Using Hardware Token-based 2FA with the WebAuthn API

By J.C. Jones, Tim Taubert

Posted on January 16, 2018 in Featured Article, Security, and Web APIs

To provide higher security for logins, websites are deploying two-factor authentication (2FA), often using a smartphone application or text messages. Those mechanisms make phishing harder but fail to prevent it entirely — users can still be tricked into passing along codes, and SMS messages can be intercepted in various ways.

Firefox 60 will ship with the WebAuthn API enabled by default, providing two-factor authentication built on public-key cryptography immune to phishing as we know it today. Read on for an introduction and learn how to secure millions of users already in possession of FIDO U2F USB tokens.

Creating a new credential

Let's start with a simple example: this requests a new credential compatible with a standard USB-connected FIDO U2F device; there are many of these compliant tokens sold with names like Yubikey, U2F Zero, and others:

```
const cose_alg_ECDSA_w_SHA256 = -7;

/* The challenge must be produced by the server */
let challenge = new Uint8Array([21,31,105 /* 29 more random bytes generated by the server */]);
let pubKeyCredParams = [{
  type: "public-key",
  alg: cose_alg_ECDSA_w_SHA256
}];
let rp = {
  name: "Test Website"
};
let user = {
  name: "Firefox User <firefox@example.com>",
  displayName: "Firefox User",
  id: new TextEncoder("utf-8").encode("firefox@example.com")
};
```

```
let publicKey = {challenge, pubKeyCredParams, rp, user};
navigator.credentials.create({publicKey})
  .then(decodeCredential);
```

In the case of USB U2F tokens, this will make all compatible tokens connected to the user's system wait for user interaction. As soon as the user touches any of the devices, it generates a new credential and the Promise resolves.

The user-defined function `decodeCredential()` will decode the response to receive a key handle, either a handle to the ECDSA key pair stored on the device or the ECDSA key pair itself, encrypted with a secret, device-specific key. The public key belonging to said pair is sent in the clear.

The key handle, the public key, and a signature must be verified by the backend using the random challenge. As a credential is cryptographically tied to the web site that requested it, this step would fail if the origins don't match. This prevents reuse of credentials generated for other websites.

The key handle and public key will from now on be associated with the current user. The WebAuthn API mandates no browser UI, which means it's the sole responsibility of the website to signal to users they should now connect and register a token.

Getting an assertion for an existing credential

The next time the user logs into the website they will be required to prove possession of the second factor that created the credential in the previous section. The backend will retrieve the key handle and send it with a new *challenge* to the user. As `allowCredentials` is an array, it allows sending more than one token, if multiple tokens are registered with a single user account.

```
/* The challenge must be produced by the server */
let challenge = new Uint8Array([42,42,33 /* 29 more random bytes generated by the server */]);
let key = new Uint8Array(/* … retrieve key handle … */);

let allowCredentials = [{
  type: "public-key",
  id: key,
  transports: ["usb"]
}];

let publicKey = {challenge, allowCredentials};

navigator.credentials.get({publicKey})
  .then(decodeAssertion);
```

Again, all connected USB U2F tokens will wait for user interaction. When the user touches a token it will try to either find the stored key handle with the given ID, or try to decrypt it with the internal secret key. On success, it will return a signature. Otherwise the authentication flow will abort and will need to be retried by the website.

After decoding, the signature and the key handle that were used to sign are sent to the backend. If the public key stored with the key handle is able to verify the given signature over the provided challenge, the assertion is considered valid and the user will be logged in.

First Factor Authentication

Web Authentication also defines the mechanisms to log in without a username and password at all using a secure token — such as the trusted execution environment on your smartphone. In this mode, your token would attest that you not only have possession of it, but also that you, as a person, unlocked the token using a passcode (*something you know*) and/or a biometric (*something you are*).

In this world, websites could let you enroll to perform seamless authentication to a web application on your desktop by answering a prompt which appears on your smartphone.

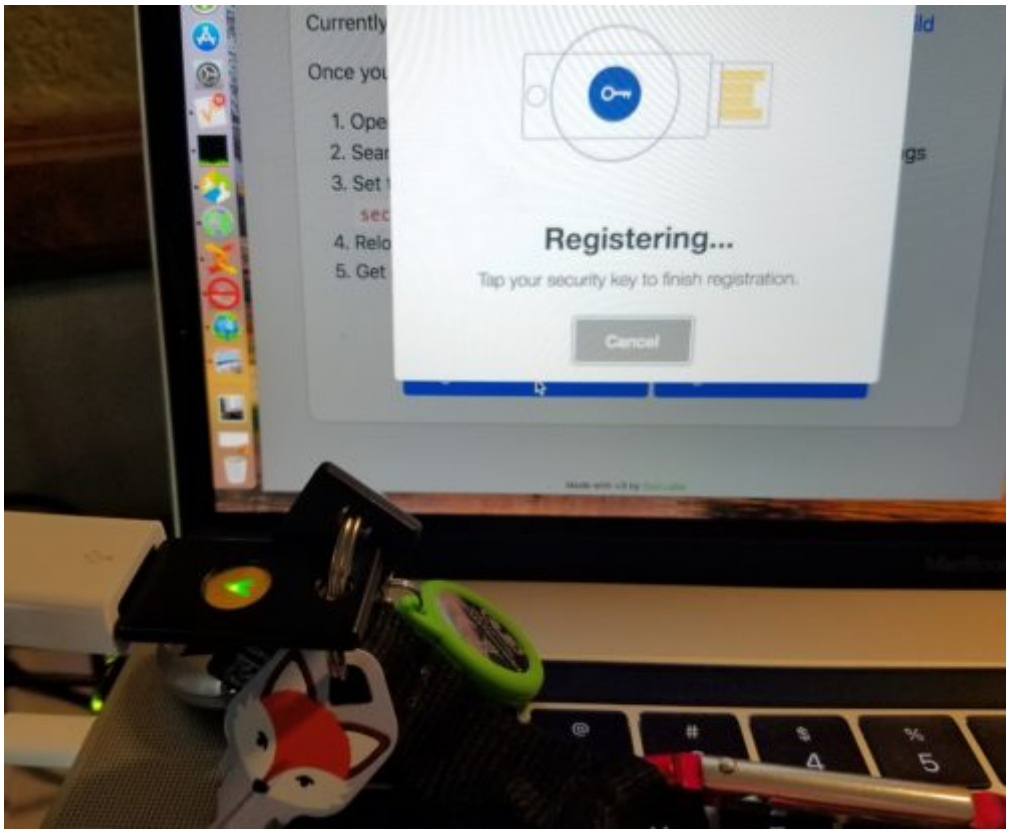
The FIDO U2F tokens deployed today aren't sophisticated enough to make this happen yet, but the next generation of tokens will be, and web developers will interact with those FIDO 2.0 tokens using Web Authentication.

WebAuthn, coming to a Firefox near you

This was a very short introduction to the world of Web Authentication and it intentionally omits a lot of nitty-gritty details such as CBOR encoding and COSE_Key formats, as well as further parameters that can be passed to the `.create()` and `.get()` functions.

We would like to encourage developers to start experimenting with WebAuthn, and allow users to secure their logins with second factors, as the API becomes available. We are not aware of any WebAuthn-U2F polyfill libraries at the time of writing, but hope that these will be available soon. If you have seen something promising, please let us know in the comments.

It is very exciting to bring standardized two-factor authentication to the web; public-key cryptography already protects our data as it travels the Internet via the TLS protocol, and now we can use it to make phishing a lot, lot harder. Give WebAuthn a try in Firefox Nightly!



A final note about testing

Web Authentication is a powerful feature. As such, it can only be used in Secure Contexts, and if used in a frame, only when all of the frames are from the same origin as the parent document. This means that you are likely to encounter security errors when experimenting with it on some popular testing websites (such as jsfiddle.net).

About J.C. Jones

Keeping people safe on the 'net. Cryptography Engineering lead for Firefox.

https://tacticalsecret.com/
@JamesPugJones

More articles by J.C. Jones...

About Tim Taubert

Security Engineer working on Firefox and NSS.

https://timtaubert.de/
@ttaubert

More articles by Tim Taubert...

9 comments

T

Totally unclear what runs in the server, and what in the client.
Can you provide a full worked example, including both sides ?

January 17th, 2018 at 04:33

> **J.C. Jones** AUTHOR
>
> Sorry for the confusion! As for full open-source implementations, we're aware of google/webauthndemo, but it doesn't appear to be fully caught up to the current state of the specification, so it doesn't work yet in Firefox. It looks like there's activity happening, though. That repository is also a good

> example of the split of client/server responsibility when you run through the code, even if the code isn't quite finished.
>
> January 17th, 2018 at 06:27

Eli F.

Duo Lab's put together a demonstration app here: https://github.com/duo-labs/webauthn#webauthn-demo
I can't speak to how fully it implements the specification but it looks useful for exploring.

January 22nd, 2018 at 05:24

desar

The sample code in this blog isn't very helpful.
The W3C draft already has the sample code. Check chapter 12.
http://www.w3.org/TR/webauthn/

You got the credential, so what. You need to check/verify it.
The W3C draft doesn't have the sample code but it has the spec.
If you have time you should check it out.

Look at webauthn.bin.coffee for sample on how to verify
the credential on client side (browser). Checking on client
is not ideal, this only a POC/sample of course.
https://webauthn.bin.coffee/
https://github.com/jcjones/webauthn.bin.coffee

The spec has more than U2F, but current sample are focusing on that.
Since security key is easy to get/make.

For client server implementation vendor is working on their proprietary code I guess.
But, Duo Lab's sample above is worth checking. Hope more open source community effort for this
to be success.

BTW, if anyone interested, Chrome also had working beta today.
https://groups.google.com/a/fidoalliance.org/forum/#!topic/fido-dev/GH7AahxrE8o

http://webauthndemo.appspot.com/

January 24th, 2018 at 17:25

> **kaiju**
>
> I wrote Duo Labs' blog post and code repo. If you have any questions about the how the server side code works, feel free to message me at https://twitter.com/codekaiju
>
> You can also check out the repo's code in production at https://webauthn.io
>
> January 31st, 2018 at 14:39

James Ahern

Thanks for the great article. I believe that Mozilla will be releasing this with Firefox 60. Will the security.webauth.webauthn flag remain and/or will it be true by default?

January 30th, 2018 at 07:45

> **J.C. Jones** AUTHOR
>
> The flag will remain, but is being set `true` by default. Thanks!
>
> January 30th, 2018 at 08:46
>
> > **James Ahern**
> >
> > Really appreciate you taking the time to reply. Thanks, James
> >
> > January 31st, 2018 at 08:50

Wellington Torrejais da Silva

Nice, very useful.

February 9th, 2018 at 07:58

Comments are closed for this article.