Security Analysis of Google Authenticator, Microsoft Authenticator, and Authy

Aleck Nash[1], Hudan Studiawan[2], George Grispos[3], and Kim-Kwang Raymond Choo[1(✉)]

[1] Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, USA
aleck.nash@my.utsa.edu, raymond.choo@fulbrightmail.org
[2] Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
hudan@its.ac.id
[3] School of Interdisciplinary Informatics, University of Nebraska-Omaha, Omaha, USA
ggrispos@unomaha.edu

Abstract. As the use of authenticator applications for two-factor authentication (2FA) has become increasingly common, there is a growing need to assess the security of these applications. In this paper, we present a security analysis of authenticator applications that are widely used on various platforms, such as Google Authenticator, Microsoft Authenticator, and Authy. Our analysis includes an examination of the security features of these applications (e.g., level of protection) as well as the communication protocols used between the applications and the servers. Our results show that these applications have significant vulnerabilities that could compromise the security of the authentication process. Specifically, we found that some authenticator applications store sensitive data, such as secret keys, in plain text, making them vulnerable to attacks. Overall, our findings indicate that there is a need for better security practices in the design and implementation of authenticator applications. We recommend that developers follow best practices for secure coding and use well-established cryptographic algorithms to generate one-time codes.

Keywords: Security analysis · Man-in-the-middle (MITM) attack · Authenticator applications · Authentication protocols

1 Introduction

With the increasing use of digital platforms and services, the security of user accounts has become a critical concern. One way to enhance the security of user

S. Goel and P. R. Nunes de Souza (Eds.): ICDF2C 2023, LNICST 571, pp. 197–206, 2024.
https://doi.org/10.1007/978-3-031-56583-0_13

accounts is by implementing two-factor authentication (2FA), which requires users to provide two forms of authentication, typically a password and a one-time code, to access their accounts. Authenticator applications, which generate one-time codes on users' devices, have become a popular form of 2FA due to their convenience and ease of use [1].

While studies such as [12] identified a number of factors that may lead to poor acceptance of authenticator applications, one cannot deny that authenticator application usage has become more widespread. This reinforces the importance of evaluating the security of these applications since compromised applications can allow attackers to bypass 2FA and gain access to sensitive user data.

Common attacks include man-in-the-middle (MITM), where an MITM attacker seeks to intercept data exchanged between endpoints and consequently compromise the data's confidentiality and integrity [3]. During a MITM attack, a typical situation includes: two endpoints (targets), and an intervening party (the attacker). The attacker gains control over the communication channel connecting the two endpoints and possesses the ability to listen or alter their exchanged messages. This is a common capability considered in many adversary and threat models, such as the Dolev-Yao model and those used in security proofs [5,14,17]. In addition to vulnerabilities in the 2FA protocols, other vulnerabilities (e.g., implementation-related vulnerabilites) may also be exploited [9,15]. For example, SSL vulnerabilities in Android applications can lead to information leakage and the transmission of sensitive data unencrypted. This can put users' personal information at risk, such as login credentials, financial data, and other sensitive information. SSL vulnerabilities can also be exploited by attackers to intercept and manipulate network traffic, leading to potential security breaches. Therefore, it is important for developers to address these vulnerabilities and for users to be aware of the risks associated with using applications that have SSL vulnerabilities [16].

Attacks on SSL/TLS protocols may take many different forms, such as session hijacking, version degradation, Heartbleed, and Berserk [11]. Attacks on the server end and attacks in transit may be divided into two groups. Side channel attacks, manipulation attacks, and certificate manipulation are examples of assaults at the server end. Cipher-based and compression-based attacks are both types of transit attacks. Furthermore, SSL stripping is a frequent attack that takes advantage of the flaw of many open ports for the same application server [11]. A taxonomy for MITM attacks against HTTPS is provided in another study [18], which contains a variety of these attacks' attributes. The capacity to intercept and alter encrypted communication, the application of fake digital certificates to pretend to be trustworthy websites, and the abuse of holes in the SSL/TLS protocols used by HTTPS are some features of MITM attacks against HTTPS that are often seen. However, designing secure 2FA protocols is not an easy feat, as partly demonstrated by the number of 'break-and-fix' attempts [9,19]. For example, the authors of [14] identified weaknesses in another previously proposed 2FA protocol for the Internet of Things (IoT) and proposed a fix to addressing the weaknesses.

In this paper, we present a security analysis of authenticator applications that are widely used in several platforms, namely: Google Authenticator[1], Microsoft Authenticator[2], and Authy[3]. Our analysis aims to identify vulnerabilities and weaknesses in the security features and protocols of these applications and provide recommendations for improving their security. We mainly use the MITM technique to perform the analysis.

The rest of this paper is organized as follows. Section 2 describes the methodology used in our analysis. Section 3 presents the results of our security analysis, including an assessment of the security features and protocols of the authenticator applications. Finally, Sect. 4 concludes the paper.

2 Research Method

As previously discussed, a MITM attacker generally seeks to intercept the communication between two parties, and in our context, an authenticator application (client) and a web application (server). However in this study, we act as a security researcher who can then eavesdrop on the data in the communication, as shown in Fig. 1.

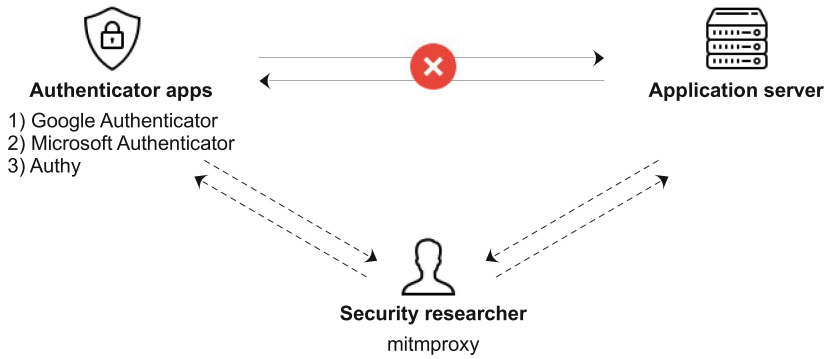

Fig. 1. Methodology to analyze authenticator applications

For the security analysis, we set the three authentication applications as the 2FA method for the service applications shown in Table 1. Then, we simulated a man-in-the-middle (MITM) attack by utilizing a proxy to capture the traffic generated by the authentication app and service applications such as Facebook and Twitch. We used Mitmproxy [4], which is an open-source Java-based SSL proxy that acts as an intermediary between the server and end-users, such as computers, nameservers, and mobile devices. It acts as an HTTP and HTTPS proxy and records the traffic and requests being made.

[1] https://support.google.com/accounts/answer/1066447.
[2] https://www.microsoft.com/en-us/security/mobile-authenticator-app.
[3] https://authy.com/.

Mitmproxy was installed and configured on all laptops and mobile devices mentioned in Table 1. We then started the data generation phase by login into each application using Google Authenticator, which was configured as the authentication method across the applications in Table 1. Then the data was obtained from Mitmproxy for security analysis. The experiment was repeated after configuring Microsoft Authenticator and then Authy as the authentication method across the applications.

In analyzing the data, we set the following considerations as guidelines to evaluate the security of these applications and services:

1. What level of protection do these authentication applications offer?
2. What type of standard authorization protocol is implemented by the applications? Are there any weaknesses associated with it?
3. Are sensitive data still appearing in the traffic when performing MITM attacks?
4. How can an attacker use the sensitive data to log in to the server as the user?

3 Results and Analysis

In this section, we present the results of the study and highlight the key findings and their significance. This section showcases our contribution to the field by providing valuable insights into the security of authentication applications and recommendations to address vulnerabilities found.

3.1 Experimental Results

From Table 1, we were able to obtain the authentication tokens for all applications. This means that, during a MITM attack, the attacker can obtain the authentication token and use it to log in to the server as the original user. We are going to demonstrate exactly how that can be done later in this paper. We also observed that some applications, namely Robinhood, use poor methods to exchange credentials for access tokens, as shown in Fig. 2a. Robinhood uses the password grant type, in which the client application collects the user's password and sends it to the authorization server. This method allowed us to obtain the password in plain text, without any encryption. This method is not recommended at all.

In fact, the latest OAuth 2.0 Security Best Current Practice specification recommends against using this grant entirely, and it is also being removed in the OAuth 2.1 update. Although we were able to see that Robinhood uses some good security features, such SHA348 to hash SRI (Subresource Integrity). However, the user password was left without encryption. Contrary to Robinhood, we found that Twitch encrypted the password with RSA-OAEP (Optimal Asymmetric Encryption Padding) as depicted in Fig. 2b. Twitch uses Persisted Query, which is a query string received on the server side with a unique identifier (SHA-256 hash) as shown in Fig. 2c. This identifier can be sent by the client instead of the corresponding query string, thereby reducing the request size dramatically.

Fig. 2. Robinhood unencrypted credentials and Twitch encryption

We also observed that both the signature method and the digest method algorithms are based on SHA256 in Outlook, as Microsoft recommends SHA256 or better over weaker algorithms such as SHA1, as shown in Fig. 3a. For Facebook, Slack, and Discord, the authentication passcodes generated by the 2FA app were found in plaintext, which can come in handy for the attacker (Fig. 3b).

For most of the applications, we were able to detect the operating system used, but not the version, as shown in this parameter: *sec-ch-ua-platform: "Windows"*. The traffic from Outlook included user agent information such as the 2FA App and version, and OS type and version, and CFNetwork/Darwin and versions, along with client information such as email address and ePCT (Fig. 3c).

Additionally, we were able to obtain the Client ID, Client Secret, Access Token, and Refresh Token for the Gmail application, as shown in Fig. 4a. This allows not only to log in to the server using the access token but also to use the refresh token to generate new access tokens indefinitely as long as the token is not expired or revoked. For this reason, Gmail seems to be the least secure application when MITM attacks are performed (Fig. 4b).

```
<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#
                                hmac-sha256">
</SignatureMethod>
<Reference URI="#EncPsf">
  <Transforms>
    <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
    </Transform>
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
  </DigestMethod>
  <DigestValue>6vWvweaFM/kJWFVyCka35b+CAoyw5/BzyfCPrUp1QBA=
  </DigestValue>
</Reference>
```

(a) Outlook signature method

2fa_magiclogin: z-app-2943702915378-4002966658259-truncated-here
remember: 1
has_remember: true
2fa_code: 485965
2fa_action: submit_primary

(b) Slack authentication passcode in plain text

client_id: 81feaced-5ddd-41e7-8bef-3e20a2689bb7
redirect_uri: https://account.microsoft.com/auth/complete-signin-oauth
response_type: code
login_hint: authenticateme6@outlook.com
x-client-SKU: MSAL.Desktop
x-client-Ver: 4.45.0.0
uaid: 25c8caf7849b4aa9a3af263e5be12836
issuer: mso
ui_locales: en-US
epct: AQABAAAAAAD--truncated-here

(c) Outlook captured information

Fig. 3. Outlook signature method, Slack plain passcode, and Outlook captured information

It is important to mention that the reason we were able to obtain the authentication access tokens for these applications was that certificate pinning was not implemented. Certificate pinning is an online application security technique designed to thwart MITM attacks, by accepting only authorized certificates for the authentication of client-server connections. This technique makes it extremely difficult for cybercriminals to utilize fraudulent certificates to gain access to applications through MITM, compromise certificate authorities, and issue invalid certificates.

If a single certificate is compromised, an attacker can produce a valid certificate for any hostname by signing it with the compromised CA root certificate, which enables the attacker to MITM any TLS connection [10]. To manage these issues, Evans et al. [6] suggested certificate pinning technique which can be implemented in two ways:

1. Leaf certificate (public key): pins the server's specific public key certificate. This is usually achieved by hard coding its fingerprint. An alternative way is to just pin the server's public key.

Table 1. Summary of results

Application	2FA App	Attributes						Potential Risk
		ID	Secret	Auth token	Refresh token	OS	App info	
Facebook	Google	No	No	Yes	No	Yes	Yes	Passcode is in plaintext
	Microsoft	No	No	Yes	No	Yes	Yes	
	Authy	No	No	Yes	No	Yes	Yes	
Twitch	Google	Yes	No	Yes	No	Yes	Yes	The token can be reused;
	Microsoft	Yes	No	Yes	No	Yes	Yes	Passcode is in plaintext
	Authy	Yes	No	Yes	No	Yes	Yes	
Dropbox	Google	No	No	Yes	No	Yes	Yes	The token can be reused
	Microsoft	No	No	Yes	No	Yes	Yes	
	Authy	No	No	Yes	No	Yes	Yes	
Slack	Google	Yes	No	Yes	No	Yes	Yes	Passcode is in plaintext
	Microsoft	Yes	No	Yes	No	Yes	Yes	
	Authy	Yes	No	Yes	No	Yes	Yes	
Discord	Google	No	No	Yes	No	No	No	Passcode is in plaintext
	Microsoft	No	No	Yes	No	No	No	
	Authy	No	No	Yes	No	No	No	
Robinhood	Google	Yes	No	Yes	No	Yes	Yes	Password is in plaintext
	Microsoft	Yes	No	Yes	No	Yes	Yes	
	Authy	Yes	No	Yes	No	Yes	Yes	
Gmail	Google	Yes	Yes	Yes	Yes	Yes	No	The token can be reused; Refresh token can be used to generate new tokens
Outlook	Microsoft	Yes	No	Yes	No	Yes	Yes	The token can be reused

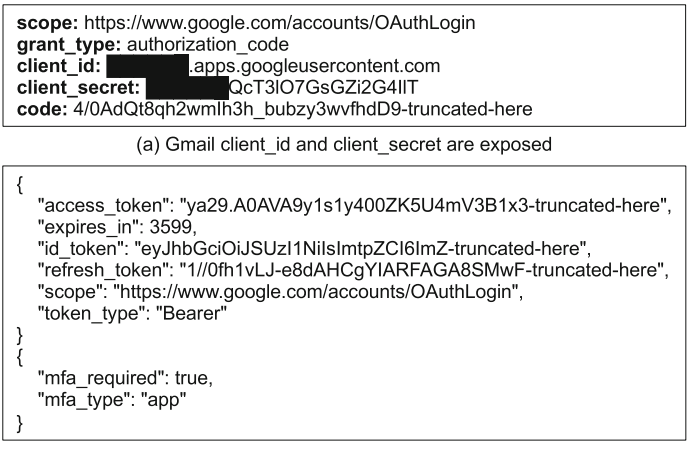

Fig. 4. Gmail client_id, client_secret, and refresh token

2. Root certificate (intermediary): pins a specific root CA or intermediary certificate. Then, the server can renew its leaf certificate as needed as long as it is signed by the intermediary certificate or the pinned root [10].

A number of studies show that applications with incorrect application of SSL pinning and certificate pinning are at risk of MITM attacks. For example, Fahl

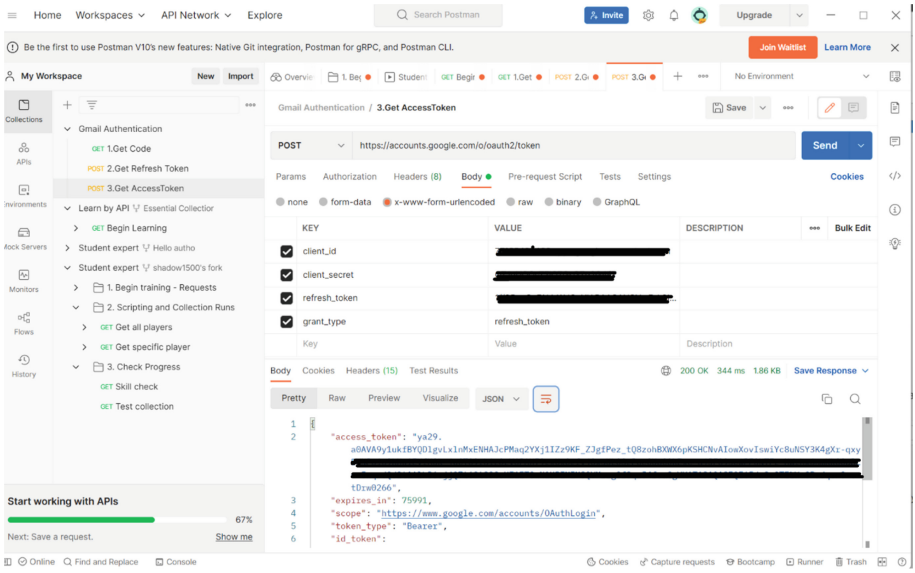

Fig. 5. Login interaction with Postman

et al. [7] performed an investigation of Android applications and found that many of them allowed self-signed certificates, did not validate the hostname, and did not encrypt certain connections at all. According to their analysis, more than 1000 applications out of 13,500 had an incorrect implementation of the validation methods, making them prone to a MITM attack. This also suggests that application developers are unable to sufficiently deal with certificate validation in general and with certificate pinning in particular [13].

Another research showed that the situation is getting worse over time rather than improving [2]. Interviews with application developers with broken validation indicate that developers do not fully completely understand the security consequences of such poor implementation [8]. Developers often ignore MITM attacks as a threat, and they prefer to use self-signed certificates as it is more convenient for them [13]. Certificate pinning stands out as the most recommended protection measure among all available advanced measures. With correct implementation, certificate pinning minimizes the risk of MITM attacks [2].

3.2 Using the Sensitive Data to Log into the Server

To demonstrate the sensitivity of the obtained data, we used Postman[4] to interact with the login servers. Postman is an API platform for developers to build, test, and iterate their APIs. It can be used to automate API calls and API testing. For our purpose, we used the information we obtained from the Gmail

[4] https://www.postman.com/.

application through Mitmproxy (such as client ID, client secret, and refresh token) to generate a new access token from `gmail.googleapis.com` as shown in Fig. 5.

We can see the status 200 OK which means that the request was correct, and the desired response has been sent to the client. We received a new access token which can be regenerated indefinitely unless the token is expired or revoked. The attacker can use the access token to retrieve messages from a specific email, using the GET function request from `https://gmail.googleapis.com/gmail/v1/users/{{USER_ID}}/messages`. However, the former request is sufficient for our demonstration purposes. We repeated this experiment multiple times, and we were able to obtain a new access token in every request. This means that once the refresh token is captured by an attacker, it can be used to generate new access tokens to maintain access to the victim's account.

4 Conclusion and Future Work

In this paper, we have conducted a thorough security analysis of authenticator applications, specifically Google Authenticator, Microsoft Authenticator, and Authy. We have identified vulnerabilities that can be exploited by attackers to compromise the security of these applications. Our findings reveal that the security of authenticator applications is not foolproof and requires constant attention and updates to prevent attacks. We recommend that developers and users of authenticator applications implement more secure protocols to enhance the security of their applications.

Overall, this paper highlighted the importance of security in authentication applications and the need for continuous research to stay ahead of potential threats. We hope that our analysis will contribute to the development of more secure authenticator applications in the future.

References

1. Aloul, F., Zahidi, S., El-Hajj, W.: Two factor authentication using mobile phones. In: 2009 IEEE/ACS International Conference on Computer Systems and Applications, pp. 641–644 (2009)
2. Buhov, D., Huber, M., Merzdovnik, G., Weippl, E.: Pin it! improving android network security at runtime. In: 2016 IFIP Networking Conference (IFIP Networking) and Workshops, pp. 297–305 (2016)
3. Conti, M., Dragoni, N., Lesyk, V.: A survey of man in the middle attacks. IEEE Commun. Surv. Tutorials **18**(3), 2027–2051 (2016)
4. Cortesi, A., Hils, M., Kriechbaumer, T., contributors: mitmproxy: a free and open source interactive HTTPS proxy (2010). https://mitmproxy.org/ [Version 9.0]
5. Do, Q., Martini, B., Choo, K.R.: The role of the adversary model in applied security research. Comput. Secur. **81**, 156–181 (2019)
6. Evans, C., Palmer, C., Sleevi, R.: RFC 7469: Public key pinning extension for HTTP (2015)

7. Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: Why eve and mallory love Android: an analysis of Android SSL (in) security. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 50–61 (2012)
8. Fahl, S., Harbach, M., Perl, H., Koetter, M., Smith, M.: Rethinking SSL development in an appified world. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 49–60 (2013)
9. Gavazzi, A., Williams, R., Kirda, E., Lu, L., King, A., Davis, A., Leek, T.: A study of multi-factor and risk-based authentication availability. In: 32nd USENIX Security Symposium, USENIX Security, pp. 1–18 (2023)
10. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: Validating SSL certificates in non-browser software. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 38–49 (2012)
11. Keerthi, V.K., et al.: Taxonomy of SSL/TLS attacks. Int. J. Comput. Netw. Inf. Secur. **8**(2), 15 (2016)
12. Marky, K., etal.: "nah, it's just annoying!" a deep dive into user perceptions of two-factor authentication. ACM Trans. Comput. Hum. Interact. **29**(5), 43:1–43:32 (2022)
13. Merzdovnik, G., Buhov, D., Voyiatzis, A.G., Weippl, E.R.: Notary-assisted certificate pinning for improved security of Android apps. In: 2016 11th International Conference on Availability, Reliability and Security (ARES), pp. 365–371 (2016)
14. Modarres, A.M.A., Sarbishaei, G.: An improved lightweight two-factor authentication protocol for IoT applications. IEEE Trans. Industr. Inf. **19**(5), 6588–6598 (2023)
15. Narayanan, A., Lee, K.: Security policy audits: why and how. IEEE Secur. Priv. **21**(2), 77–81 (2023)
16. Onwuzurike, L., De Cristofaro, E.: Danger is my middle name: experimenting with SSL vulnerabilities in android apps. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, pp. 1–6 (2015)
17. Peeters, C., Patton, C., Munyaka, I.N.S., Olszewski, D., Shrimpton, T., Traynor, P.: SMS OTP security (SOS): hardening SMS-based two factor authentication. In: ASIA CCS: ACM Asia Conference on Computer and Communications Security, pp. 2–16 (2022)
18. Stricot-Tarboton, S., Chaisiri, S., Ko, R.K.: Taxonomy of man-in-the-middle attacks on HTTPS. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 527–534 (2016)
19. Zhou, Z., Han, X., Chen, Z., Nan, Y., Li, J., Gu, D.: Simulation: demystifying (insecure) cellular network based one-tap authentication services. In: 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 534–546 (2022)