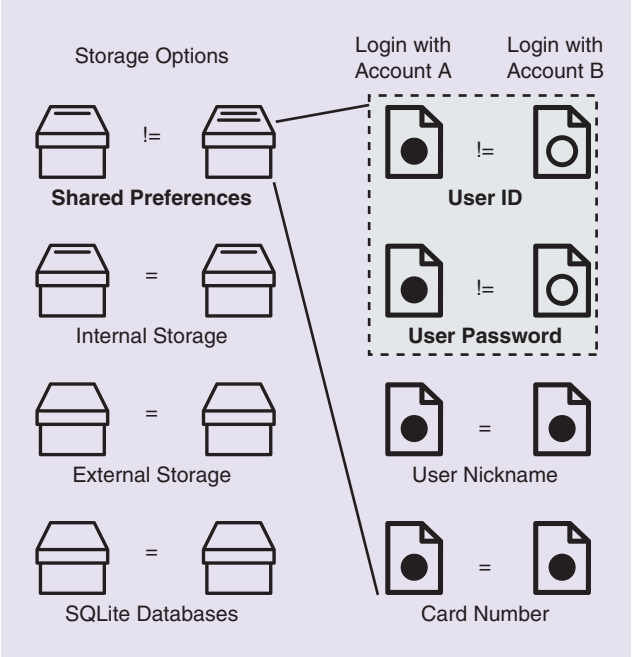

FIGURE 1. An example of finding a user credential via a monitoring tool.

USER CREDENTIAL CLONING ATTACK

The user credential cloning attack is an attempt by an attacker to impersonate a victim by stealing the person's credential data and copying them to his or her own device. Formally, such a credential is defined as a token for user authentication, e.g., a user name and password pair that is bound to a particular individual.

We show the potential risk of a user credential cloning attack in Android applications in particular. A basic attack scenario can be initiated from the time a malicious program is installed on the victim's Android device. Here, for simplicity, we assume that this malware has the appropriate permissions to access the files related to the victim's credential data. The detailed procedure is as follows:

1) The malicious program attempts to collect candidate files in a target Android application that may store user credential data.
2) Having successfully collected the candidate files, the malware secretly uploads them to the attacker's server.
3) After receiving the victim's files, the malefactor compares his or her own files with those of the victim to identify the differences between them.
4) The attacker updates his or her own files with the user credential data (carefully selected from different parts of the victim's contents) and runs the target Android application to log in to the application server with the victim's account.

To perform a user credential cloning attack, a malefactor must know which storage option is used for storing user credential data in the target app. As shown in Figure 1, there are four different local storage choices [12] for Android applications: 1) `Shared Preferences`, 2) `Internal Storage`, 3) `External Storage`, and 4) `SQLite Databases`. To identify the exact location of the user credential data for automatic login, the attacker compares the clean installation states of the target application files with the states of these files after the automatic login has been performed. The changed files can be considered to constitute the candidate storage options for storing the user credential data.

Moreover, the attacker attempts to distinguish the contents, e.g., the user identification and hash of the user password, associated with the user account from the device-specific

contents of the changed files. The culprit can implement this step by systematically testing every possible combination of fields or manually inspecting the fields until a set of the fields related to the user account is discovered. This task may be relatively slow compared to the others, but it must be performed only once.

Figure 2 shows an overview of the user credential cloning attack. We assume that both the target and the malware are installed on the victim's smartphone. The malicious application extracts the victim's user credential data files from the target app's storage and uploads them to the malefactor's remote server. Finally, the credential files located at the attacker's smartphone are replaced or updated with the victim's credential files.

CASE STUDIES

In this article, we show the feasibility of the user credential cloning attack through case studies involving two Android applications. We chose the Starbucks and MOCI apps in particular and tested their security against our attack because these two are among the most widely used in each category (mobile commerce and social media) and also provide an automatic login feature that is configured by default.

In our experiment with those applications, we used two Android devices: 1) a rooted Google Nexus 5, running Android 6.0, in the role of the victim's device (D_{victim}) and 2) a rooted Google Nexus 5X, running Android 7.1, as the culprit's device (D_{attacker}). We also used a PC, which fulfilled the role of the attacker's server (S_{attacker}).

STARBUCKS

We first performed a user credential cloning attack on the Starbucks application. Our implementation followed the procedure described in the "User Credential Cloning Attack" section. After triggering the automatic login with different user accounts on the app, we attempted to observe which storage files in the application were changed and then found that there were several changes in the `Shared Preferences` file (`com.starbucks.co_preferences.xml`). An example of this file is presented in Figure 3. We can see that basic user account information—the user identification, MD5 hash of the user password, user's name, mileage points, the application's unique device ID (UDID), number of coupons, Starbucks card number, and so forth—is included in the file. (In Figure 3, we present some fields, e.g., the hash of the password, with fake data instead of a user's original data.) Among these fields, we identified in particular those necessary for automatic login by repeatedly replacing parts randomly selected from all of the changed fields with another user account's contents until automatic login was successfully processed. The identified fields are as follows: `LOGIN_USER_NICKNAME_INFO_ID`, `LOGIN_PASSWORD_INFO_ID`, `LOGIN_REAL_NICKNAME_INFO_ID`, `LOGIN_REAL_NICKNAME_USE_ID`, and `LOGIN_ID_INFO_ID`.

> The Starbucks and MOCI apps are among the most widely used in each category (mobile commerce and social media).

We implemented an Android application and installed it on (D_{victim}) to copy the Shared Preferences file into external storage and upload it in (D_{victim}) for (S_{attacker}). When the file was uploaded, we extracted from it the values of the five necessary fields. With these values extracted from (D_{victim}), we updated the same five fields in the Shared Preferences file on D_{attacker} via Android Debug Bridge (ADB), which is a debug support tool for communication between a host and an Android device. Note that all of the steps can be processed automatically within a few seconds. After successfully completing all the tasks, we could log in to the Starbucks app on D_{attacker} with the user account used in D_{victim}. Figure 4 shows the attack results before and after performing a user credential cloning attack on the Starbucks application. In this figure, we can see that different account information is displayed according to which user credential data are used.

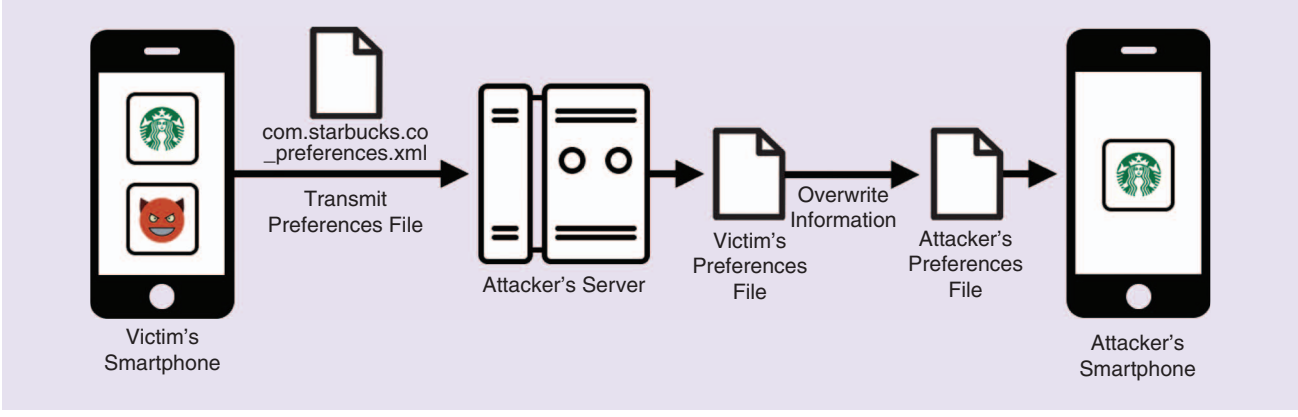

FIGURE 2. An overview of a user credential cloning attack.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <String name="LOGIN_USER_NICKNAME_INFO_ID">Junsung</String>
  <String name="LOGIN_USER_INFO_ENCRYPT">Y</String>
  <String name="rewardStarCount">8</String>
  <String name="E_FREQUENCY_MEMGRADE">M1</String>
  <String name="pushApplyYN">Y</String>
  <String name="isEfreqFirstVisit">false</String>
  <String name="STARBUCKS_APP_UDID_INFO_ID">c38bb3d8-8b95-494...</String>
  <String name="couponCount">0</String>
  <String name="LOGIN_PASSWORD_INFO_ID">LseTuIJOISi+w/R1GDrJtA==</String>
  <String name="LOGIN_REAL_NICKNAME_INFO_ID">Junsung Cho</String>
  <String name="LOGIN_REAL_NICKNAME_ID">Y</String>
  <String name="326">https://www.istarbucks.co.kr/interface/w...</String>
  <String name="E_FREQUENCY_BARCODE">9071000000001380</String>
  <String name="GCMID">APA91BFLecB3wz7wLVnhvG1545_oGPpAR1jwpj...</String>
  <String name="cardNo">6099000000002028</String>
  <String name="326">https://www.istarbucks.co.kr/interface/w...</String>
  <String name="LOGIN_ID_INFO_ID">GOROAD</String>
</map>
```

FIGURE 3. A user credential file (com.starbucks.co_preferences.xml) in the Starbucks application. Some important fields are highlighted in red boxes.

MOCI

We also performed a user credential cloning attack on the MOCI application. Unlike the Starbucks application using the `Shared Preferences` file, MOCI uses the `SQLite Databases` file (`moci.sqlite`) to store the user credential data for automatic login. The `moci.sqlite` file consists of three tables. The user credential data are stored in the `prefs` table, with 44 records. When we compared the `prefs` table for D_{attacker} with that for D_{victim}, the specific eight records were different only between those tables (see Table 1). Therefore, when we replaced those records for D_{attacker} with the ones for D_{victim}, we successfully performed the user credential cloning attack.

Figure 5 displays the results before and after performing our attack on the MOCI application. In this figure, we can also see that different account information is displayed according to which user credential data are used. Even though the user credential attack works well by simply copying all of the eight changed records, we concluded, after carefully inspecting the differences between the eight D_{attacker} and D_{victim} records, that only two (`ar` and `as`) are associated with the user credential data.

USER CREDENTIAL CLONING ATTACK PREVENTION

In this section, we discuss five possible defense strategies to mitigate user credential cloning attacks.

FIGURE 4. The results of a user credential cloning attack on the Starbucks application: (a) before and (b) after the attack.

BINDING USER CREDENTIAL DATA TO A SPECIFIC DEVICE

Our first strategy is to bind user credential data in the target application to a specific device to prevent an attacker from reusing user credential data for his or her own device. To achieve this, a typical method is to use cryptographic primitives, such as encryption.

For example, we can store user credential data in encrypted form so that only an authorized application (running on a specific device) can access the plaintext user credential data in that form. For this purpose, user credential data are encrypted with a key that is generated at runtime by the application with device-specific attributes. Such attributes may include the device model, operating system, International Mobile Equipment Identity (IMEI), mobile equipment identifier, International

Table 1. Eight different records in the moci.sqlite file between $D_{attacker}$ and D_{victim}.

Key	Value (ID_1)	Value (ID_2)
aq	1489063974653	1489061984928
ar	d3b2cd658c458400	9d2eb8a790b43f0f
as	58c149dab2d7330551ac…	58c14860cb8ac6317ca9…
ci	84789	86400
device_info	LGE Nexus 5X Android 7.1.1 (API 25)	LGE Nexus 5 Android 6.0.1 (API 23)
dv (in Korean)	No one knows who you are. What is my ideal type?	No one knows who you are. Please show off your lover!
dx	20	23
push_token	APA91bEUzS5v1J6i35p…	APA91bGcApwFvWdZOOj…

Mobile Subscriber Identity, electronic serial number, media access control (MAC) address from a device's Wi-Fi or Bluetooth hardware, and phone number. If the attacker does not know the exact algorithm to generate the device-specific key, the plaintext user credential data cannot simply be extracted from the application's storage files.

Against this defense approach, an attacker may certainly attempt to reverse-engineer the application to analyze the key generation algorithm and its parameters. However, the use of device-specific keys can considerably raise the bar for attackers, because the cost of reverse engineering is significantly greater than that of reading files. In addition, there exist a number of antireverse-engineering techniques and tools [13]–[15] to protect program codes, although it is unclear whether such techniques are sufficiently strong to protect sensitive software systems such as an encryption algorithm implementation.

INCREASING THE SEARCH TIME FOR IDENTIFYING USER CREDENTIAL DATA

As described in the "User Credential Cloning Attack" section, an attacker has to search in the collection of candidate files for user credential data for extraction. Although this search process needs to be performed only once, we can measurably boost the difficulty of the attack by increasing the number of candidate paths for the credential data. That is, the number of changed fields and/or files can be intentionally increased to make it more difficult for the attacker to guess the correct positions of the targeted information. Formally, given n candidate fields, $2^n - 1$ different combinations should be tested in the worst case. Therefore, if we use a reasonably large value of n, identifying user credential data becomes computationally more expensive.

PREVENTING SUSPICIOUS LOGIN ATTEMPTS

Unfortunately, after successfully updating the attacker's files with a victim's user credential data, it is not feasible to detect automatic login attempts performed by the malefactor's device, because the login attempt messages of both the victim and the attacker's device are transmitted in exactly the same manner. Alternatively, therefore, we can prevent suspicious login attempts aimed at identifying the positions of user credential data in candidate storage files by targeting login patterns that are significantly different from normal ones. For example, in the course of an attack, consecutive login attempts with the wrong user credential data are periodically made during a relatively short time. To prevent such login attempts, a straightforward solution is limiting the number of consecutive failed attempts from a particular device, e.g., based on its IMEI or MAC address. If we set this number to, say, three, an attacker needs to exactly identify user

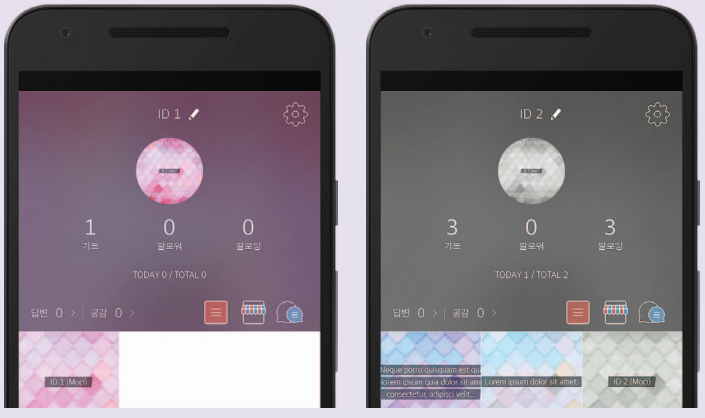

FIGURE 5. The results of a user credential cloning attack on the MOCI application: (a) before and (b) after the attack.

> We can measurably boost the difficulty of the attack by increasing the number of candidate paths for the credential data.

credential data from many possible different combinations of candidate fields within three attempts. If the number of consecutive failed attempts from a device is greater than three, automatic login attempts from the same device could be ignored or prohibited.

BLOCKING ROOTED DEVICES

To perform a user credential cloning attack, a culprit's malware needs to access the target app's files that store the user credential data in the application. The Android platform, however, implements a sandboxing mechanism to negate such risks by protecting files owned by one app from other applications. Therefore, the constraint in the sandbox must first be removed or bypassed to execute a user credential cloning attack.

Rooting is the simplest way to bypass the Android platform's sandbox mechanism for performing user credential cloning attacks. On a rooted device, the attacker's application can simply attain the root access with a single line of code, e.g., `Runtime.exec("su")`. Thus, our first line of defense should be to use rooting prevention mechanisms, although it is questionable whether existing antirooting solutions are effective in practice [16].

USING NOTIFICATION MESSAGES FOR AUTOMATIC LOGIN

We can simply deploy a warning notification system for automatic login at the server side. That is, whenever an automatic login attempt is processed, the application server can send a simple push notification to the user's application to verify that the automatic login function was invoked by the rightful person or used illegally by someone else, even when the rightful person is not using that application. If the rightful person is not aware of the automatic login attempt, he or she can take an action immediately to report this suspicious login activity to the application server.

We do not claim that this is a perfect solution that can completely thwart user cloning attacks. But this strategy may likely be effective for users with a high awareness of security issues, without incurring a significant implementation cost.

ETHICAL CONSIDERATIONS

The main goal of our experiment was not to damage Starbucks and MOCI's business or reputation. As can be seen from the previous sections, we simply aimed to identify security risks associated with the automatic login feature on Android devices and recommend practical mitigation solutions to make it difficult for attackers to perform effective user credential cloning attacks.

RELATED WORK

The automatic login feature widely employed on Android [17] can allow an attacker to steal user credential information and impersonate a victim [18]. By default, Android protects an application's private data, such as user credentials, using a sandbox policy. However, malware having the root privilege can access those data. According to an official report from Google [8], 5.6% of all Android devices were rooted. Furthermore, Android devices can be forcibly rooted through a security vulnerability [9], [16] by such means as a rootkit [19]. In theory, a secure storage facility can be used to prevent theft of user credential information, even from an attacker who has the root privilege. In practice, however, it is not easy to equip mobile devices with a secure storage facility. Hardware-based data protection would be ideal, but we cannot mandate that everyone use mobile devices that are furnished with a hardware cryptochip, e.g., ARM TrustZone [20] or Trusted Platform Module [21].

The Open Web Application Security Project (OWASP) introduced this insecure storage issue as one of the top ten mobile security risks [22]. King [23] showed that user credentials stored in the Android app FourGoat, which was developed by OWASP for educating developers and testers about Android security, can be easily obtained by anyone with physical access to the Android device.

Recently, Choi et al. [24] detailed the feasibility of user credential cloning attacks against social networking applications, such as Google Account and Facebook, in the Android platform. Park et al. [25] also showed that credential information can be extracted from instant messenger applications, even though the apps deployed several defense mechanisms, such as code signing and device authentication. All of those attacks required the installation of a malicious application with the root permission. We extend their work into a more generalized attack model for developing the automatic user credential cloning attack framework; while previous studies [24], [25] focused on specific applications through reverse-engineering them, we present a generic procedure for user credential cloning attacks that comprises four main steps (see the "User Credential Cloning Attack" section).

CONCLUSION

It is a cumbersome task for smartphone users to type their passwords on smartphones with the small keyboards provided. Therefore, people frequently accomplish the login procedure for Android applications by exploiting the option of using automatic login via user credentials stored in these applications rather than by physically keying in user passwords. However, this feature could be dangerous. According to our observations and analysis, real-world applications from the Google Play Store are potentially exposed to user credential cloning attacks.

We demonstrated such an attack's feasibility through case studies involving two Android applications (Starbucks and MOCI) by analyzing the resource changes after using the automatic login option. In those applications, we can simply perform a user credential cloning attack by replacing some files in a user's device with those in another user; repacking and installing the application was not required. Furthermore, we discussed five possible defense strategies to mitigate such user credential cloning attacks.

In future work, we plan to investigate the feasibility of user credential cloning attacks on a large sample of Android applications. In addition, we contemplate developing a fully automated implementation of user credential cloning attacks for cracking most Android applications.

ACKNOWLEDGMENTS

This work was supported by the Defense Acquisition Program Administration and the Agency for Defense Development under the grant UD060048AD. We would like to thank all of the anonymous reviewers for their valuable feedback.

ABOUT THE AUTHORS

Junsung Cho (js.cho@skku.edu) earned his B.S. degree from the Department of Computer Engineering, Korea University of Technology and Education, Chun-an, South Korea, in 2014. He is currently a graduate student with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea, supervised by Hyoungshick Kim. His current research interests include usable security, mobile security, and security engineering.

Dayeon Kim (dykim7796@skku.edu) earned her B.S. degree from the Department of Computer Engineering, Korea University of Technology and Education, Chun-an, South Korea, in 2015. She is currently a graduate student with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea, supervised by Dong Ryul Shin. Her current research interests include computer networks, mobile security, and data mining.

Hyoungshick Kim (hyoung@skku.edu) earned his B.S. degree from the Department of Information Engineering, Sungkyunkwan University, Suwon, South Korea; his M.S. degree from the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, South Korea; and his Ph.D. degree from the Computer Laboratory, Cambridge University, United Kingdom, in 1999, 2001, and 2012, respectively. He is currently an assistant professor with the Department of Software, Sungkyunkwan University. His current research interests include usable security and security engineering.

REFERENCES

[1] D. Endler, "The evolution of cross site scripting attacks," iDEFENSE Labs, Chantilly, VA, Tech. Rep., 2002.

[2] E. von Zezschwitz, A. De Luca, and H. Hussmann, "Honey, I shrunk the keys: Influences of mobile devices on password composition and authentication performance," in *Proc. 8th Nordic Conf. Human–Computer Interaction: Fun, Fast, Foundational*, 2014, pp. 461–470.

[3] D. DeFigueiredo, "The case for mobile two-factor authentication," *IEEE Security Privacy*, vol. 9, no. 5, pp. 81–85, 2011.

[4] S. Grzonkowski, A. Mosquera, L. Aouad, and D. Morss, "Smartphone security: An overview of emerging threats," *IEEE Consum. Electron. Mag.*, vol. 3, no. 4, pp. 40–44, 2014.

[5] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party Android marketplaces," in *Proc. 2nd ACM Conf. Data and Application Security and Privacy*, 2012, pp. 317–326.

[6] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: Scalable and accurate zero-day Android malware detection," in *Proc. 10th Int. Conf. Mobile Systems, Applications, and Services*, 2012, pp. 281–294.

[7] Infosecurity Group. (2011, Mar. 10). 260,000 Android users infected with malware. [Online]. Available: https://www.infosecurity-magazine.com/news/260000-android-users-infected-with-malware/

[8] Android. (2017, Mar.). Android security 2016 year in review. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final.pdf

[9] H. Zhang, D. She, and Z. Qian, "Android root and its providers: A double-edged sword," in *Proc. 22nd ACM SIGSAC Conf. Computer and Communications Security*, 2015, pp. 1093–1104.

[10] F. Howarth. (2015, Oct. 12). Is rooting your phone safe? The security risks of rooting devices. *Insights*. [Online]. Available: https://insights.samsung.com/2015/10/12/is-rooting-your-phone-safe-the-security-risks-of-rooting-devices

[11] A. Boxall. (2015, Apr. 16). 80% of Android phone owners in China have rooted their device. *Business of Apps*. [Online]. Available: http://www.businessofapps.com/80-android-phone-owners-china-rooted-device

[12] Android Developers. Storage Options. [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html

[13] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proc. 10th ACM Conf. Computer and Communications Security*, 2003, pp. 290–299.

[14] M. N. Gagnon, S. Taylor, and A. K. Ghosh, "Software protection through anti-debugging," *IEEE Security Privacy*, vol. 5, no. 3, pp. 82–84, 2007.

[15] S. Schrittwieser and S. Katzenbeisser, "Code obfuscation against static and dynamic reverse engineering," in *Proc. 13th Int. Conf. Information Hiding*, 2011, pp. 270–284.

[16] S. Sun, A. Cuadros, and K. Beznosov, "Android rooting: Methods, detection, and evasion," in *Proc. 5th Workshop Security and Privacy Smartphones and Mobile Devices*, 2015, pp. 3–14.

[17] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proc. 2nd Symp. Usable Privacy and Security*, 2006, pp. 44–55.

[18] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android security," *IEEE Security Privacy*, vol. 7, no. 1, pp. 50–57, 2009.

[19] W. Song, H. Choi, J. Kim, E. Kim, Y. Kim, and J. Kim, "Pikit: A new kernel-independent processor-interconnect rootkit," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 37–51.

[20] ARM Security Technology, "Building a secure system using TrustZone technology," ARM, Cambridge, England, Tech. Rep. PRD29-GENC-009492C, 2009.

[21] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn, *A Practical Guide to Trusted Computing*. Indianapolis, IN: IBM Press, 2007.

[22] Open Web Application Security Project. (2016, Mar. 31). Mobile top 10 2014-M2. [Online]. Available: https://www.owasp.org/index.php/Mobile_ Top_ 10_ 2014-M2

[23] J. King, "Android application security with OWASP mobile top 10 2014," Master's thesis, Lulea Univ. Technology, Sweden, 2014.

[24] J. Choi, H. Cho, and J. H. Yi, "Personal information leaks with automatic login in mobile social network services," *Entropy*, vol. 17, no. 6, pp. 3947–3962, 2015.

[25] S. Park, C. Seo, and J. H. Yi, "Cyber threats to mobile messenger apps from identity cloning," *Intell. Automation Soft Comput.*, vol. 22, no. 3, pp. 379–387, 2015.