Open Sesame! Web Authentication Cracking via Mobile App Analysis

Hui Liu(✉), Yuanyuan Zhang, Juanru Li, Hui Wang, and Dawu Gu

Computer Science and Engineering Department,
Shanghai Jiao Tong University, Shanghai, China
hui.liu803@gmail.com

Abstract. Web authentication security can be undermined by flawed mobile web implementations. Mobile web implementations may use less secure transport channel and enforce less strict brute-force-proof measures, making web authentication services vulnerable to typical attacks such as password cracking. This paper presents an in-depth penetration testing based on a comprehensive dynamic app analysis focusing on vulnerable authentication implementations of Android apps. An analysis of Top 200 apps from China Android Market and Top 100 apps from Google Play Market is conducted. The result shows that 71.3 % apps we analyze fails to protect users' password appropriately. And an experiment carried out among 20 volunteers indicates that 84.4 % passwords can be cracked with the knowledge of password transformation process.

Keywords: Android apps · Web authentication · Password cracking

1 Introduction

As the prevailing of mobile smart devices, entry of web authentication is migrating from browsers to mobile apps. Many apps, however, do not implement secure authentication process and thus are vulnerable to typical web attacks such as password cracking. Therefore, the security of mobile web is significantly weakened compared to that of traditional web. With millions of apps released nowadays, it is essential to shed light on the status quo of how mobile web authentication processes are handled by those apps.

Previous studies has revealed many aspects of vulnerable implementation in Android apps that affect remote web authentication, especially on cryptographic misuses [2,4] and insecurity of user and session authentication [3]. However, they either rely on manual reverse engineering or only concern about simple vulnerabilities such as using hard-coded key. In this paper, we mainly consider two general types of web authentication vulnerabilities exposed by mobile apps concerning mobile web authentication implementations.

One is the transport channel downgrade vulnerability. Services serving for both mobile and web apps are most likely to use the same identity database, since maintaining separate databases is profitless and resource-consuming. But they

© Springer International Publishing Switzerland 2016
F. Li et al. (Eds.): APWeb 2016, Part II, LNCS 9932, pp. 483–487, 2016.
DOI: 10.1007/978-3-319-45817-5_51

would offer different interfaces when dealing with users from different platforms, out of the consideration of optimizing user experience and usability. That is, the **authenticator**[1] sent to server is the same for both mobile and web apps, while the transport channel of the authenticator can be diverse. The less secure channel that mobile app uses can result in the authenticator leakage, which opens a door for attackers to attack the web authentication. Web authentication security is completely compromised if the transformation process is weak enough such that an attacker can reveal the corresponding password simply knowing the authenticator. If the password can not be deduced from the authenticator at once, with the knowledge of the transformation process obtained by app analysis, an attacker can still launch an off-line password brute-forcing attack, trying all possible passwords until the output of the transformation process matches the objective authenticator, which manifests the correct password.

The other vulnerability is the CAPTCHA bypass defect. When an attacker has no access to a user's authenticator, she may actively launch a password guessing attack, pretending to be the user and trying possible passwords until getting a correct response from server. Since this is a well-known on-line attack, a countermeasure involving CAPTCHAs is proposed. Nevertheless, for mobile apps, the enforcement of this measure can be omitted, which makes this kind of attack possible again. With the knowledge of the transformation process, the corresponding authenticators that may be accepted by the server are successfully generated.

2 Inferring Authentication Process via App Analysis

As described above, the key factor to successfully launch both the off-line and on-line brute-forcing/password guessing attack is the ability of transforming the guessed passwords to authenticators. If the generated authenticator matches the objective, the password is cracked. In this section, we introduce the way how to get a sketch of the authentication process via app analysis.

We need to analyze apps to get an overview of what apps have done to the password. And we only concern about the authentication process part of an app. Therefore, our general idea of analyzing an app is (1) running the sign-in process, (2) logging method call traces and specified parameters, and (3) getting the sketch of the transformation process by mapping the input-output relationship and profiling critical method calls. Figure 1 depicts the analysis process. We use appium [1] to automatically run the app's login process, filling in the pre-chosen username and password. Then an instrumented Dalvik VM on which the app is running is used to log down the method call trace and specified method parameters. At last, we extract these logs and abstract the semantic information of the transformation process.

[1] During user authentication, an app typically receives a password, encodes or encrypts it, and sends the result together with other data to a remote web server. We let authenticator denote the result for the rest of this paper.

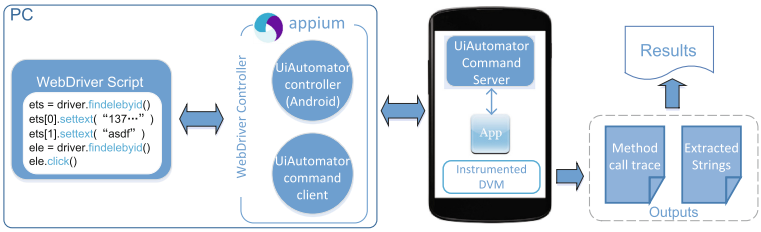

Fig. 1. The process of app analysis.

3 Testing Results

Using the method introduced in Sect. 2, we analyzed 100 top free apps from Google play and 200 top free apps from China Android Market. We successfully analyze 209 apps[2] and classify them into four categories and nine types according to the type of transformation process, as listed in Table 1. The category basically matches discoveries found in [3].

Table 1. Summary of the analysis results.

Type of Transformation Process		China Android Market			Google Play Market		
		HTTP	HTTPS[a]	HTTPS[b]	HTTP	HTTPS[a]	HTTPS[b]
Trivial Transformation	Plaintext	**44**	**19**	8	**1**	**8**	42
	Encoding	**2**	0	0	0	**1**	0
Hash	One-time MD5	**21**	**8**	5	0	0	0
	Fixed-salt MD5	**6**	0	0	0	0	0
	Multi-time hash	**3**	0	0	0	0	0
Symmetric Encryption	AES/DES with hard-coded key	**24**	**7**	2	0	0	0
	AES/DES with randomly generated key	**1**	0	0	0	0	0
Asymmetric Encryption	RSA/ECB/NoPadding	**3**	**2**	0	0	0	0
	RSA/ECB/PKCS1Padding	2	0	0	0	0	0
	Sum	106	36	15	1	9	42

[a]apps using HTTPS connection and inadequately validating the certificates.
[b]apps using HTTPS connection and correctly validating the certificates

Overall, 149 apps (71.3 %) fail to provide secure user authentication. Nearly all apps from Google Play use the same type of authentication process, while apps from China Android Markets use different types of transformation process.

[2] Other apps are either packed or involved with native APIs, in which case manual intervention is needed.

And for those apps using HTTPS, 45 out of 92 apps (48.9 %) fail to validate the certificate correctly, resulting apps vulnerable to active attackers.

The distribution of transformation process type differs between apps from Google play and China Markets, and apps from China Markets are exposed to more security threats than ones from Google Play. Apps from Google Play seem to obey the same specification. They send password directly to the server. The security totally relies on a secure channel, mostly implemented as HTTPS connections. However, apps from China Markets are completely different. On one hand, most apps send the authentication material through HTTP. On the other hand, apps try to protect password by making transformations (security by obscurity), which, as we will show in Sect. 4, basically enforce no security.

Case Study. *DangDang* is the Chinese most popular online bookstore. While its web entry sends user's password in plain text via a secure TLS channel, its Android client sends the authentication information without any protection.

In terms of CAPTCHA Bypass, on the website of *Meilishuo*, which is the largest fast fashion e-commerce platform in China, a group of pictures are presented for users to click and rotate each of them until all the pictures are in the right position when a user logs in. This is a pretty strong approach to defend against password guessing attacks. Nevertheless, the entry that its mobile app provide totally breaks this defense. It neither presents any CAPTCHAs nor limits the times a user can submit the authentication information, making the online password guessing attack unimpeded. This weakness also appears in *Sogou*, *Jiayuan*, *Mia* etc.

4 Password Cracking Cost Measurement

To intuitively demonstrate the consequences of password cracking enabled by the knowledge of password transformation process, we recruit 20 volunteers to randomly use those 209 apps and launch the authentication process in a controlled network environment. We collect 180 authenticators and conduct the password cracking attack.

The attack cost of some transformation type is trivial, such as encoding and symmetric encryption, since password can be directly calculated from authenticator by inversing the transformation process. As for hash and asymmetric encryption type, cracking is needed. We let two GPU cards (Telsa K20c, 4800 MB global memory, 2496 CUDA cores, 706 MHz clock rate; Quadro K620, 2047 MB global memory, 384 CUDA cores, 1124 MHz clock rate) work together to crack hash. And we use an Intel Xeon Processor E5-2643v3 with 20M smart cache and 3.4 GHz base frequency CPU to run the RSA calculation. Finally, we successfully recover 84.4 % of the passwords in 2 days.

Acknowledgments. This work is supported by the Major program of Shanghai Science and Technology Commission (15511103002).

References

1. Appium automation for apps. http://appium.io/. Accessed 20 Apr 2016
2. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A., Shastry, B.: Towards taming privilege-escalation attacks on android. In: NDSS (2012)
3. Cai, F., Hao, C., Yuanyi, W., Yuan, Z.: Appcracker: widespread vulnerabilities in user and session authentication in mobile apps. In: IEEE Mobile Security Technologies. IEEE (2015)
4. Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: Why eve, mallory love android: an analysis of android ssl (in) security. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 50–61. ACM (2012)