Enabling Strong Authentication with WebAuthn

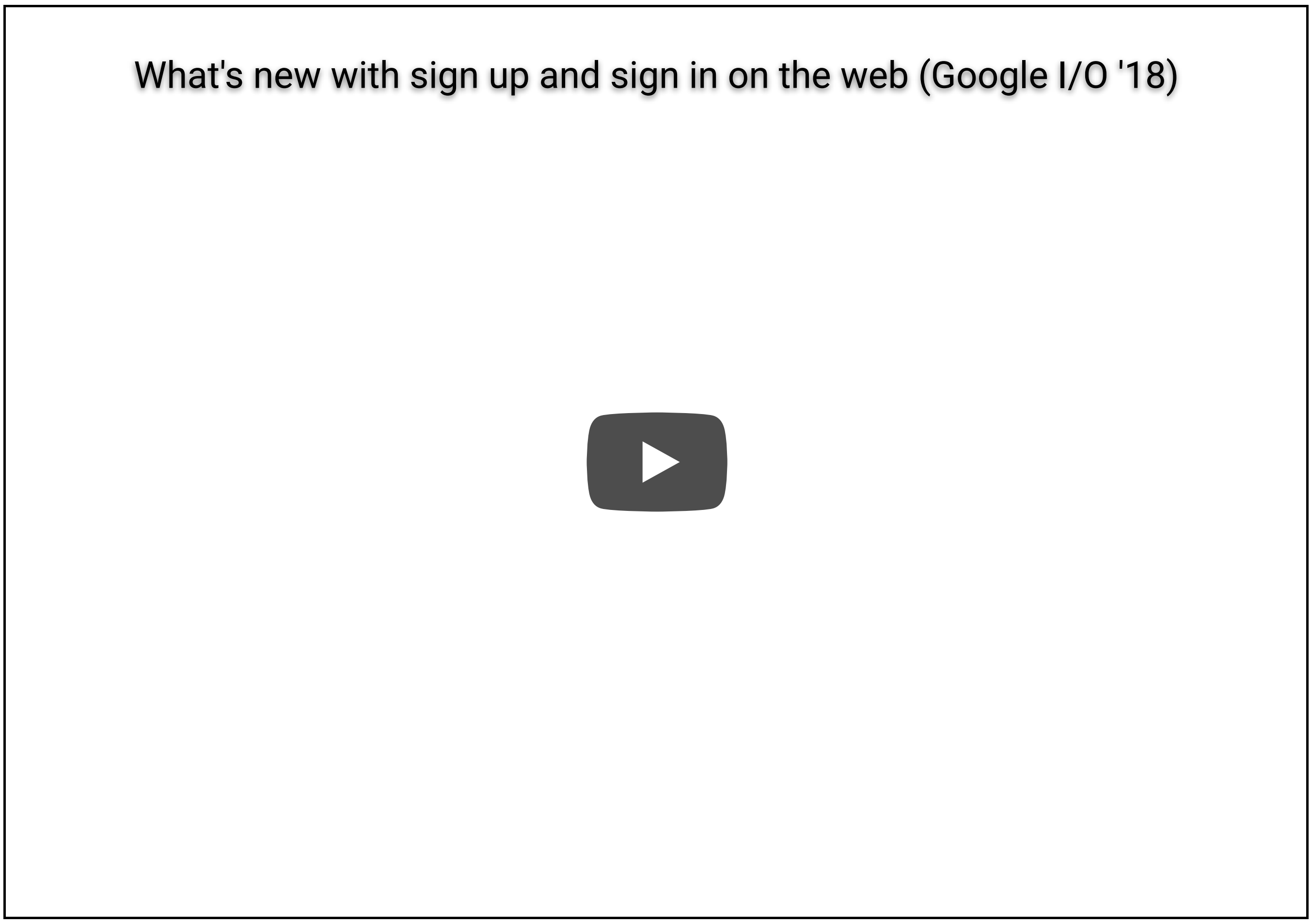
The problem Phishing is the #1 security problem on the web: 81% of hacking-related account breaches last year leveraged weak or stolen passwords. The industry's collective response to this problem has been multi-factor authentication, but implementations are fragmented and most still don't adequately address phishing. We have been working with the FIDO Alliance since 2013 and, more recently, with the W3C to implement a standardized phishing-resistant protocol that can be used by any Web application. What is WebAuthn? The Web Authentication API gives Web applications user-agent-mediated access to authenticators – which are often hardware tokens accessed over USB/BLE/NFC or modules built directly into the platform – for the purposes of generating and challenging application-scoped (eTLD+k) public-key credentials. This enables a variety of use-cases, such as: Low friction and phishing-resistant 2FA (to be used in conjunction with a password) Passwordless, biometrics-based re-authorization Low friction and phishing-resistant 2FA without a password (to be used for passwordless accounts) The API is on track to be implemented by most major browsers, and is intended to both simplify the UI encountered when having to prove your identity online and significantly reduce phishing. WebAuthn extends the Credential Management API and adds a new credential type called `PublicKeyCredential`. WebAuthn abstracts the communication between the browser and an authenticator and allows a user to: Create and register a public key credential for a website Authenticate to a website by proving possession of the corresponding private key Authenticators are devices that can generate private/public key pairs and gather consent. Consent for signing can be granted with a simple tap, a successful fingerprint read, or by other methods as long as they comply with FIDO2 requirements (there's a certification program for authenticators by the FIDO Alliance). Authenticators can either be built into the platform (such as fingerprint scanners on smartphones) or attached through USB, Bluetooth Low Energy (BLE), or Near-Field Communication (NFC). How it works Creating a key pair and registering a user When a user wants to register a credential to a website (referred to by WebAuthn as the "relying party"): The relying party generates a challenge. The relying party asks the browser, through the Credential Manager API, to generate a new credential for the relying party, specifying device capabilities, e.g., whether the device provides its own user authentication (with biometrics, etc). After the authenticator obtains user consent, the authenticator generates a key pair and returns the public key and optional signed attestation to the website. The web app forwards the public key to the server. The server stores the public key, coupled with the user identity, to remember the credential for future authentications.

```
let credential = await navigator.credentials.create({ publicKey: {
  challenge: Uint8Array(32) [117, 61, 252, 231, 191, 241,…]
  rp: { id: "acme.com", name: "ACME Corporation" },
  user: {
    id: Uint8Array(8) [79, 252, 83, 72, 214, 7, 89, 26]
    name: "jamiedoe",
    displayName: "Jamie Doe"
  },
  pubKeyCredParams: [ {type: "public-key", alg: -7} ]
}});
```

Warning: Attestation provides a way for a relying party to determine the provenance of an authenticator. Google strongly recommends that relying parties not attempt to maintain whitelists of authenticators. Authenticating a user When a website needs to obtain proof that it is interacting with the correct user: The relying party generates a challenge and supplies the browser with a list of credentials that are registered to the user. It can also indicate where to look for the credential, e.g., on a local built-in authenticator, or on an external one over USB, BLE, etc. The browser asks the authenticator to sign the challenge. If the authenticator contains one of the given credentials, the authenticator returns a signed assertion to the web app after receiving user consent. The web app forwards the signed assertion to the server for the relying party to verify. Once verified by the server, the authentication flow is considered successful.

```
let credential = await navigator.credentials.get({ publicKey: {
  challenge: Uint8Array(32) [139, 66, 181, 87, 7, 203,…]
  rpId: "acme.com",
  allowCredentials: [{
    type: "public-key",
    id: Uint8Array(80) [64, 66, 25, 78, 168, 226, 174,…]
  }],
  userVerification: "required",
}});
```

Try WebAuthn yourself at https://webauthndemo.appspot.com/. What's ahead? Chrome 67 beta ships with support for `navigator.credentials.get({publicKey: ...})` and `navigator.credentials.create({publicKey:... })` and enables using U2F/CTAP 1 authenticators over USB transport on desktop. Upcoming releases will add support for more transports such as BLE and NFC and the newer CTAP 2 wire protocol. We are also working on more advanced flows enabled by CTAP 2 and WebAuthn, such as PIN protected authenticators, local selection of accounts (instead of typing a username or password), and fingerprint enrollment. Note that Microsoft Edge also supports the API and Firefox will be supporting WebAuthn as of Firefox 60. Resources We are working on more detailed documentation: WebAuthnDemo relying party example implementation Analysis of WebAuthn article by Adam Langley The session "What's new with sign-up and sign-in on the web" at Google I/O 2018 covered WebAuthn. rss_feed Subscribe to our RSS or Atom feed and get the latest updates in your favorite feed reader!


What's new with sign up and sign in on the web (Google I/O '18)

The problem

Phishing is the #1 security problem on the web: 81% of hacking-related account breaches last year leveraged weak or stolen passwords. The industry's collective response to this problem has been multi-factor authentication, but implementations are fragmented and most still don't adequately address phishing. We have been working with the FIDO Alliance since 2013 and, more recently, with the W3C to implement a standardized phishing-resistant protocol that can be used by any Web application.

What is WebAuthn?

The Web Authentication API gives Web applications user-agent-mediated access to authenticators – which are often hardware tokens accessed over USB/BLE/NFC or modules built directly into the platform – for the purposes of generating and challenging application-scoped (eTLD+k) public-key credentials. This enables a variety of use-cases, such as:

- Low friction and phishing-resistant 2FA (to be used in conjunction with a password)
- Passwordless, biometrics-based re-authorization
- Low friction and phishing-resistant 2FA **without** a password (to be used for passwordless accounts)

The API is on track to be implemented by most major browsers, and is intended to both simplify the UI encountered when having to prove your identity online and significantly reduce phishing.

WebAuthn extends the Credential Management API and adds a new credential type called `PublicKeyCredential`. WebAuthn abstracts the communication between the browser and an authenticator and allows a user to:

1. Create and register a public key credential for a website
2. Authenticate to a website by proving possession of the corresponding private key

Authenticators are devices that can generate private/public key pairs and gather consent. Consent for signing can be granted with a simple tap, a successful fingerprint read, or by other methods as long as they comply with FIDO2 requirements (there's a certification program for authenticators by the FIDO Alliance). Authenticators can either be built into the platform (such as fingerprint scanners on smartphones) or attached through USB, Bluetooth Low Energy (BLE), or Near-Field Communication (NFC).

How it works

Creating a key pair and registering a user

When a user wants to register a credential to a website (referred to by WebAuthn as the "relying party"):

1. The relying party generates a challenge.
2. The relying party asks the browser, through the Credential Manager API, to generate a new credential for the relying party, specifying device capabilities, e.g., whether the device provides its own user authentication (with biometrics, etc).
3. After the authenticator obtains user consent, the authenticator generates a key pair and returns the public key and optional signed attestation to the website.
4. The web app forwards the public key to the server.
5. The server stores the public key, coupled with the user identity, to remember the credential for future authentications.

```
let credential = await navigator.credentials.create({ publicKey: {
  challenge: Uint8Array(32) [117, 61, 252, 231, 191, 241,…]
  rp: { id: "acme.com", name: "ACME Corporation" },
  user: {
    id: Uint8Array(8) [79, 252, 83, 72, 214, 7, 89, 26]
    name: "jamiedoe",
    displayName: "Jamie Doe"
  },
  pubKeyCredParams: [ {type: "public-key", alg: -7} ]
}});
```

Warning: Attestation provides a way for a relying party to determine the provenance of an authenticator. **Google strongly recommends that relying parties not attempt to maintain whitelists of authenticators.**

Authenticating a user

When a website needs to obtain proof that it is interacting with the correct user:

1. The relying party generates a challenge and supplies the browser with a list of credentials that are registered to the user. It can also indicate where to look for the credential, e.g., on a local built-in authenticator, or on an external one over USB, BLE, etc.
2. The browser asks the authenticator to sign the challenge.
3. If the authenticator contains one of the given credentials, the authenticator returns a signed assertion to the web app after receiving user consent.
4. The web app forwards the signed assertion to the server for the relying party to verify.
5. Once verified by the server, the authentication flow is considered successful.

```
let credential = await navigator.credentials.get({ publicKey: {
  challenge: Uint8Array(32) [139, 66, 181, 87, 7, 203,…]
  rpId: "acme.com",
  allowCredentials: [{
    type: "public-key",
    id: Uint8Array(80) [64, 66, 25, 78, 168, 226, 174,…]
  }],
  userVerification: "required",
}});
```

Try WebAuthn yourself at https://webauthndemo.appspot.com/.

What's ahead?

Chrome 67 beta ships with support for **`navigator.credentials.get({publicKey: ...})`** and **`navigator.credentials.create({publicKey:... })`** and enables using U2F/CTAP 1 authenticators over USB transport on desktop.

Upcoming releases will add support for more transports such as BLE and NFC and the newer CTAP 2 wire protocol. We are also working on more advanced flows enabled by CTAP 2 and WebAuthn, such as PIN protected authenticators, local selection of accounts (instead of typing a username or password), and fingerprint enrollment.

Note that Microsoft Edge also supports the API and Firefox will be supporting WebAuthn as of Firefox 60.

Resources

We are working on more detailed documentation:

- WebAuthnDemo relying party example implementation
- Analysis of WebAuthn article by Adam Langley

The session "What's new with sign-up and sign-in on the web" at Google I/O 2018 covered WebAuthn.

rss_feed Subscribe to our RSS or Atom feed and get the latest **updates** in your favorite feed reader!