FIDO U2F Application Isolation through Facet Identification

Specification Set: fido-u2f-v1.0-rd-20140209 REVIEW DRAFT

Editors:

Dirk Balfanz (balfanz@google.com)

Contributors:

Abstract:

This document specifies how FIDO should enforce application isolation. In particular, it outlines a mechanism that relies on two properties of the FIDO client:

1. The FIDO client, and only the FIDO client, can talk to the FIDO authenticator directly.
2. The FIDO client can security identify the application making a FIDO request.

The document explains why it is reasonable to assume Point (1) above, and also explain how an addition level of indirection between what we call a facet id and an application identity, combined with Point (2), allows us to move authenticators between devices.

Status:

Table of Contents

1 Notation

Type names, attribute names and element names are written in *italics*.

String literals are enclosed in “”, e.g. “UAF-TLV”.

In formulas we use “|” to denote byte wise concatenation operations.

U2F specific terminology used in this document is defined in [FIDOGlossary]

1.1 Key Words

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

2 Background

Note: Reading the 'FIDO U2F Overview' [U2FOverview] is recommended as a background for this document.

Identity assertions in FIDO should be application-specific. In other words, Phishers-Я-Us must not be able to to obtain a user's PayPal credentials from that user's authenticator. This can be achieved by always including the requesting application (Phishers-Я-Us vs. PayPal) in the identity assertion (thus making the identity assertion obtainable by Phishers-Я-Us unusable with the PayPal app), but that is not enough: for privacy reasons, the user's authentication key itself (i.e., the key making the identity assertion) should be application-specific, so as to not allow user identity correlation across different applications (i.e., a user's authenticator should use a different authentication key for Phishers-Я-Us than it uses for PayPal). A particularly strong expression of this principle is that FIDO authenticators should indeed **refuse** to make "cross-application" identity assertions (i.e., a user's authentication key for PayPal will never be used by the user's authenticator to issue identity assertions for Phishers-Я-Us, even assuming that such an assertion would correctly identify the Phishers-Я-Us application as the target of the authentication), so as to not give Phishers-Я-Us a tool to learn the PayPal identity of the user.

The problem, therefore, is how we can enforce this application-binding of keys, and prohibit cross-application identity assertions. This document specifies a simple solution:

1. FIDO authenticators record somehow which user authentication keys should be used with which application, and

2. a trusted piece of software (the FIDO client) provides the FIDO authenticator with the application identity every time it asks the FIDO authenticator to issue an identity assertion. The authenticator then simply compares the application that a given authentication key was bound to with the application identity provided by that trusted piece of software and only issues an identity assertion if the application identities match.

This general approach enables *portable* authenticators, i.e., if I unplug an authenticator from one computer and plug it into another, I will be able to authenticate from the second computer without having to re-register the authenticator with the web site that I want to use. For example, if I use an authenticator to authenticate to paypal.com from computer A, I will be able to authenticate to paypal.com from computer B. This is because both computers will identify the application in question identically to the authenticator.

But what happens when PayPal gets bought by eBay, and their URL changes to ebay-payments.com? What happens when I use the PayPal Android app instead of the paypal.com desktop web site? The authenticator should re-use the same user authentication key in those cases, even though the application identity arguable is different. In this document, we assume that the application that wishes to make use of a FIDO authenticator is identified by two separate monikers: the *application identity*, and the *facet identity*. Across all facets of an applications (the various web origins it uses, its Anroid app, its iOS app, etc.) the application identity remains the same, while the facet identity identifies the particular application facet.

Identity assertions are made specific to a *facet identity*, but they’re signed with a key that is specific to an *application identity*. (More on this below.)

3 Overview

The main idea is that instead of binding user authentication keys to web origins, we bind them to an application identity. So instead of saying “this keypair can only be used with paypal.com”, we say “this keypair can only be used by the PayPal application”.

An “application”, for the purpose of this specification, can have multiple facets. For example, the various facets of the “PayPal application” could be:

- The web site paypal.com
- The web site ebay-payments.com
- An Android app signed with a certain public key
- The iOS app with the iOS Bundle ID com.paypal
- …

An application is identified through an HTTPS URL. The document at that URL lists all the facets that belong to the application identified by the URL as a JSON array. The FIDO client can therefore verify that a particular facet that is requesting an identity assertion in fact belongs to the application that it claims to be a facet of.

4 Definitions

- **Application**: a set of functionality provided by a common entity (the *application owner,* aka the *Relying Party* in FIDO parlance), and perceived by the user as belonging together. For example, “PayPal” is an application that allows users to pay for stuff.
- **Facet**: an (application) facet is how an application is implemented on various platforms. For example, the application PayPal may have an Android app, an iOS app, and a Web app. These are all facets of the PayPal application.
- **Facet ID**: a platform-specific identifier (URI) for an application facet.
    - For the Web, the facet id is the web origin, written as a URI without a path (e.g., “https://login.paypal.com” (default ports are omitted)).
    - For Android, the facet id is the URI

      ```
      android:apk-key-hash:<hash-of-apk-signing-cert>
      ```

      where the hash of the APK-signing cert is obtained by running the following command:

      ```
      keytool -exportcert -alias androiddebugkey -keystore <path-to-apk-sign-
      ing-keystore> &>2 /dev/null | openssl sha1 -binary | openssl base64 | sed
      's/=//g'
      ```

    - For iOS, the facet id is the URI **ios:bundle-id:*<ios-bundle-id-of-app>***
- **Application Identity**: an HTTPS URL that resolves to a list of facet ids.

5 Detailed Specification

The picture below shows the overall architecture of a FIDO deployment (on the client side). On the various platforms (Web, Android, iOS), we imagine a platform interface that handles the API calls (“enroll”, “getIdentityAssertion”) from apps. The component inside the platform that implements this API is called the FIDO Client.

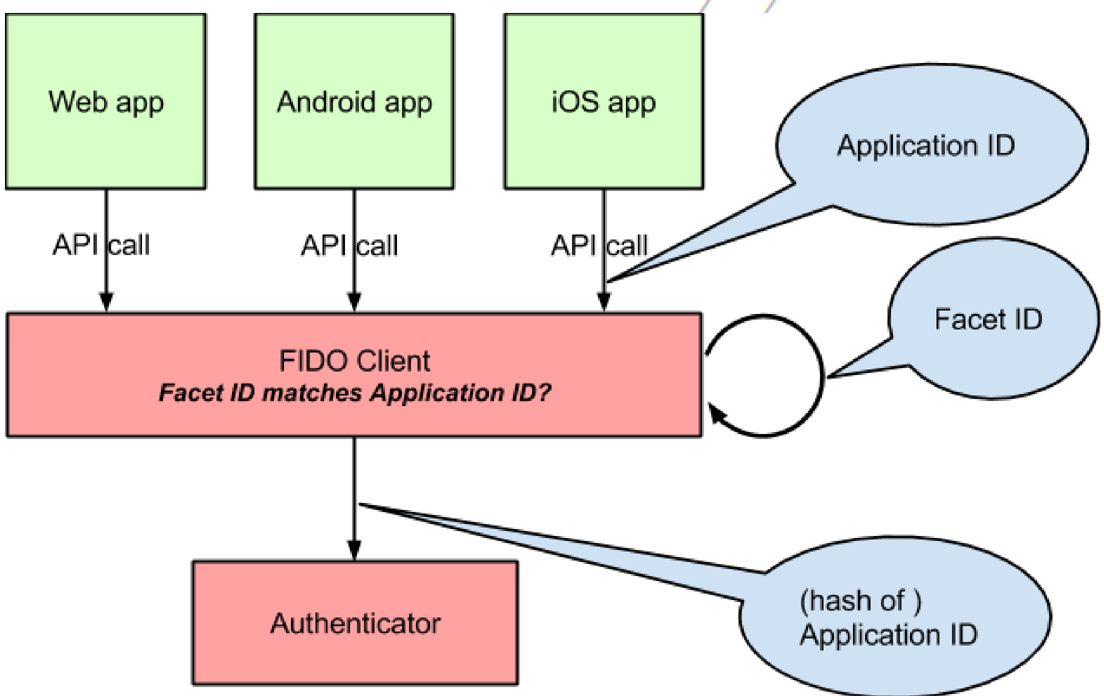

Figure 5.1: Architecture Overview

On the different platforms (Web, Android, iOS), the FIDO Client will be implemented differently. For example, for the Web we can imagine a browser extension that plays the role of the FIDO Client[1]. On Android, the Android Account Manager could play the role of the FIDO Client.

On each platform, the FIDO Client will be able to identify the calling app, and thus determine its facet id. For example, the browser extension (or, in the future the browser itself) will be able to see the web origin of the calling app. Similarly, an Android system component like the Account Manager can identify the APK signing key of the Android app making an API call into the Account Manager There is a similar mechanism in iOS.

The main idea is that each app (or “application facet”, be it a web app, an Android app, or an iOS app) will provide to the API call its application identity.

The FIDO Client then establishes the facet identity of the calling app and checks that the provided application identity identifies an application that contains the calling facet as follows:

[1] In the future, we hope that this functionality will be built into the browser itself.

1. It identifies the calling facet: On Android, the O/S provides facilities to obtain the APK signing cert of a calling app. On iOS, the O/S provides facilities to obtain the iOS Bundle ID of the calling app. On the Web, the browser (and servers) usually know the Web origin of callers.
2. It resolves the URL that is passed by the calling app as the Application Identity. This will result in a list of facet ids, represented as a JSON array of strings.
3. If the calling facet is on the list of facet ids published through the Application Identity URL, then the platform will consider the application identity verified, and continue processing the request for the specified Application Identity.

Finally, the FIDO Client uses the (hash of) the application identity to direct the FIDO authenticator as to which authentication key to use.

Let's look at registration and sign-in separately:

5.1 Registration

The registration API allows the calling facet to pass, among other things, the application identity to the FIDO client.

Because the FIDO client can identify the calling facet (see above), it now knows two things:

1. The identity of the calling facet, and
2. the application identity that the calling facet wants to invoke.

The FIDO client now checks the facet identity assertion and thus verifies that the application claims the calling facet as one of its own (see above). The FIDO client requests that the authenticator generate a user authentication keypair that is bound to the application identity URL. The authenticator responds with the following data:

- a key handle
- a public user authentication key (signed by an attestation key),

which the FIDO client passes on to the application. The application stores (presumably server-side) the key handle and public key.

5.2 Sign-In

The sign-in API allows the calling facet to pass, among other things, the following data to the FIDO client:

- the application identity

- a challenge from the relying party
- a key handle.

The FIDO client checks that the facet identity matches the provided application identity, using the mechanism described above. It then creates an authenticator-challenge by hashing the following data:

- the challenge from the relying party
- the facet identity (note that in the case of the Web this is the origin)
- optionally some channel-binding data such as the client's Channel ID

It sends the authenticator-challenge, the key handle, and the (hash of the) application identity key to the authenticator. The authenticator checks that the key indicated by the key handle can be used for the provided application identity and if so, signs the authenticator-challenge.

The FIDO client, upon receiving the signature, returns the signature along with the authenticator-challenge preimage (i.e., the facet identity, channel-binding data, etc.) to the calling facet, which sends the data to its server. The server checks (among other things) that the facet identity in the authenticator-challenge preimage is one of its facets, and verifies the signature with the public user authentication key.

5.3 Example

ACME, Inc. might create the following application identity: https://acme.com/app-identity. This URL, when resolved by a client, could return the following content:

```
[
    ‘https://login.acme.com’,
    ‘android:apk-key-hash:2jmj7l5rSw0yVb/vlWAYkK/YBwk’
    ‘ios:bundle-id:com.acme.app’
]
```

The ACME Android app might decide to create a keypair by using an API such as this:

```
KeyPair keyPair =
  FIDO_U2F_API.enroll(“https://acme.com/app-identity”);
```

The FIDO_U2F_API class passes the call to the operating system, which performs the following steps:

1. It identifies the calling Android app as being signed by certain APK signing key, and hence its Android "facet id" as *android:apk-key-hash:2jmj7l5rSw0yVb/vlWAYkK/YBwk*

2. It resolves the supplied URL https://acme.com/app-identity and obtains the JSON array shown above.

3. It checks whether the facet id is in the list of ids contained in the application URL document. (It is.)

4. It instructs the authenticator to create a new key pair that is bound to the application identity ‘https://acme.com/app-identity’.

Let’s assume that the authenticator is now moved from the Android device to a laptop running a web browser. The user visits https://login.acme.com/login-page, which contains Javascript calling a similar API, this time making use of the key pair:

```
assertion = navigator.u2f.sign(challenge, “https://acme.com/app-identity”);
```

This time, the browser will perform the following steps:

1. It identifies the calling origin as https://login.acme.com

2. It resolves the supplied URL https://acme.com/app-identity and obtains the JSON array shown above.

3. It checks whether the calling origin is in the list of ids contained in the application URL document. (It is.)

4. It then forwards the request to sign the challenge to the authenticator, noting the application identity to be ‘https://acme.com/app-identity’.

Facet Identity Confusion

A rogue application facet must not be allowed to talk to the authenticator directly, since it could forge the facet identity in the authenticator-challenge (and lie about its application identity), thus obtaining an identity assertion for a different application. On the various platforms, we achieve this in different ways:

- On the web, we simply don’t expose the API that would allow direct access to the authenticator to web applications. A browser extension (and obviously the browser itself) on the other hand, will have access to such an API (e.g., this is already the case if the authenticator is connected through USB).
- On mobile operating systems, we imagine that special permissions will be required to talk to the authenticator directly. The FIDO client will have such permissions, and it will be rare for other applications to need such permissions. All applications that request such permissions should be audited by the respective owner of the app stores on the various platforms, and should be removed from the app store if they are found to abuse these permissions.

Application Identity Confusion

What happens when a rogue application facet can trick the FIDO client into associating it with the wrong application? Since the facet identity will always be part of the authenticator's identity assertion (except if there is facet identity confusion - see above), the resulting identity assertion will be issued to the rogue facet. When the facet attempts to use the identity assertion with the application that it (wrongly) claimed to be part of, this will therefore be detected. What *can* happen, however, is that the authenticator uses a different signing key to issue to the identity assertion.

In summary, a weakness in the facet identification mechanism results in a *security* vulnerability, i.e., identity assertions that are issued to facets other than those legitimately belonging to an application. In contrast, a weakness in the application-id matching mechanism results in a *privacy* (but *not* the above-mentioned security) vulnerability, causing the authenticator to use a key (in other words, a user identifier) that should have been reserved for a different application.

Discussion

Q: What about Windows and Mac OS?
A: Windows and Mac OS are in the process of being able to isolate and identify applications similar to mobile operating systems. Until such mechanisms become available, we can provide best-effort app identification (but obviously with much lower reliability). Alternatively, we could decide to only support the Web platform on these operating systems for the time being.

Q: What about browsers on Android/iOS?
A: One approach would be to support two (types of) FIDO Clients on these platforms: One that lives inside (each) browser, and one that handles API calls from native apps. Another approach would be to have one FIDO client on the platform, and treat browser special: Unlink other applications, a (white-listed) browser would be able to assert facet ids to the FIDO client.

Bibliography

FIDO Alliance Documents:

[FIDOGlossary] Rolf Lindemann, Davit Baghdasaryan, Brad Hill, John Kemp. FIDO Technical Glossary. Version v1.0-rd-20140209, FIDO Alliance, February 2014. See http://fidoalliance.org/specs/fido-glossary-v1.0-rd-20140209.pdf

[**U2FOverview**] Sampath Srinivas, Dirk Balfanz, Eric Tiffany. FIDO Universal 2nd Factor (U2F) Overview. Version v1.0-rd-20140209, FIDO Alliance, February 2014. See http://fidoalliance.org/specs/fido-u2f-overview-v1.0-rd-20140209.pdf

Other References:

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels (RFC2119), S. Bradner, March 1997