FIDO beyond the browser

Joost van Dijk

FOSDEM - 3 Feb 2023
Overview

- What is FIDO?
- FIDO Security Keys
- Device attestation and the FIDO metadata service
- libfido2
- FIDO2 extensions
- Example use cases:
  - Authentication: pam-u2f, ssh
  - Signing: git commit signing
  - Encryption: LUKS2 disk encryption
  - Storage: SSH certificates
FIDO2

- Specifications:
  - CTAP - using a FIDO authenticator from a client (e.g. a browser)
  - Webauthn: API for using FIDO credentials in web applications
CTAP and Webauthn (simplified)

```
result = verify(p, response, challenge) +origin
```

```
response = sign(k, challenge) +origin
```
FIDO Security Keys: roaming authenticators
CTAP: operations (simplified)

<table>
<thead>
<tr>
<th>RP user store</th>
<th>Client</th>
<th>Authenticator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uID</strong></td>
<td><strong>credId</strong></td>
<td><strong>pub</strong></td>
</tr>
<tr>
<td>jdoe</td>
<td>0xa1b2</td>
<td>0x41…</td>
</tr>
</tbody>
</table>

```plaintext
makeCredential(rpId, uID, cData, …)
```

- `credId`, `pub`, `signature`, …

```
(priv, pub) = genKeyPair()
credId = genCredId(…)
map {credId, uID, rpId} → priv
signature = sign(…)
```

```
verify(resp, signature)
store[uID] = {credId, pub}
```

```
{credId, pub} = store[uID]
```

```
verify(pub, resp, signature)
```

```
getAssertion(credID, rpId, cData, …)
```

- `{ credId, pub, signature, … }

```
... getAssertion(credId, rpId, cData, …)
```

```
{ signature, … }
```

```
priv = lookup {credID}
signature = sign(priv, rpId, cData…)
```
Attestation and Metadata

- Attestation provides verifiable evidence as to the authenticator’s origin
- Based on a hardware attestation key and certificate
- Use FIDO Alliance Metadata Service to determine provenance
- Implement Allow/Deny lists to filter Authenticators
- Typically used in high-assurance (enterprise) use cases
Metadata example

- **aaguid**
  (Authenticator unique ID)

- **keyProtection**
  e.g. `secure_element`

- **transports**
  e.g. `usb`

- **status**
  (certification level)
FIDO Metadata Service

- retrieve FIDO metadata JWT:
  ```
  curl -Ls https://mds3.fidoalliance.org/ --output md.jwt
  ```
- extract certificate chain from JWT header (skipped):
  ```
  [mds.pem, intermediates.pem]
  ```
- verify certificate chain:
  ```
  openssl verify -CAfile cacerts.pem -untrusted intermediates.pem mds.pem
  ```
- verify JWT (using smallstep CLI):
  ```
  cat md.jwt | step crypto jwt verify --key mds.pem --alg RS256 --subtle \\
  | jq .payload > md.json
  ```
- parsing JSON (using jq):
  ```
  cat md.json | jq -r '.entries[] | select(.aaguid) | .metadataStatement \\
  | [.aaguid,.description] | @tsv'
  ```
- example output:
  ```
  ee882879-721c-4913-9775-3dfcce97072a YubiKey 5 Series
  ```
libfido2

- [https://github.com/Yubico/libfido2](https://github.com/Yubico/libfido2)
- Library and command-line tools
- Communicate with a FIDO device over USB or NFC
- Supports both FIDO U2F (CTAP 1) and FIDO2 (CTAP 2) protocols
- Works on Linux, macOS, Windows, OpenBSD, and FreeBSD
- Bindings for [.NET, Go, Perl, Rust](https://github.com/Yubico/python-fido2)
- Also available:
  - [https://github.com/Yubico/python-fido2](https://github.com/Yubico/python-fido2)
Example: pam-u2f

ftp

sshd

login

sudo

Pluggable Authentication Module (PAM)

smart card

NTLM

Kerberos

FIDO2/U2F
pam-u2f

- Pluggable Authentication Module (PAM) for U2F and FIDO2
- Require a FIDO credential on a security key when authenticating to a service
- Included in many distributions by default
- See also
  - Code: https://github.com/Yubico/pam-u2f
  - Docs: https://developers.yubico.com/pam-u2f/
**example: require MFA for sudo**

- **Install:**
  
  ```bash
  apt install libpam-u2f
  ```

- **Register a FIDO credential:**
  
  ```bash
  pamu2fcfg > ~/.config/fido_keys
  ```

- **Configure `/etc/pam.d/sudo`:**
  
  ```bash
  @include common-auth
  auth required pam_u2f.so cue authfile=.config/fido_keys
  ```

- **Test:**
  
  ```bash
  $ sudo echo Ok
  [sudo] password for joost:
  Please touch the device.
  Ok
  $
  ```
SSH keys backed by FIDO Security Keys

- Private key: $k$
- Public key: $p$
- Authenticator

CTAP Challenge → ssh

- Flags
- Key Handle ($=\text{credId}$)
- Private Key $k$

SSH Challenge → sshd

- Response

**ID** | **RP ID** | **private_key**
---|---|---
key_handle | “ssh:” | 0xdeadbeef...

**id_ecdsa**

- "sk-ecdsa-sha2-nistp256@openssh.com"
- nistp256
- 0
- “ssh:” (rpID)
- flags
- key_handle ($=\text{credId}$)

**id_ecdsa.pub**

- "sk-ecdsa-sha2-nistp256@openssh.com"
- nistp256
- 0
- “ssh:” (rpID)
SSH example: generating keys

- Generate hardware backed ECDSA SSH key pair:
  
  ```bash
  $ ssh-keygen -t ecdsa-sk -f ./id -N ""
  Generating public/private ecdsa-sk key pair.
  You may need to touch your authenticator to authorize key generation.
  Enter PIN for authenticator: ********
  You may need to touch your authenticator again to authorize key generation.
  Your identification has been saved in ./id
  Your public key has been saved in ./id.pub
  The key fingerprint is:
  SHA256:eGO0VlLwvP4rWPMdRYprfQ8F1tmnTQJWGq8WjV97pHk user@machine
  The key's randomart image is:
  ...
  
  - Idem, using ED25519:
    
    ```bash
    $ ssh-keygen -t ed25519-sk -f ./id2 -N ""
    ```
SSH example: authenticating to GitHub

- Register your public key at GitHub:
  [https://github.com/settings/keys](https://github.com/settings/keys)

- Authenticate to a git repo over SSH:
  
  ```sh
  $ git clone git@github.com:user/repo.git
  Cloning into 'repo'...
  Confirm user presence for key ED25519-SK SHA256:FpybChVXHU/…+8x0
  User presence confirmed
  ...
  Resolving deltas: 100% (6/6), done.
  ```
SSH example: signing files

- Generate hardware backed SSH key pair:
  
  ```
  ssh-keygen -t ecdsa-sk -f ./id1 -N ""
  ```

- Sign (in file namespace):
  
  ```
  ssh-keygen -Y sign -n file -f ./id1 datafile
  ```
  (generates signature in datafile.sig)

- Verify:
  
  ```
  ssh-keygen -Y verify -n file -s datafile.sig \
  -f ./allowed_signers -I user1@example < datafile
  ```

- Trusted public keys in allowed_signers:
  
  ```
  # user        key type                           pubkey
  user1@example sk-ecdsa-sha2-nistp256@openssh.com AAAAIInNr...3No0g==
  ```
Git example: commit signing

- Generate hardware backed SSH key pair:
  ```
  ssh-keygen -t ecdsa-sk -f ~/.ssh/id1 -N ""
  ```

- Configure git:
  ```
  git config gpg.format ssh
  git config user.signingKey ~/.ssh/id1
  git config gpg.ssh.allowedSignersFile ~/.ssh/allowed_signers
  ```

- Sign a commit:
  ```
  git commit -S -m 'initial import'
  ```

- Sign a tag:
  ```
  git tag -s v1.0 -m 'signed v1.0 tag'
  ```

- Show and verify signatures:
  ```
  git log --oneline --show-signature
  ```
hmac-secret extension

- Goal: retrieve a symmetric secret from the authenticator
- On Windows: ensures you can sign-in to your device when it's off-line or in airplane mode
- Part of the secret is held by the client (the salt) and the other part of the secret is attached to the credential on the key.
- Combining the salt and the credential will produce the secret.
- Maintain different secrets on different systems.
CTAP: hmac-secret (simplified)

Client

\[\text{makeCredential}(..., \text{createSecret}=true)\]
\[\{..., \text{createSecret}=true\}\]

\[
\begin{array}{c}
\text{key} = \text{random()}\\
\text{map} \{\text{credId},...,\text{key}\} \rightarrow \text{priv}
\end{array}
\]

\[
\begin{array}{c}
\text{salt} = \text{random()}\\
\text{store}[\text{uID}] = \{\text{credId},\text{pub},\text{salt}\}
\end{array}
\]

\[
\begin{array}{c}
\{\text{credId},\text{pub},\text{salt}\} = \text{store}[\text{uID}]
\end{array}
\]

\[
\begin{array}{c}
\text{getAssertion}(..., \text{salt})
\end{array}
\]

\[
\begin{array}{c}
\{..., \text{output}\}
\end{array}
\]

\[\text{output} = \text{hmac}_\text{sha256}(\text{key}, \text{salt})\]

\[
\begin{array}{c}
\text{encryption_key} = \text{KDF}(\text{output})
\end{array}
\]
hmac-secret: example

- Generate a resident (-r) FIDO credential with hmac-secret (-h):
  cred -r -h -k ./pubkey -P myPIN /dev/hidraw1

- Generate some random salt:
  cat /dev/urandom | head -c32 > ./salt

- Generate a secret key using our FIDO credential and our salt:
  assert -p -h ./secret -s ./salt -P myPIN ./pubkey /dev/hidraw1

- Some data to encrypt:
  echo some secret data > plaintext

- Encrypt our data (safe to delete secret key and plaintext)
  openssl enc -aes256 -e -pass file:./secret \
  -in ./plaintext -out ./ciphertext

- Decrypt our data (may need to regenerate secret key first)
  openssl enc -aes256 -d -pass file:./secret -in ./ciphertext
hmac-secret application: LUKS

- LUKS (Linux Unified Key Setup) disk encryption
- systemd v248+ supports unlocking LUKS2 volumes using FIDO2 security keys
- derives the disk decryption key from a FIDO credential using the hmac-secret extension

- Enroll a FIDO2 authenticator to a LUKS2 encrypted volume:
  systemctl-cryptenroll -fido2-device=/dev/hidraw1 /dev/sda1
largeBlobs extension

- store arbitrary data associated with credentials
- size ~1KB
- Requires a CTAP 2.1 security key
- Example use: store SSH certificates
example: store SSH certificates

- **store (-S)** an SSH certificate as a largeBlob:
  ```
  fido2-token -S -b -n ssh: ./id_ecdsa-cert.pub /dev/hidraw1
  ```

- **list (-L)** largeBlobs:
  ```
  fido2-token -L -b /dev/hidraw1
  ```

- **retrieve (-G)** a largeBlob and save it to file:
  ```
  fido2-token -G -b -n ssh: out.blob /dev/hidraw1
  ```

- **delete (-D)** a largeBlob:
  ```
  fido2-token -D -b -n ssh: /dev/hidraw1
  ```
Device attestation example: SSH commit

- Prove that the key that signed a file (or git commit, or ...) was backed by a FIDO security key
- Produce an attestation statement when generating keys:

  ```
  ssh-keygen -t "ed25519-sk" -f ./sk -N "" \
  -O write-attestation=attest.bin -O challenge=challengefile
  ```

- Extract attestation certificate, signature, and data from `attest.bin`
- Verify attestation certificate using FIDO MDS
- Verify attestation signature over data using attestation certificate
- Lookup AAGUID in FIDO MDS to find security key make and model
Conclusion

- FIDO2 comprises open standards CTAP and Webauthn
- FIDO2 security keys can be used as an alternative to traditional hardware security devices such as smart cards
  - More affordable
  - More user-friendly
  - More privacy-preserving
  - Primarily for authentication, but also for signing and encryption
  - Easy to integrate using Open Source software libraries
- Need help? Reach out!
Resources

- Developer Program: [https://dev.yubi.co/](https://dev.yubi.co/)
- FIDO dev forum: [https://groups.google.com/a/fidoalliance.org/g/fido-dev](https://groups.google.com/a/fidoalliance.org/g/fido-dev)
- libfido2: [https://github.com/Yubico/libfido2](https://github.com/Yubico/libfido2)
- FIDO Metadata:
  - [https://fidoalliance.org/metadata/](https://fidoalliance.org/metadata/)
  - [https://opotonniee.github.io/fido-mds-explorer/](https://opotonniee.github.io/fido-mds-explorer/)
- SSH support for security keys:
- PAM u2f: [https://github.com/Yubico/pam-u2f](https://github.com/Yubico/pam-u2f)
- LUKS2 and cryptsetup:
Questions?