Universal 2\textsuperscript{nd} Factor

2015-02-01
FOSDEM Security devroom
Simon Josefsson
simon@yubico.com
What is U2F?
Through 20 years of effort, we've successfully trained everyone to use passwords that are hard for humans to remember, but easy for computers to guess.

- **~28 bits of entropy**
  - Numerical: $2^{20} = 3$ days at 1000 guesses/sec
  - Difficulty to guess: EASY
  - Difficulty to remember: HARD

- **~44 bits of entropy**
  - Combining four random common words
  - Numerical: $2^{44} = 550$ years at 1000 guesses/sec
  - Difficulty to guess: HARD
  - Difficulty to remember: YOU'VE ALREADY MEMORIZED IT

**Tr0ub4dor & 3**
- Unique (non-gibberish) base word
- Common substitutions
- Numerical
- Punctuation
- Order unknown

**Correct horse battery staple**
- Four random common words

Was it trombone? No, Troubadour, and one of the Os was a zero?
And there was some symbol...

Plausible attack on a weak remote web service: yes, cracking a stolen hash is faster, but it's not what the average user should worry about.
The U2F solution

One device, many services

Easy: Insert and touch button

Safe: Unphishable Security
Pre-History of U2F: Gnubby

Yubico designed a precursor to U2F with Google and NXP. Deployed to Google staff around the world.

To reach mass market, standardization and multiple vendors are needed. During 2012 the FIDO Alliance started working on U2F.
Over 150 members
What is this U2F protocol?

Core idea: Standard public key cryptography

- User's device mints new key pair, "registers" public key and key-handle with server
- Key handle contain data to restore private key on device
- Server provides key-handle and asks user's device to sign data to verify the user
- One device, many services - “Bring Your Own Authenticator”

Design considerations

- **Privacy**: Site-specific keys, no unique device ID
- **Security**: No phishing or man-in-the-middle, no soft private keys
- **Trust**: User decides what authenticator to use
- **Pragmatics**: Affordable today
- **Usability**: No delays, fast crypto on device, no driver installs
Think:

Driverless smartcard for the modern consumer web, plus privacy
USB today, the world tomorrow

Hardware separation important! Software in complex hosts too fragile → keys stolen on 0day vuln.
Demo
**Authentication**

**U2F Device**

- retrieve: key $k_{priv}$ from handle $h$; counter++

- $h, a;$ challenge, origin, channel id, etc.

**Browser - FIDO Client**

- check app id

- $c$

- counter, signature(a,c,counter)

- $s$

**Relying Party**

- retrieve: key $k_{pub}$ from handle $h$

- check: signature using key $k_{pub}$

- counter, c, s

- set cookie
U2F Authentication JSON blobs

Server sends: 

```
{ "keyHandle": "yQ_cxLOEDDrQ1rGesE249-QYNjGoNWpY2QRSQzE9p0qQZNk2i3Z6ioYAAumOZnJQhuQDJ2VVtOcUD85kYRdjuQ",
"version": "U2F_V2",
"challenge": "cDftdgcY3SOYMaKPq6JFt0nmpFACTzuJ5EbRr-VTnxA",
"appld": "http://example.org" }
```

Client responds: 

```
{ "signatureData": 
  "AQAAADMwRglhAKCAGKKDcZe1Rt4HdOnD2JkF5yU711AxjngH-_dW9-e5AiEAYlw5kzYKRg2rSI0JU1zsJibF3MIWtOCXGv1h4KazCys=",
  "clientData": 
  "eyAiY2hhbGxlbdIlJjogImNEZnRkJ2Nz1NPWU1hs1BxNkpGdBubXBGQUNUWnVKNUViUnI	tVIRueEElCAib3JpZ2IuljogImh0dHA6XC9cL2V4YW1wbGUub3JnliwglR5cCI6ICJuYXZpZ2F0b3luaWQuZ2V0QXNzZXJ0aW9uIiB0"
  "keyHandle": "yQ_cxLOEDDrQ1rGesE249-QYNjGoNWpY2QRSQzE9p0qQZNk2i3Z6ioYAAumOZnJQhuQDJ2VVtOcUD85kYRdjuQ" }
```
USB HID Authenticate
Registration

U2F Device

Browser - FIDO Client

Relying Party

generate:
key $k_{\text{pub}}$
key $k_{\text{priv}}$
handle $h$

check
app id
c

app id, challenge

$\text{check} \ app \ \text{id}$

$c$

$\text{app id}, \ \text{challenge}$

$c, k_{\text{pub}}, h, \text{attribution cert, signature}(a,c,k_{\text{pub}},h)\$

$c, k_{\text{pub}}, h, \text{attribution cert, s}$

store:
key $k_{\text{pub}}$
handle $h$
for user

key $k_{\text{pub}}$

key $k_{\text{priv}}$

handle $h$
U2F Register JSON blobs

Server sends: { "challenge": "oVXT29EiA16cFFlQCzwPp-waGiMahl2WlevJXcFQCVCc", "version": "U2F_V2", "applId": "http:////example.org" }

Client responds: { "registrationData": "BQQ91soQ8zQlX-yBzGJtOWMvKbWPkIsOqA_1psdwK7fid03vAXcDreXFFgcYEaxI5dUyWcs3jiw67Z_D0KxZMTP2QMKp3MSzhAw60NaxnrBNuPfkGDYxqDVqWNkEUKmxPadKkGTZNot2eoqGAALpjmyUlbkAydlVbTnFA_OZGEXY7kweg...W_AMRED0ExAGowC0YQMvgbqWGZiZAiBUt00SBB1TTtFbwf4Lp1daS5L6gqMQxtiHlrHjZwFkw==", "clientData": "eyAiY2hhbGxlbmlljogIm9WWFQyOUVpQTE2Y0ZGSVFdendQcC13YUdpTWFoSTJXSWV2SlhjRIFDVmMiLCAib3JpZ2luljogImh0dHA6XC9cL2V4YW1wbGUub3JnliwgInR5cCI6ICJuYXZpZ2F0b3luaWQuZmluaXNoRW5yb2xsbWVudCIgfQ==" }
Registration: USB HID
**Application**

A set of functionality provided by a common entity (the application owner), and perceived by the user as belonging together. For example, *PayPal* is an application that allows users to pay for stuff.

**Facets**

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. Simplest case: facet id and application id is the same.

- For the Web, the Facet ID is the web origin, written as a URI without a path (e.g. `https://login.paypal.com`).

- For Android, the Facet ID is the URI `android:apk-key-hash:<hash-of-apk-signing-cert>`.

- For iOS, the Facet ID is the URI `ios:bundle-id:<ios-bundle-id-of-app>`.
What if I want to support U2F?

- Server/Browser: Call Javascript APIs
  - Send key handle in HTML/JavaScript to browser
- Server: Implement registration flow
  - Decide how to handle attestation certificates
  - Verify registration response
  - Store public key, key handle with user account
- Server: Implement login flow
  - Check username/password, look up key handle
  - Verify authentication response (origin, signature, counter, …)
- Relying Party: Check your account recovery flow
So many keys...

- Authentication public/private key
  - Unique for every RP
  - Generated during U2F Registration
  - Public key sent to RP during Registration
  - Key handle can be used to derive private key
    - Unlimited number of RPs on small device
  - Hard coded to ECDSA using NIST P.256 curve
So many keys...

- Device-unique symmetric secret
  - Unwrap/derive per-RP ECDSA key from key handle
  - Unique random key for every device
  - Yubico derives private key using HMAC-SHA256
Yubico’s U2F KeyHandle

- Key handle is nonce+MAC instead of encrypted
- Device can derive ECDSA private key from nonce and symmetric device secret
- MAC detects invalid key handle or malicious RP
So many keys...

- ECDSA attestation key (unique per batch)
  - Linked with device attestation certificate
  - Signs U2F Registration blobs
U2F attestation

- Proves what U2F device the user used
- X.509 Certificate with batch-unique key

Why batch-unique and not device-unique?

- Privacy: device-unique key permits conspiring RPs to link a physical key to particular user
- Common batch size could be 10k-100k (could be 1 breaking the privacy aspects)
Yubico U2F software

Our idea is to publish host and server libraries in common languages as FOSS code

- **C**: libu2f-host & libu2f-server
- **Java**: java-u2flib-server
- **PHP**: php-u2flib-server
- **Python**: python-u2flib-host & python-u2flib-server
U2F C Libraries

- github.com/Yubico/libu2f-\{server,host\}
- Portable C99 few dependencies (json, OpenSSL, HIDAPI)
- server: Generate U2F challenges and verify responses
- host: Parse challenges and talk USB to get responses
- Command line tool
Resources

Libraries, Plugins, Sample Code, Documentation  developers.yubico.com/U2F

U2F Protocol Specification  fidoalliance.org/specifications

Yubico U2F Demo Server - Test your U2F device here!  demo.yubico.com/u2f
Thank you!