Client Certificates Are Going Away

What now?

Garrett Wollman, TIG

May 2, 2016

Overview of this talk

1. Review of the current situation and how we got here

2. Our response to the deprecation of client certificate authentication in Web browsers

3. Help and input we need from you to make this transition

Three components to client certificate authentication

- TLS and HTTP protocol support for clients to (re)authenticate using a certificate (implemented in browsers, Web servers, and middleware/libraries)
- Online provisioning protocol (implemented by browsers and certificate authorities)
- CA implementation proper

HTTP over SSL/TLS (1)

- Support for client certificate authentication was added by Netscape in SSL 3.0
- Part of initial Diffie-Hellman key exchange: client cert is required immediately on opening connection
  - Private key used to sign client's Diffie-Hellman values
- Renegotiation protocol allows servers to redo key exchange after authorization failure
  - Original SSL renegotiation mechanism had a serious design flaw requiring emergency protocol update in 2013 (MitM vulnerability)
  - Google left renegotiation out of their "SPDY" protocol, which became HTTP/2.0
- Caching (on both sides) means user generally not re-prompted for choice of certificate once made

HTTP over SSL/TLS (2)

- A plethora of UI problems:
    - Is certificate store in browser or operating system?
    - How does the user select an appropriate certificate…?
    - …When they don't even know which server is asking for it!
    - Incomprehensible error messages
    - No way to revisit client cert selection (short of exiting browser) — *i.e.*, no "logout" button

- Most browser developers don't work for an enterprise that makes use of client certificates!

HTTP over SSL/TLS (3)

- When server wants a client certificate, it sends the client a list of acceptable client CAs
- This might include CAs of unrelated enterprises, tax authorities, banks, etc., potentially revealing PII
- Because of bad certificate selection UI, client may have no idea that they just gave their identity to a “bad guy” (cf. VZW “supercookie”)

HTTP over SSL/TLS (4)

- Even if the client is talking to a server they trust to know their identity, they might have multiple identities

- We have this problem: servers have the option of accepting both MIT and CSAIL client certificates

- If the client fails authorization because only one identity is authorized, no way to ask them to try a different one
    - Partial workaround: the "`SSLVerifyClient optional`" hack; reimplement certificate validation in application (doesn't work for static content, can't invoke renegotiation)

Client certificate provisioning (1)

- Two ways to do certificate provisioning:
  a) native code on every platform (doesn’t do anything for Firefox), or
  b) in-browser with HTML forms and `<KEYGEN>` element (doesn’t work for IE/Edge or now Chrome)

- Original provisioning protocol defined by a `README` file distributed with Netscape 4.0.

- Lots of problems with this: obsolete crypto (uses MD5 for proof-of-possession) and lack of policy functionality (*e.g.*, non-exportable private keys)

- This is the functionality that is going away most immediately

Client certificate provisioning (2)

- Client sends a certificate request to a CA, but then what?

- No binding between request (or response) and client identity: must be implemented ad-hoc on the CA side

- How does the client certificate get installed once signed by the CA?
  - In-browser vs. system certificate stores
  - Netscape protocol: magic media types used to trigger certificate installation

CA implementation (1)

- CSAIL’s CA implementation was written
  - by me, working alone
  - more than ten years ago
  - in a version of Ruby (1.8) that isn’t supported any more
  - using a templating system that was abandoned by its author and doesn’t work in modern Ruby

CA implementation (2)

- CSAIL’s CA also has a number of cryptographic weaknesses that would need to be addressed if we were going to keep it going
  SHA-1, predictable serial numbers, etc.

- We have periodically reviewed alternative client CA implementations to get us off this code base

- Effort to implement a new CA ≥ effort to implement alternative auth system for Web servers & apps
  In fact, much of the effort involved is the same

Desiderata

- Single sign-on: users are not constantly prompted for credentials
- Limited scope for exposure of primary credentials
- Potential to support new credential types, 2FA (*e.g.*, Fido Alliance U2F)
- Same solution works for both static-file Web resources (*e.g.*, `.htaccess` files on people.csail) and server-side applications (*e.g.*, calendar.csail)
- Expeditious access revocation
- Offline verification
- Higher-level scopes for authorization (*e.g.*, research group, appointment status)
  - Lots of `.htaccess` files with inaccurate, manually maintained lists of group members

Alternatives to client certs

1. Plain old username and password (a/k/a “HTTP Basic Auth”)
2. Kerberos (GSSAPI) with SPNEGO
3. SAMLv2
4. OpenID Connect

Sadly, all of these require online verification!

OpenID Connect

- Plethora of confusing terminology
- **N.B.:** OpenID Connect has nothing to do with the (now mainly historical) OpenID 2.0
- OIDC adds an identity layer on top of OAuth 2.0, using OAuth for authentication and application (“client”) authorization
- Identity service (“Userinfo”) is what provides user name, profile information, and other “claims” to the application

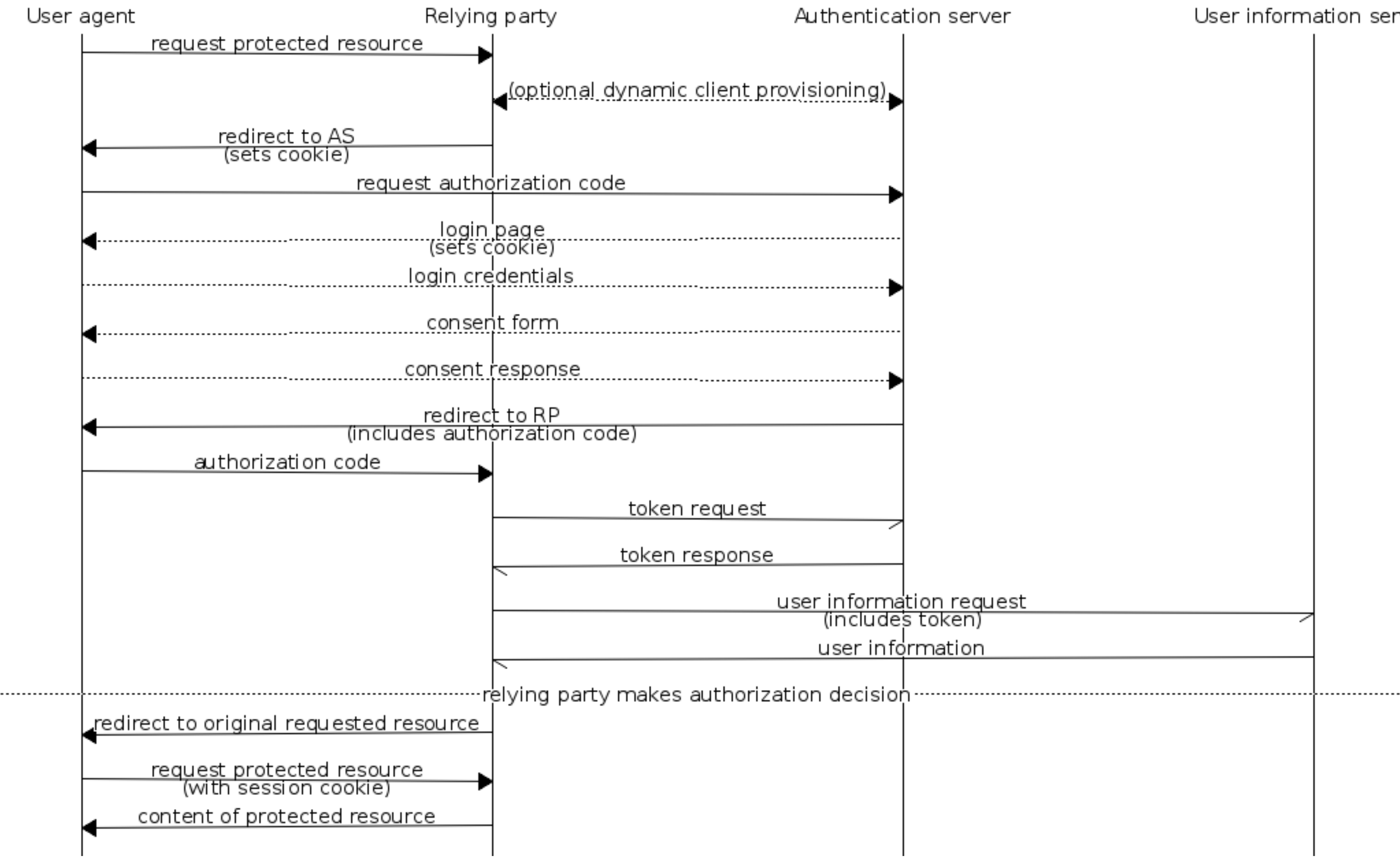
OIDC and our requirements

- Single sign-on: users are not constantly prompted for credentials
  - Yes (with some caveats related to token and auth cookie lifetime)
- Limited scope for exposure of primary credentials
  - Yes (only identity provider ever sees user's primary credentials)
- Potential to support new credential types, 2FA (*e.g.*, Fido Alliance U2F)
  - Yes (whatever we can implement in the identity provider)
- Same solution works for both static Web resources and server-side apps
  - Yes (Apache module `mod_auth_openidc`; libraries for applications in Perl/PHP/Python/Ruby/Java/Insert Your Favorite Language Here)
- Expeditious access revocation
  - Yes (with some caveats related to token and auth cookie lifetime)
- Offline verification
  - No
- Higher-level scopes for authorization
  - Yes (whatever we can implement in the identity provider, exposed as strings to client)

Demo

Parties to an OIDC protected transaction

- OpenID Connect Authentication Server (= OAuth 2.0 Authorization Server)
- Relying Party (called a “client” in OAuth/OIDC terminology)
- User Agent (most of the time, a browser)
- Userinfo Service (usually integral with Auth Server)


User agent
Relying party
Authentication server
User information ser
request protected resource
(optional dynamic client provisioning)
redirect to AS
(sets cookie)
request authorization code
login page
(sets cookie)
login credentials
consent form
consent response
redirect to RP
(includes authorization code)
authorization code
token request
token response
user information request
(includes token)
user information
relying party makes authorization decision
redirect to original requested resource
request protected resource
(with session cookie)
content of protected resource

(Not exactly) Greenspun’s Tenth Rule

Any sufficiently complicated network access control system contains an ad-hoc, informally specified, bug-ridden, slow implementation of half of Kerberos

– Not original to me

Where do we go from here?

- We are working on standing up an OIDC identity provider using MITREid Connect, which was developed under contract with the MIT Internet Trust Consortium
- We also need to figure out a transition plan
- Goal is to have this completed by the time new students arrive for the fall term

Transition issues

- What do we do about the hundreds, if not thousands, of existing `.htaccess` files?
- What sort of directory information should we make available for authorization decisions?
  - Supervisor/group membership
  - Appointment type and status
- How do we link up campus and CSAIL identities?
  - Do we even want or need to do this?
- Can we leverage "the big origins" for identity in the general case?
  - Current thinking: no, because all require per-client prior setup

Dealing with `.htaccess` files

- Apache 2.2 and 2.4 have completely different access-control mechanisms
- Apache 2.2's mechanism is too limited to make good use of OIDC, so 2.4 seems like a must (and desirable for other reasons anyway)
- Automatic translation is impractical
  - `SSLRequire <`*`boolean logic & regexps`*`>` vs. `Require claim <`*`string`*`>` with `RequireAll`/`RequireAny` containers for boolean logic
  - But we could mechanically identify `.htaccess` files that require human attention
- Fail open or fail closed?

Summary

- It’s not our fault, really
- We actually really like public-key authentication, and it has some great properties we wish the browser community valued
- Nonetheless, we’re going to be implementing OpenID Connect over the summer
- There are still a bunch of implementation choices to be made that we’d like your help with

Questions?