Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8819790, 14 pages
https://doi.org/10.1155/2020/8819790

Research Article

Authenticator Rebinding Attack of the UAF Protocol on Mobile Devices

Hui Li[1], **Xuesong Pan**[1], **Xinluo Wang**[1], **Haonan Feng**[1] and **Chengjie Shi**[2]

[1]*School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100195, China*

Correspondence should be addressed to Hui Li; lihuill@bupt.edu.cn

Received 24 April 2020; Revised 21 June 2020; Accepted 9 August 2020; Published 1 September 2020

Academic Editor: Ding Wang

We present a novel attack named "Authenticator Rebinding Attack," which aims at the Fast IDentity Online (FIDO) Universal Authentication Framework (UAF) protocol implemented on mobile devices. The presented Authenticator Rebinding Attack rebinds the victim's identity to the attacker's authenticator rather than the victim's authenticator being verified by the service in the UAF protocol, allowing the attacker to bypass the UAF protocol local authentication mechanism by imitating the victim to perform sensitive operations such as transfer and payment. The lack of effective authentication between entities in the implementations of the UAF protocol used in the actual system causes the vulnerability to the Authenticator Rebinding Attack. In this paper, we implement this attack on the Android platform and evaluate its implementability, where results show that the proposed attack is implementable in the actual system and Android applications using the UAF protocol are prone to such attack. We also discuss the possible countermeasures against the threats posed by Authenticator Rebinding Attack for different stakeholders implementing UAF on the Android platform.

1. Introduction

FIDO UAF is an authentication mechanism based on public key cryptography designed for replacing password-based authentication [1], which has been criticized for its inconvenience and insecurity because it requires users and verifiers to maintain a growing list of login credentials as well as passwords. With FIDO UAF, users can first register their devices installed with a FIDO UAF stack to the online service by selecting a local authentication mechanism such as fingerprint and face recognition; then, users only need to repeat the local authentication operation instead of entering their passwords whenever they need to be authenticated by the service. Because of its convenience and security, UAF has attracted lots of attention in both the academic and industrial societies since its release. By April 2020, there have already been 436 certified FIDO UAF products in the market [2].

Recently, some researchers focus on analyzing the security of UAF and point out that FIDO UAF may face various potential security threats in the design and implementation of the protocol. Hu and Zhang formalize the UAF protocol and propose hypothetical attacks such as misbinding attack, parallel session attack, and multiuser attack [3], but they neither elaborate on the assumptions required to perform these attacks nor give the concrete implementation of these attacks. Xenakis et al. present an informal security analysis of the UAF protocol and identify a list of vulnerabilities that can cause attacks such as intercepting switching data, imitating the user's online service, and presenting false information to the user screen during the transaction [4]. However, they fail to provide any specific verification process for these attacks and ignore the actual factors when implementing the FIDO protocol, so some of the proposed attacks lack feasibility.

Most of the abovementioned FIDO UAF attacks are caused by the fact that the running environment of the UAF protocol can meet neither the UAF security assumptions described in the FIDO Security Reference [5] nor the requirements of the security standards provide by FIDO Certification [6] for FIDO products. Moreover, although FIDO UAF is widely used on mobile devices [2, 7], due to the openness and diversity of mobile devices, currently there is no

specific unified standard for the implementation of the UAF protocol on them, and certain FIDO UAF products cannot meet the UAF security assumptions, and their security levels are not suitable for actual scenarios. Our previous work [8] presents an attack for the implementation of the UAF protocol caused by the lack of a trusted display module on the mobile device, so the attacker may successfully tamper such displayed information as transaction data.

In consideration of the fact that Android is one of the most popular mobile operating systems and there are many certified providers of certified products on the Android platform [9, 10], we focus on analyzing the security of the UAF protocol implementation on mobile devices and propose a novel attack named "Authenticator Rebinding Attack". The proposed Authenticator Rebinding Attack rebinds the victim's identity to the attacker's authenticator and allows the attacker to impersonate the victim to perform sensitive operations such as transfer and payment.

To the best of our knowledge, our work is the first to study the threat of active Authenticator Rebinding Attack of the UAF protocol on the Android platform. On the one hand, we study the actual implementation of this attack according to the different modes in the UAF protocol on mobile devices. On the other hand, we point out that the reason for this attack is the lack of effective authentication between entities in the implementations of the UAF protocol used in the real world. We also evaluate the impact of this attack by analyzing 42 FIDO UAF applications and find that 19% of the applications that call third-party UAF Client Applications are unable to resist the attack, while the other 81% applications that implement the UAF protocol inside themselves might also suffer from this attack if they run in a compromised environment.

The contributions of this paper can be summarized as follows:

(i) We present a novel attack called Authenticator Rebinding Attack, which impersonates the victim to perform sensitive operations by rebinding the victim's identity to the attacker's authenticator

(ii) We demonstrate the technical feasibility of Authenticator Rebinding Attack by giving the details of the attack on the Hebao Pay and Jingdong Finance applications

(iii) We prove the practical significance of this attack by analyzing their security on the UAF applications mined from applications in the real world

(iv) We present the main causes of this threat and the countermeasures against this attack for different stakeholders on implementing the UAF protocol on the Android platform

The rest of this paper is organized as follows. In Section 2, we present the architecture, trust model, and operations of the UAF protocol. In Section 3, we analyze two UAF implementation modes, i.e., Out-App Authenticator Mode and In-App Authenticator Mode. In Section 4, we present the Authenticator Rebinding Attack under both the Out-App and In-App Authenticator Modes as well as verify such an attack on typical applications. In Section 5, we analyze the security of the actual applications using the UAF protocol to evaluate the implementability of the attack and present the main causes of such threat, as well as the countermeasures against the threat. In Section 6, we finally give our conclusions.

2. UAF Protocol

In this section, we introduce the architecture, trust model of the client side, and simplified operations on the Android platform of the UAF protocol.

2.1. Architecture. Figure 1 shows the architecture of the UAF protocol, which includes six entities—User Agent, UAF Client, UAF ASM, UAF Authenticator, Web Server, and UAF Server [11]. These entities are deployed on the User Device and the Relying Party. The User Device works as a client and interacts with the user, generates and stores the unique *Authentication Keys*, and computes and returns a response for the *challenge* from the server side. The Relying Party works as a server and initiates the challenge-response mechanism and verifies and stores the user credentials, e.g., unique *Authentication Public Keys*. The User Device and the Relying Party communicate with each other using a secure transport protocol (such as TLS/HTTPS [12]) established between the FIDO UAF Client and the Relying Party. Moreover, the internal communication between entities in the UAF protocol differs and depends on the protocol implementations [13].

The UAF Authenticator is the entity that can be inserted (such as a USB hardware device with PIN code protection) or embedded (such as a fingerprint sensor in a smartphone) into the User Device. On the Android platform, it is recommended to implement the UAF Authenticator as a module based on the TEE. The UAF Authenticator contains two kinds of asymmetric keys, a pair of *Attestation Keys* and several pairs of *Authentication Keys*. *Attestation Keys* are prestored in the UAF Authenticator and used in the registration operation. *Authentication Keys* are generated by the UAF Authenticator in the registration operation and used in the authentication operation.

The UAF ASM is a software interface between the UAF Client and the UAF Authenticator, which provides uniform API to the upper layer so that a UAF Client can support diverse UAF Authenticators with different biometric factors.

The UAF Client acts as the client of the UAF protocol. It interacts with diverse UAF Authenticators through the UAF ASM and UAF Server through a Relying Party. The User Agent interacts with the user and initiates the whole operation when the user enables biometric authentication.

On the Android platform, the UAF Client and the UAF ASM can be independent applications separated from the User Agent or built-in modules of the User Agent, which will be introduced in detail in Section 3. The Web Server provides the user application service and interacts with the UAF Server to transfer UAF protocol messages. The UAF Server

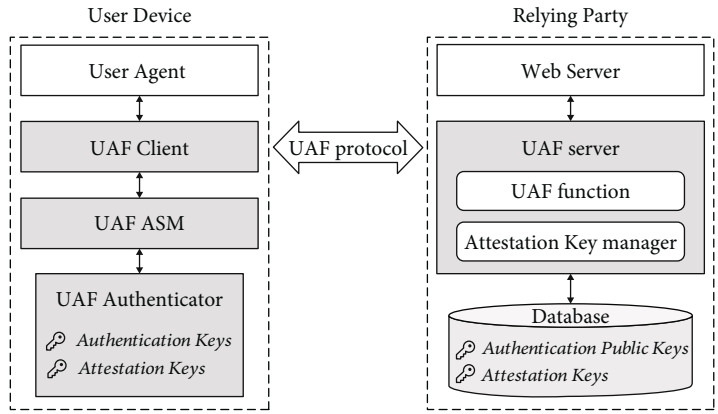

Figure 1: Architecture of the UAF protocol.

is responsible for communicating with the client, verifying the response message, and updating the public key related to the user. In the following section, we will use one server entity to represent the Web Server and the UAF Server to make the description more concise.

2.2. FIDO UAF Client Trust Model. We first introduce the FIDO UAF Client Trust Model described in FIDO UAF specification to show how these entities of the client side authenticate each other; then, we present why these authentication measures might not be effective when they are implemented on Android platform in Section 5.2.

The FIDO UAF Client Trust Model is shown in Figure 2 [14]. The FIDO UAF specification describes the data structures for authentication and access control between entities, in which *FacetID* is used for the UAF Client to authenticate the User Agent; *CallerID* is used for the UAF ASM to authenticate the UAF Client; *KHAccessToken* is used to provide access control for an *Authentication Key*. The UAF Authenticator ensures that a UAF ASM provides a specific *KHAccessToken* to access the correct user *Authentication Key*. The *KHAccessToken* is exported by the UAF ASM during the registration operation using data such as *AppID*, *PersonalID*, *ASMToken*, and *CallerID* [15]. If the *AppID* received by a UAF Client is a valid HTTPS URL, the UAF Client will obtain a trusted *FacetID* list by accessing the URL (HTTPS guarantees the list is trusted), check if the *FacetID* of the User Agent is in this list and then verify the validity of the User Agent. If the *AppID* is empty, the UAF Client directly sets the *FacetID* of the User Agent to the *AppID* field and the *FacetID* will be finally verified by the server [16]. Besides, the *AAID* (Authenticator Attestation ID) identifies a model, class, or batch of UAF Authenticators that share the same characteristics. The *AAID* also identifies a pair of *Attestation* (*Public*/*Private*) *Keys* [17].

According to our research, the ASM-Authenticator Applications of the same version and vendor have the same *AAID* and *Attestation Keys* on the Android platform. The *FacetID* is a URI derived from the Base64 encoding SHA-1 hash of the APK signing certificate of the User Agent by the UAF Client [16]. The *CallerID* of a UAF Client is derived by the UAF ASM in the same way [15].

2.3. UAF Protocol Operations. The UAF protocol has two critical operations, namely, registration and authentication [13]. As shown in Figure 3, in order to describe the FIDO UAF protocol more concisely, we depict the UAF protocol operations as a challenge-response process merged from the registration and authentication operations by omitting some details.

The server and the UAF Authenticator first successfully share necessary data such as the *Attestation Public Key*, *AAID*, and protocol policies through the process of FIDO Metadata Service before the registration operation. Then, the UAF Authenticator stores its *Attestation Private Key* securely; the server sends a *challenge to* the UAF Authenticator and checks the received response while the UAF Authenticator generates a response according to the *challenge* after verifying the user's biological factors in either the registration operation or the authentication operation. The difference between these two operations is that the UAF Authenticator generates the response with the *Attestation Private Key* in the registration operation and with an *Authentication Private Key* in the authentication operation. Both the *Public_Key* and the *Private_Key* (in Figure 3) are referred to the *Attestation Keys* in the registration operation, as well as the *Authentication Keys* in the authentication operation. Figure 3 also shows a case where the *AppID* from the server is empty as Section 2.2 describes.

In the registration operation, the UAF Authenticator generates a pair of *Authentication Keys* associated with user profile and sends the public key signed with *Attestation Key* (*Private_Key*) in the response message to the remote server; the server then stores the user's public key after verifying its signature by the *Attestation Public Key*; in the authentication operation, the authenticator unlocks the related *Authentication Keys* after receiving the *challenge* from the server and generates a response including a signature with *Authentication Keys* (*Private_Key*) and sends the response message to the remote server; then, the server locates the user's public key stored in registration operation, uses it to verify the signature in the message, and finally achieves the purpose of authenticating the user's presence.

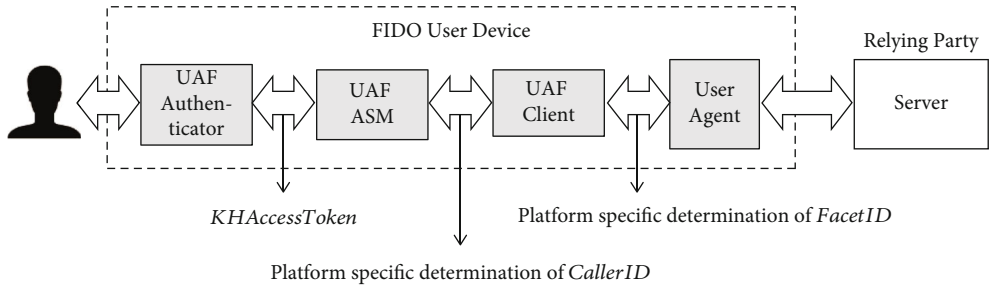

Figure 2: Trust Model of FIDO UAF Client.

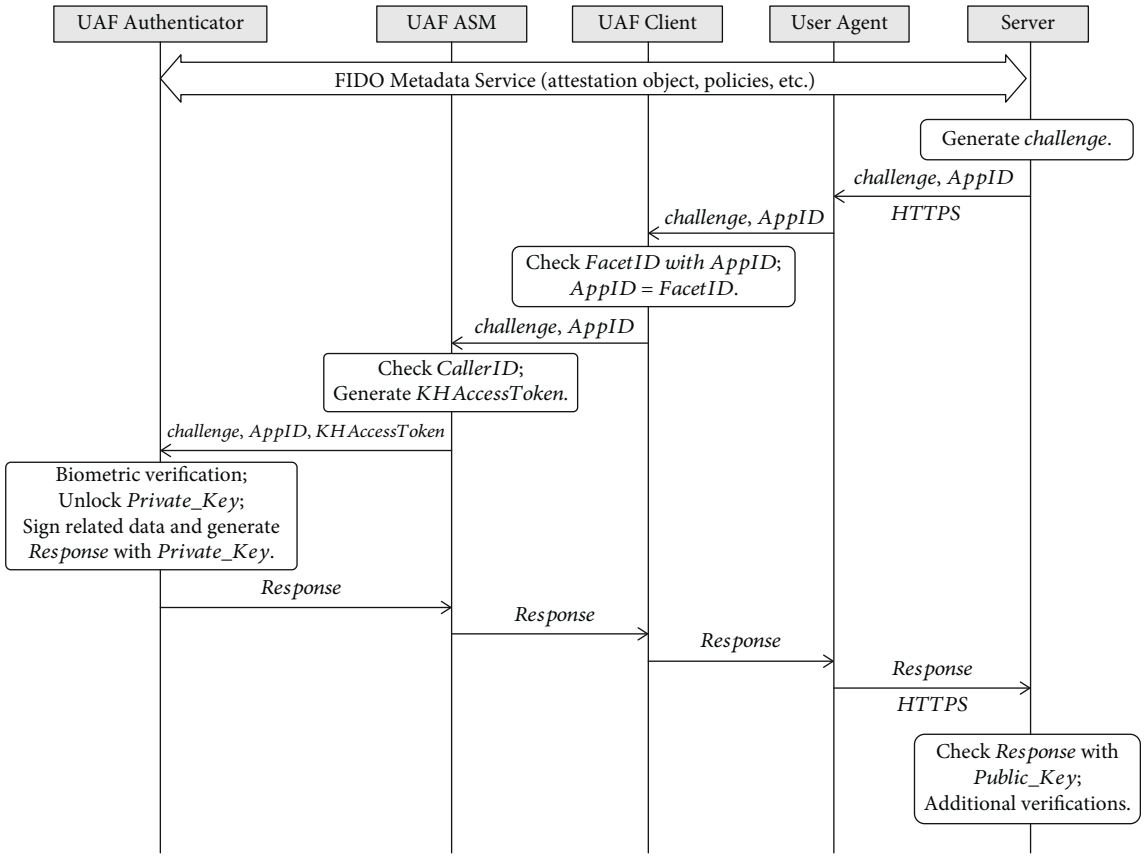

Figure 3: Simplified UAF protocol operation.

3. Implementations of the UAF Protocol

In this section, we describe two commonly implemented UAF protocol modes on the Android platform: UAF implementation based on Out-App Authenticator Mode and UAF implementation based on In-App Authenticator Mode.

3.1. Out-App Authenticator Mode. Out-App Authenticator Mode refers to the implementation mode where the User Agent, the UAF Client, and the ASM-Authenticator are three separate Android applications. One example is Hebao Pay, a third-party mobile payment product launched by China Mobile. [18] In the following section, we describe its implementation.

UAF Client Applications can be preinstalled in the phone by the manufacturer or installed by the user, which provide UAF Client functions that are compliant with the FIDO specifications and expose the standard interface. Upper-layer applications can implicitly call the UAF Client functions, which means that the upper-layer application and the UAF Client Application are decoupled. Therefore, an application can call different UAF Client Applications on devices of different brands without modifying their source codes. There are multiple implementations of UAF ASM and authenticators; some applications provide a UAF ASM interface to the UAF Client Application and implement the function of an authenticator at the same time through the native methods

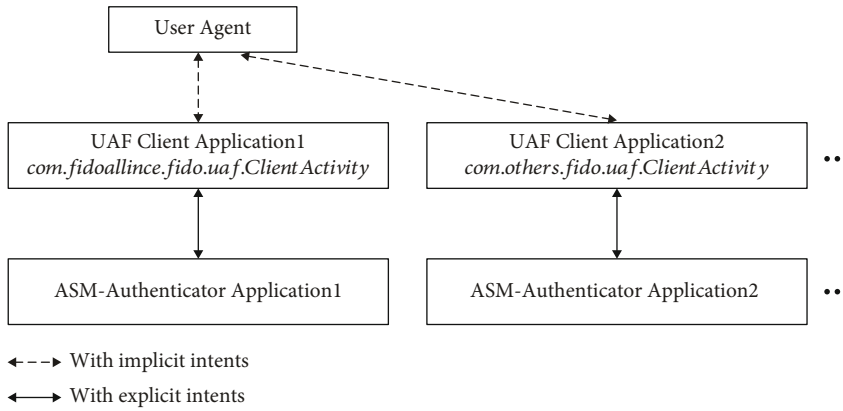

Figure 4: UAF implementation in Out-App Authenticator Mode.

or using TEE. We call such an application ASM-Authenticator Application.

Figure 4 describes the UAF implementation of Out-App Authenticator Mode; the specific process is as follows:

(1) As shown in Figure 4, the User Agent starts an Activity component of the UAF Client Application with implicit intents and uses them to pass the registration or authentication request. The Android system can automatically match the intent-filter of Activity components with the intent parameters. When multiple Activity components are matched, the user will be prompted to select one of them to start. The intent-filter of an Activity component in the UAF Client is defined in Figure 5. Implicit intents enable User Agents to call multiple UAF Client Applications

(2) After the related Activity component in the UAF Client Application is started by the User Agent, the Activity component calls *getCallingActivity*() function to obtain the caller's package name, calculates the hash of the signature certificate of the application corresponding to this package name, and generates the *FacetID* of the caller. Then, the *FacetID* is checked with *AppID*

(3) The UAF Client Application sends the request to the ASM-Authenticator Application by starting the Activity component with explicit intents, which means that such UAF Client Application explicitly specifies the ASM-Authenticator Application to call.

(4) After receiving the FIDO Client Application request, the ASM-Authenticator Application calculates the *CallerID* of FIDO Client Application. The calculation method is the same as that of *FacetID*. The ASM-Authenticator Application then verifies whether the caller is a valid FIDO Client Application by checking a whitelist. If the verification fails, the operation is aborted. Otherwise, the UAF Authenticator with the native implementation is called by the JNI mechanism to perform the FIDO operation

```
<intent-filter>
    <action android:name="org.fidoalliance.intent.FIDO_OPERATION" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="application/fido.uaf_client+json" />
</intent-filter>
```

Figure 5: Intent-filter exposed by a UAF Client.

3.2. In-App Authenticator Mode. In the In-App Authenticator Mode, the UAF Client, UAF, ASM, and UAF Authenticator modules are implemented internally inside the User Agent. For example, Jingdong Finance, a financial and third-party payment application launched by Jingdong [19], implements the UAF protocol in this mode. Such applications generally implement the UAF protocol by integrating the FIDO UAF SDK that includes the above modules. Different FIDO UAF SDKs have different implementation details, but the modules and calling processes implemented in these SDKs conform to the FIDO UAF framework described by UAF protocol specification.

We summarize the implementation of a typical In-App Authenticator Mode as shown in Figure 6. UAF Client and UAF ASM send parameters by calling the interface method of the next level entity, respectively; UAF ASM stores the authentication information (such as *KeyHandle*, *KeyID*, and *UserName*) of each registration operation in the SQLite database; the authenticator starts the FingerActivity through explicit intents to complete user authentication and other authentication functions; FingerActivity calls Android's fingerprint authentication service to verify the user's identity, calls the Android KeyStore to generate the *Authentication Key* and signature, and saves the *SignCounter* to SQLite. The *FacetID* and *CallerID* of this mode are generated by calculating the hash of the User Agent's signature certificate, so these two values do not authenticate the UAF Client and UAF ASM modules in the SDK.

4. Authenticator Rebinding Attack

In this section, we propose an attacking method called the Authenticator Rebinding Attack which enables an attacker to rebind the victims' identity to a misused authenticator, bypass the biofactor authentication of the victim's device, and initiate unauthorized payment operations. We present

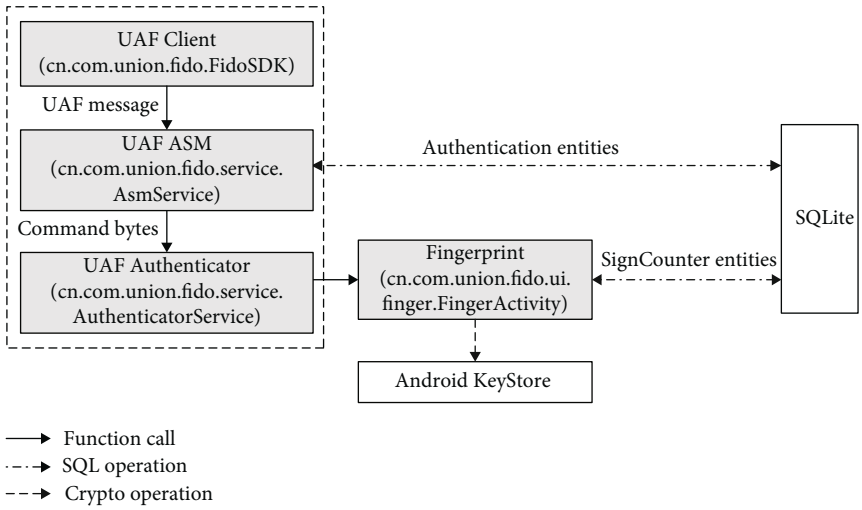

Figure 6: In-App Authenticator Mode.

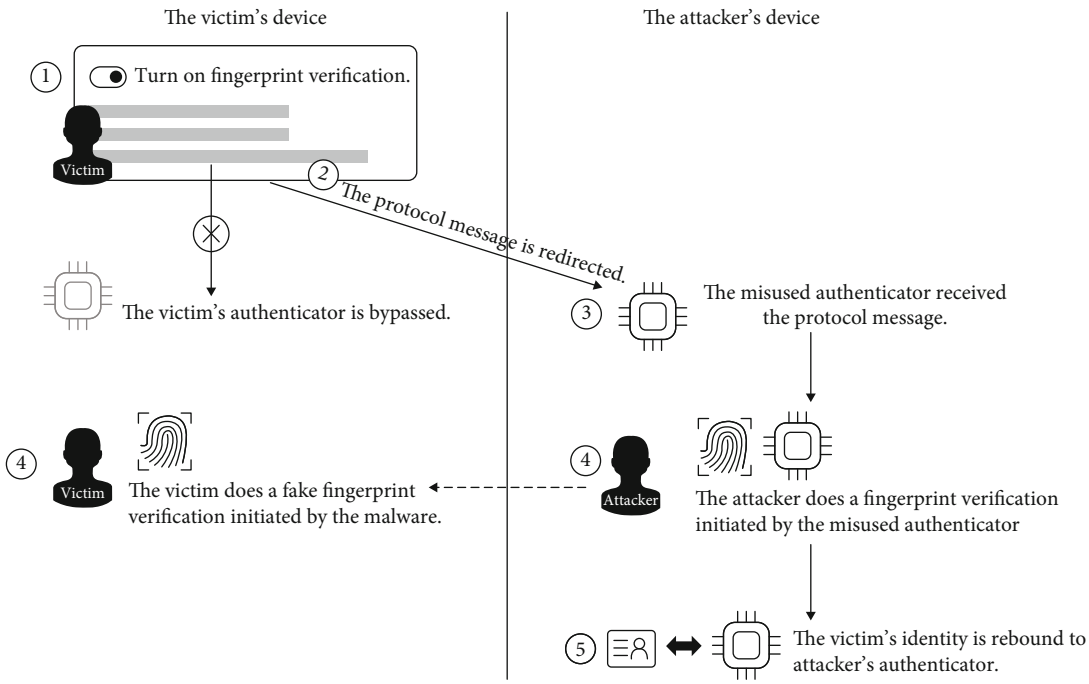

Figure 7: Overview of Authenticator Rebinding Attack.

the overview and details of this attack under the two implementation modes of the UAF protocol on Android, including the threat model, the attack process, and the verification of the attack on real-world applications.

4.1. Overview of Authenticator Rebinding Attack. Figure 7 shows an overview of the Authenticator Rebinding Attack. In the following part, we take the fingerprint authentication mechanism as a local authentication example and assume that the attacker has installed malware on the victim's device.

(1) A victim turns on the fingerprint authentication function of an application to register a FIDO UAF service in an Android application

(2) The malware redirects the protocol message from this application to the attacker's cracked device

(3) The attacker tricks his/her authenticator to continue the UAF operations with the redirected message

(4) The misused authenticator initiates a fingerprint authentication as expected. At the same time, the malware running on the victim's device uses the fake fingerprint authentication window to pretend to verify the victim's fingerprint which makes the victim not aware of any abnormalities

(5) The attacker completes the UAF protocol registration operation on behalf of the victim and rebinds the

victim's identity to the attacker's misused authenticator. Thereafter, the attacker can bypass the fingerprint verification in the user's device and perform a transfer or payment without the user's authorization

We call this attack Authenticator Rebinding Attack because the victim's identity is eventually rebound to the attacker's authenticator. Compared with the approach using malware to steal user's passwords, this type of attack is less difficult because the attacker does not need to hack the password input window, which is always protected by the Android operating system using such techniques as TEE. This attack can be used to bypass the biometric authentication process of the FIDO UAF protocol without destroying the fingerprint verification mechanism of the Android system. Therefore, with this attack, the biometric authentication process can be bypassed in the case of remote control or temporary access to the victim's device.

We have proven that this attack is effective for both UAF protocol implementation modes, and we will present the detailed processes and verifications of such attack under different protocol implementation modes in the following sections.

4.2. Attack under Out-App Authenticator Model. When the User Agent of FIDO UAF is implemented using the Out-App Authenticator Mode, even if the Android operating system is not corrupted, it may suffer from an Authenticator Rebinding Attack. Meanwhile, an attacker can complete this attack at a lower cost. In this case, we call the attack Type-A Rebinding Attack.

4.2.1. Threat Model. In Type-A Rebinding Attack, we assume that an attacker has the following abilities.

We assume that the attacker can install malware on a victim's Android devices through system vulnerabilities, inducing users, DNS hijacking, ARP attacks, or other measures. This assumption is reasonable because the public Wi-Fi users may suffer from these attacks for the existence of Rogue Access Point (RAP) [20]. Moreover, the spread of malware is still prevalent; for example, the total number of mobile malware infections in 2018 exceeded 110 million [21]. We assume that the attacker is able to remotely control the victims' mobile device temporarily or has the opportunity to temporarily access the device without root permission. These two situations will cause the attacker to implement similar attacks using different attack schemes. For example, an attacker's malware obtains the remote control permission of the victim's device by deception, or an attacker is an acquaintance of the victim and therefore can temporarily access the phone. But in both cases, the attacker cannot replace the victim to complete the fingerprint verification process on the Android device. We also assume that the malware cannot deceive the fingerprint verification service on Android devices, because the fingerprint matching should be performed in a Trusted Execution Environment (TEE) or on a chip with a secure channel to the TEE according to the requirements of Google after Android 7.0 [22].

The attacker may crack the Android device and gain the root permission. This is necessary because the attacker has to trick the FIDO ASM-Authenticator Application in his/her own device to process the UAF protocol request forwarded from the victim's device. In fact, this can be easily satisfied for two reasons. First, many Android device vendors provide bootloader unlocking services directly or indirectly, so users can also obtain root permission by flashing a third-party ROM. Second, various automated root permission acquisition tools such as KingRoot reduce the difficulty for ordinary users to obtain root permission of the Android system. Therefore, we assume that the attacker has a device with the same model and the same software version as the victim; i.e., their FIDO ASM-Authenticator Applications have the same *AAID* and *Attestation Keys*.

4.2.2. Processes. Based on the above threat model, detailed attack processes of Type-A Rebinding Attack are as follows:

(1) When a victim uses the User Agent in the user's device to open the fingerprint verification service, the registration operation of the UAF protocol is triggered to start

(2) The User Agent obtains the FIDO UAF registration request containing *AppID* and *challenge* over the TLS channel

(3) In Out-App Authenticator Mode, User Agent launches an Activity component of the UAF Client Application via implicit intent. The intent contains the FIDO UAF registration request

(4) As shown in Figure 8, the Attack Agent Client and UAF Client Application expose the same intent-filter as described in Section 3.1. Therefore, the Android operating system will prompt the victim to select a UAF Client Application in the user's device for further operation by a pop-up window as shown in Figure 9

(5) It is difficult for the victim to manually select the correct UAF Client from multiple UAF Client Applications that match implicit intents because the UAF protocol works under User Agents and is usually transparent to users. Therefore, the victim may choose the Attack Agent Client by mistake to perform further operations

(6) Through network communication, the Attack Agent Client forwards the FIDO UAF registration request to Attack Agent Server running on the attacker's device and performs a fake fingerprint verification operation, waiting for the registration response message returned by Attack Agent Server

(7) On the attacker's device, the Attack Agent Server passes the received FIDO UAF registration request to the ASM-Authenticator Application. Since the signature certificate of the Android application is packaged and published with the APK file, the

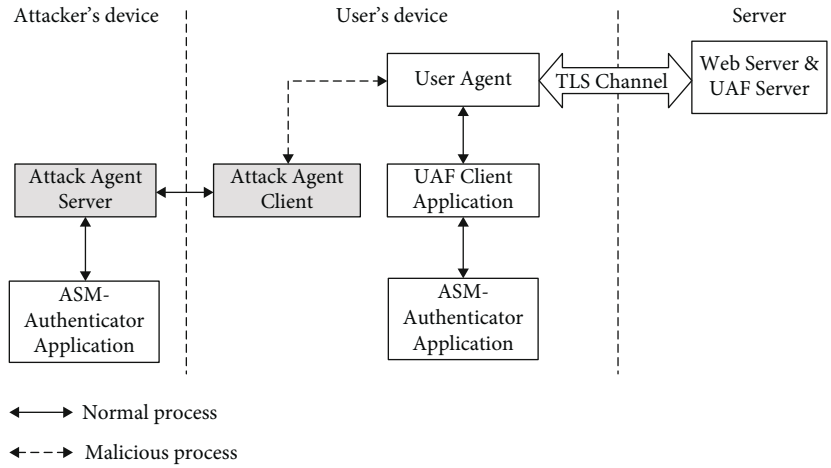

Figure 8: Type-A Rebinding Attack.

Figure 9: A pop-up window asking the victim to choose a UAF Client.

FacetID and *CallerID* can be easily forged. The Attack Agent Server changes the *FacetID* and *CallerID* to the correct value and then passes the modified parameters to the ASM-Authenticator Application

(8) The ASM-Authenticator Application verifies the UAF Client Application by *CallerID*, uses the system fingerprint verification service to verify the attacker's fingerprint, and calculates the response with the *Attestation Key*. Since *CallerID* and *FacetID* are calculated in the same way and the attacker also has the root permission of the device, *CallerID* can be changed into a correct *CallerID* easily. However, it may not be necessary in cases such as the attack example described below

(9) The registration response message generated by the misused ASM-Authenticator Application is returned to the User Agent running on the victim's device step by step according to the above path

(10) After the victim enters his/her payment password in the User Agent for confirmation, he/she completes the registration operation of the UAF protocol using the attacker's authenticator. Thereafter, the attacker can bypass the fingerprint verification through the Attack Agent Client on this victim's device and complete the payment operations

4.2.3. Validation. We choose Hebao Pay as the attack target to verify the effectiveness of the Type-A Rebinding Attack. One reason for our choice is that Hebao Pay is widely used, and the cumulative number of total downloads of Hebao

Pay in China has surpassed 129 million by the end of November 2019 [23]. Another reason is that Hebao Pay uses Out-App Authenticator Mode to provide users with fingerprint verification services based on the UAF protocol. In Huawei's smart mobile devices, Hebao Pay calls system applications UAF Client and UAF ASM in EMUI (Emotion UI) to complete the UAF protocol flow. Through reverse analysis, we find that UAF ASM in EMUI includes the functions of ASM and authenticator, so it can correspond with the ASM-Authenticator Application in the above descriptions.

We implement two attack modules: Attack Agent Client and Attack Agent Server. The former exposes the same intent-filter and sets the application name and application icon similar to the UAF Client in the victim's device. The latter is achieved by using the hook methods to modify the return value of the *Activity.getCallingActivity*() function of the UAF Client in the victim's device.

In our implementation, Hebao Pay is installed on the same device with the Attack Agent Server and the return value of the *Activity.getCallingActivity*() function is changed to the package name of Hebao Pay so that UAF Client Application can always calculate the *FacetID* of Hebao Pay. The Attack Agent Client can also calculate the caller's *FacetID* and pass it to the Attack Agent Server; then, the Attack Agent Server can modify the return value of the *FacetID* calculating function to the received *FacetID*. This could make such an attack applicable to other User Agents of Out-App Authenticator Modes.

Based on the above work, we simulate the entire process of such an attack. First, the victim attempts to open the fingerprint verification service in Hebao Pay according to the described operation in the previous sections. The fingerprint verification window pops up on the screen of the attacker's mobile phone instead of the victim's phone. After the attacker performs fingerprint verification, the victim's Hebao Pay application jumps directly to the payment password input screen. The victim inputs his/her payment password to confirm this operation, and the fingerprint verification service is successfully opened. The attacker can then perform a transfer operation, and the fingerprint verification window pops up again on the screen of the attacker's mobile phone. After verifying the attacker's fingerprint, the transfer operation is successful, which means that Type-A Rebinding Attack can bypass the fingerprint verification mechanism of Out-App Authenticator Mode as expected.

4.3. Attack under In-App Authenticator Mode. Compared with the Type-A Rebinding Attack, the attack in the In-App Authenticator Mode that is called Type-B Rebinding Attack has the same impact on the victim but requires a higher cost. This is caused by the fact that the Relying Party function modules and authenticator in In-App Authenticator Mode are highly coupled, which prevents the User Agent from calling multiple UAF Clients, thus reducing the attack surface and increasing the difficulty of such attacks.

4.3.1. Threat Model. We assume that the attacker has the ability to download the User Agent and reverse the source code of the UAF protocol so that the attacker can find the attack point at which he can redirect protocol messages in an application by manually analyzing the UAF protocol source code. It is also assumed that the malware is installed on the victim's device by the attacker and can obtain the root permission of the target device to inject the malicious code into the User Agent because the UAF protocol module of this mode is implemented inside the Reply Party Application. It also means that the attacker is able to remotely control the victims' mobile device with the root permission. The attacker is assumed to run the same In-App Authenticator Mode application on his/her cracked device, inject the malicious code, and use it as a tool to complete this attack.

4.3.2. Processes. According to the above threat model, the attack processes of Type-B Rebinding Attack are as follows. Steps (1) and (2) are the same as those of Type-A Rebinding Attack. (3) The attacker uses the malware to inject the malicious code into the victim's application, hook key functions related to the UAF protocol, and obtain the protocol messages. This operation requires root permissions of the victim's device. (4) The malware redirects the protocol message to the attacker's device through network communication. At the same time, the malware displays a fake fingerprint verification window to mislead the victim to wait until it receives the response from the attacker's device. (5) The broken In-App Authenticator Mode application on the attacker's device receives the protocol message and calls its authenticator mode to verify the attacker's fingerprint to generate the registration response message. (6) The broken In-App Authenticator Mode application sends back the registration response message to the victim's device. The following step is the same as step (10) in the Type-A Rebinding Attack.

4.3.3. Validation. We choose Jingdong Finance as the representative application of In-App Authenticator Mode to validate such attack. As of November 2019, its cumulative number of total downloads in China has exceeded 730 million [24]. Jingdong Finance implements the UAF protocol in In-App Authenticator Mode and introduces the third-party library http://cn.com.union.fido to implement this protocol. This library is also referenced by many other UAF applications in the In-App Authenticator Mode.

Through the reverse analysis, we find that a function named *process* is the entry function for the UAF ASM module to call the authenticator module. The parameters and return values are byte arrays. We hook this function and inject the code of parameters forwarding to implement the Attack Client and Attack Service modules. The function of the malicious code injected is shown in Figure 10, in which the *process* function is replaced by the *processHook* function and the parameters are forwarded to the remote Attack Server module. The Attack Server module is implemented by replacing this function to receive Attack Client's forwarded parameters.

Based on the above analysis, after the victim enables the fingerprint payment function in the Jingdong Finance application, the registration and authentication requests of the UAF protocol are forwarded to the attacker's device and the fingerprint verification mechanism of Jingdong Finance

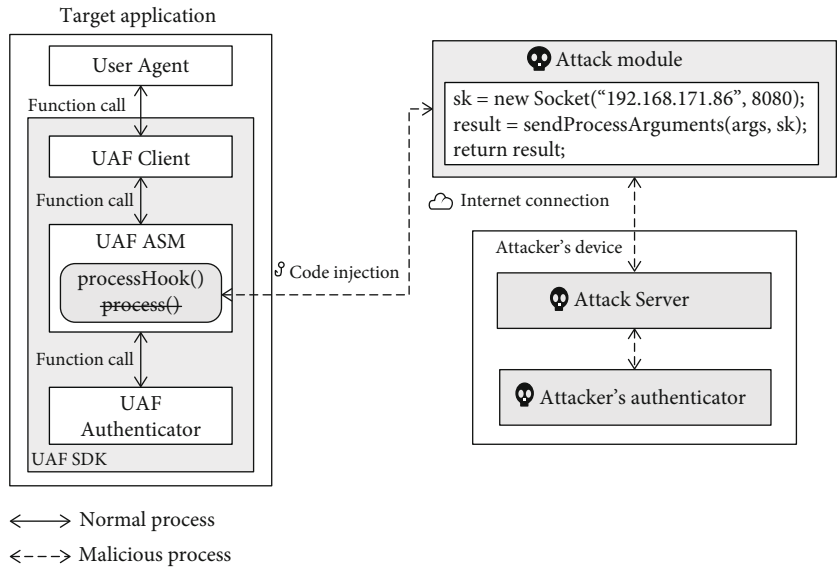

Figure 10: Injecting the malicious code to the target User Agent.

running on the victim's device is successfully bypassed. Despite requiring more rigorous attack conditions, Type-B Rebinding Attack is possible to happen in In-App Authenticator Mode User Agents.

4.4. Comparison of These Two Attacks. Both attacks under different UAF protocol implementation modes may lead to the fingerprint authentication mechanism of User Agent Applications running on the victim device to be bypassed. In general, the Type-A Rebinding Attack is easier to be implemented because the attacker does not need to obtain the root permission of the victim's device or perform a reverse analysis of the target User Agent. Moreover, some User Agents may become the potential targets during the attack because they communicate with the UAF Clients in the same way (implicit intent). However, Type-B Rebinding Attack is not easy to detect because it can be carried out without any extra interaction with the victim. Table 1 shows the difference between these two attacks.

5. Discussions

In this section, we first analyze the impact scope of this threat by studying the security of related applications in the actual system; then, we present its main causes and finally provide possible countermeasures that will remedy the threats.

5.1. Impact Scope. We manually analyze several applications that use the UAF protocol, find their characteristics, and develop programs to automatically mine such applications from a large number of Android applications. As what is claimed in the UAF protocol, if an Android application calls other UAF Client Applications to complete the FIDO UAF operation, it must declare the FIDO-related permissions in its Android manifest file [25]. Therefore, FIDO-related permissions in the manifest file can be used for searching Out-App Authenticator Mode applications. However, the application code in the In-App Authenticator Mode does not contain the code that implements the UAF protocol but uses a third-party Java library that implements the UAF protocol instead. We automatically mine the target application by retrieving the package name and critical component name of the third-party libraries contained in an application and checking whether these names contain the FIDO keywords.

Altogether, we find 42 FIDO UAF applications in Out-App Authenticator Mode and In-App Authenticator Mode. The total download number of these 42 applications in app markets is more than 222.9 million by the end of 2019. Among these 42 applications, 8 (19%) applications call third-party UAF Client Applications (Out-App Authenticator Mode), while the remaining 34 (81%) applications use the In-App Authenticator Mode to complete the operation of the UAF protocol.

For the UAF applications in Out-App Authenticator Mode, we confirm with manual analysis methods that they all use implicit calls to interact with third-party UAF Client Applications, which means that the Type-A Rebinding Attack is effective for these applications. Even if these applications use code obfuscation and packing protections, they still cannot resist such a threat. The total downloads of these applications as shown in Table 2 have exceeded 27.1 million by far.

For the UAF applications in In-App Authenticator Mode, if users use these applications on Android devices that leak root permissions, they may become the target of Type-B Rebinding Attack. These applications are protected by code obfuscation technology for the code of the UAF protocol, and their critical method names are randomly replaced with different strings. Therefore, although attackers can determine from the package names what kind of third-party FIDO UAF libraries that the developers have used, the attackers have to manually analyze the obfuscated code of every kind of applications to find the possible hook point. This will undoubtedly increase the difficulty of carrying out this attack. Table 3

Table 1: The difference between the two kinds of attacks.

	Type-A Rebinding Attack	Type-B Rebinding Attack
Attack target	Some User Agents calling third-party UAF Clients	A specific User Agent with In-App Authenticator
Requiring the root permission	No	Yes
Requiring additional user interaction	Yes	No
Requiring reverse analysis	No	Yes

Table 2: Out-App Authenticator Mode applications.

Package name	Category	Interaction method	Downloads (million)	Attack effectiveness
com.ecitic.bank.mobile	Bank	Implicit intents	14.59	√
com.bankcomm.maidanba	Bank	Implicit intents	5.38	√
cn.com.cmbc.newmbank	Bank	Implicit intents	2.32	√
com.cmbc.cc.mbank	Bank	Implicit intents	2.32	√
com.forms	Bank	Implicit intents	0.86	√
com.cmcc.hebao	Third-party payment	Implicit intents	0.75	√
com.unicom.wopay	Third-party payment	Implicit intents	0.49	√
com.hsbank.mobilebank	Bank	Implicit intents	0.39	√

shows the third-party library package names and total downloads of the In-App Authenticator Mode applications. The attack effectiveness of third-party library cn.com.union.fido is confirmed in our attack validation stage, and the attack effectiveness of other libraries stays unconfirmed.

By analyzing the applications that use the UAF protocol, we can conclude that the Authenticator Rebinding Attack has already caused substantial threats to applications with a large number of downloads, especially the applications of Out-App Authenticator Mode with implicit calls.

5.2. Main Causes. The authentication between FIDO UAF entities is not effectively implemented in both modes. Invalid authentication between FIDO UAF entities will cause the UAF Authenticator to be abused by attackers and become an attacker's tool for the attack. In Out-App Authenticator Mode, UAF Client Application authenticates User Agent via *FacetID* and ASM-Authenticator Application authenticates UAF Client Application via *CallerID*. As an example of our research, both *FacetID* and *CallerID* are obtained by calculating the hash of the target application's signature certificate. However, the signature certificate can only guarantee the integrity of the Android application static code or APK file and cannot guarantee the integrity of the application at runtime. Similarly, in In-App Authenticator Mode, *FacetID* and *CallerID* cannot be used to ensure that the internal modules of a User Agent are not tampered by an attacker at runtime. Therefore, *FacetID* and *CallerID* cannot be used in these situations to guarantee the authentication between UAF protocol entities. On the contrary, if entities are effectively authenticated and the authentication information is included in the response, at least the remote server can detect whether the integrity of some entities has been compromised and then abort the protocol operation. In conclusion, it is the lack of effective authentication between entities in the implementations of the UAF protocol that the UAF protocol used in the actual system is vulnerable to the Authenticator Rebinding Attack.

5.3. Countermeasures. We now discuss possible countermeasures to effectively mitigate Authenticator Rebinding Attack from the perspective of protocol designers, developers of the User Agent Applications, and mobile device providers and users.

For designers of the UAF protocol, our suggestion is to enhance the authentication mechanism between the UAF entities by adding the verification of Android platform integrity based on TEE or hardware. Although the Android operating system has an isolation mechanism for applications, Android applications, for example, the application of the User Agent or the UAF Client, may still be damaged at runtime when the Android operating system is corrupted, which leads to the attack mentioned above. Therefore, if the FIDO server can authenticate the integrity of the Android operating system and combine this with the verification mechanism of *FacetID* and *CallerID*, the authentication between FIDO UAF entities can be indirectly guaranteed. For example, the TrustZone-based Integrity Measurement Architecture (TIMA) proposed by Samsung can prove the applications running in a trusted environment to the remote server [26]. And this technology can be integrated with the UAF protocol so that the authenticator can sign the *challenge* along with the attestation data, which contains boot component cryptographic hashes to indicate the integrity of the operating system. In this way, the server can determine whether the authenticator is running in a secure device by checking the TIMA attestation data.

For the developers of User Agent Applications, we first suggest using explicit intent to call the third-party UAF Client. In this case, the Package Manager Service (PMS) of the

Table 3: In-App Authenticator Mode libraries and applications.

Library package name	Attack effectiveness	Application package name	Code protection measure	Downloads (million)
cn.com.union.fido	√	com.jd.jrapp	Code obfuscation	23.83
		com.csii.sns.ui	App reinforcement	0.80
		com.cebbank.mobile.cemb	App reinforcement	0.36
		cn.com.bhbc.mobilebank.per	App reinforcement	0.30
		com.chinamworld.klb	App reinforcement	0.06
		cn.com.gdbank.direct	App reinforcement	0.01
		com.csii.ly.ui	App reinforcement	0.01
		com.csii.wjnsbank	App reinforcement	Less than 0.01
		com.urthinker.langfangbank.lfbank	App reinforcement	Less than 0.01
		com.csii.yk.ui	App reinforcement	Less than 0.01
		com.csii.zbdirect	App reinforcement	Less than 0.01
com.daon.fido.client.sdk	Unconfirmed	com.bochk.com	Code obfuscation	0.05
com.fido.android.framework	Unconfirmed	com.chinatelecom.bestpayclient	App reinforcement	34.45
com.iss.sdpersonalbank.fidofinger	Unconfirmed	com.iss.weifangbank	App reinforcement	0.17
		com.iss.rizhaobank	App reinforcement	0.13
		com.uccb.mobile	App reinforcement	0.13
		com.iss.changanbank	App reinforcement	0.12
		com.iss.weihaibank	App reinforcement	0.10
		com.iss.qilubank	App reinforcement	0.09
		com.iss.qishangbank	App reinforcement	0.09
		com.iss.jiningbank	App reinforcement	0.08
		com.iss.taianbank	App reinforcement	0.08
		com.iss.dongyingbank	App reinforcement	0.07
		com.iss.laishangbank	App reinforcement	0.07
		com.iss.ysantaibank	App reinforcement	0.07
		com.iss.dezhoubank	App reinforcement	0.06
		com.iss.zaozhuangbank	App reinforcement	0.02
com.lenovo.fido.framework	Unconfirmed	com.baidu.wallet	App reinforcement	1.69
		com.bill99.kuaiqian	App reinforcement	1.58
Unknown	Unconfirmed	com.icbc	App reinforcement	69.67
		com.chinamworld.bocmbci	App reinforcement	38.06
		com.icbc.im	App reinforcement	22.57
		com.baixin.mobilebank	App reinforcement	0.52
		com.icbc.collegestudents	App reinforcement	0.11

Android system can accurately locate the real UAF Client, so the malicious UAF Client hence has no chance to launch an attack. Second, the developers should consider implementing the verification mechanism to the third-party UAF Client in their applications (e.g., verifying the hash value of the third-party FIDO UAF signing certificate with a whitelist). Moreover, if the UAF protocol is implemented in In-App Authenticator Mode, application reinforcement and code obfuscating technology can be used to prevent static analysis of the applications. Finally, the hook detection mechanism [27] may also be applied so that when the attacker tries to hook functions related to the UAF protocol as described in Section 4.3, the FIDO UAF service can be disabled in time, which can prevent Type-B Rebinding Attack.

For mobile device providers, besides protecting the authenticator, a strict root detection mechanism also supported by TEE [28] should be used to protect the FIDO UAF components, which will not be compromised by malicious codes without hardware-based protections. Once it is detected that the FIDO UAF components have been corrupted, disabling the FIDO UAF service can prevent the device from being exploited by attackers in the manner shown in Section 4.2.

For users, when choosing from multiple UAF Clients, they should be careful and confirm the source and security of UAF Client; for example, check whether the UAF Client is a system application; if not, then refuse to install to make the malware difficult to disguise as a system application

without the root permission. Besides, the user should avoid using FIDO UAF authentication when the root permission of the Android device is leaked, because the malware can easily use the root permission to launch this attack silently (without additional user interaction).

6. Conclusions

In this paper, we analyze a novel attack named Authenticator Rebinding Attack of the UAF protocol, which makes the victim's identity be rebound to the attacker's authenticator so that the attacker can impersonate the victim's identity. In order to comprehensively study the threats of such an attack, we first analyze the applications related to third-party payment, banking, and online shopping; mine those applications that use the UAF protocol; and model two main implementations of the UAF protocol, i.e., Out-App Authenticator Mode and In-App Authenticator Mode. We then describe the detailed attack process of these two implementation modes. We also demonstrate that the proposed attacks do work by performing attack verification on typical actual applications. Besides, the applications that use UAF protocol on the Android platform in the actual system are threatened by this attack and the applications that make implicit calls in Out-App Authenticator Mode are more vulnerable. This threat can be attributed to the lack of effective authentication between entities when the UAF protocol is implemented on the Android platform. The *FacetID* and *CallerID* used by the UAF protocol cannot prove the integrity of the User Agent and UAF Client. We finally present countermeasures that can prevent this threat. We believe that our research on the Authenticator Rebinding Attack of the UAF protocol can help protocol designers, User Agent Application developers, and mobile device providers and users to improve the security of the UAF protocol.

Data Availability

The APK files used to support the findings of this study are downloaded from http://zhushou.360.cn/. The python script used to support the findings of this study is uploaded to the git repository https://github.com/PandaQ2014/FindFIDO. The statistical data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This research is supported by the National Science and Technology Major Project of China (2018ZX03001010-005).

References

[1] S. Machani, R. Philpott, S. Srinivas, J. Kemp, and J. Hodges, *FIDO UAF Architectural Overview*, FIDO Alliance, 2017.

[2] FIDO Alliance, "FIDO certified products," 2019, https://fidoalliance.org/certification/fido-certified-products/.

[3] K. Hu and Z. Zhang, "Security analysis of an attractive online authentication standard: FIDO UAF protocol," *China Communications*, vol. 13, no. 12, pp. 189–198, 2016.

[4] C. Xenakis, C. Panos, S. Malliaros, C. Ntantogian, and A. Panou, *A security evaluation of FIDO's UAF protocol in mobile and embedded devices*, International Tyrrhenian Workshop Springer, Cham, 2017.

[5] R. Lindemann, D. Baghdasaryan, and B. Hill, *FIDO security reference*, FIDO Alliance Proposed Standard, 2015.

[6] FIDO Alliance, "Certification Overview," 2019, https://fidoalliance.org/certification/.

[7] International Data Corporation, "Smartphone market share," 2020, https://www.idc.com/promo/smartphone-market-share/vendor.

[8] Y. Zhang, X. Wang, Z. Zhao, and H. Li, "Secure display for FIDO transaction confirmation," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pp. 155–157, New York, NY, USA, 2018.

[9] StatCounter, "Mobile operating system market share worldwide," 2020, https://gs.statcounter.com/os-market-share/mobile/worldwide.

[10] FIDO Alliance, "FIDO certified showcase," 2019,). https://fidoalliance.org/fido-certified-showcase.

[11] "FIDO AllianceFIDO UAF architectural overview," 2017, https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html.

[12] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach, "Origin-bound certificates: a fresh approach to strong client authentication for the web," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pp. 317–331, Bellevue, WA, 2012.

[13] FIDO Alliance, "FIDO UAF protocol specification," 2017, https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-protocol-v1.1-id-20170202.html.

[14] R. Lindemann, E. Tiffany, B. Davit, D. Balfanz, B. Hill, and J. Hodges, *FIDO UAF protocol specification v1.1*, FIDO Alliance, 2017.

[15] FIDO Alliance, "FIDO UAF authenticator-specific Module API," 2017, https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-asm-api-v1.1-id-20170202.html.

[16] FIDO Alliance, "FIDO AppID and Facet specification," 2017, https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-appid-and-facets-v1.1-id-20170202.html.

[17] FIDO Alliance, "FIDO technical glossary," 2017, https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-glossary-v1.1-id-20170202.html.

[18] China Mobile, *Hebao Pay, pay for reliability*, China Mobile Limited, 2020, https://www.cmpay.com/.

[19] JD Digits, *A Friend Who Understands Finance*, JD Digits, 2020, https://jr.jd.com/.

[20] W. Yang, X. Li, Z. Feng, and J. Hao, "TLSsem: a TLS security-enhanced mechanism against MITM attacks in public WiFis," in *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*, Fukuoka, Japan, 2017.

[21] Beijing Qihu Keji Co Ltd, *2018 Android Malware Special Report*, Technical Report, 2018.

[22] Google Inc, "Android compatibility definition (Android 7.0)," 2017, https://source.android.google.cn/compatibility/7.0/android-7.0-cdd.

[23] Kuchuan, "Hebao payment application data page," 2019, https://android.kuchuan.com/page/detail/download?package=com.cmcc.hebao&infomarketid=10&site=0#!/sum/com.cmcc.hebao.

[24] Kuchuan, "Jingdong Finance application data page," 2019, https://android.kuchuan.com/page/detail/download?package=com.jd.jrapp&infomarketid=1&site=0#!/sum/com.jd.jrapp.

[25] B. Hill, D. Baghdasaryan, B. Blanke, J. Hodges, and K. Yang, *FIDO UAF application API and transport binding specification v1.1*, FIDO Alliance, 2017.

[26] A. M. Azab, P. Ning, J. Shah et al., "Hypervision across worlds: real-time kernel protection from the ARM TrustZone secure world," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, pp. 90–102, New York, NY, USA, 2014.

[27] M. Szczepanik, I. J. Jóźwiak, P. P. Jóźwiak, M. Kędziora, and J. Mizera-Pietraszko, "Android hook detection based on machine learning and dynamic analysisWeb, Artificial Intelligence and Network Applications," vol. 1150 of WAINA 2020. Advances in Intelligent Systems and Computing, Springer, Cham, 2020.

[28] GlobalPlatform, *The trusted execution environment: delivering enhanced security at a lower cost to the mobile market*, GlobalPslatform Inc, 2015.