Week 2, Lecture 2

COMP6[48]43

Hamish Cox

Feb 26th, 2025

Lecture 1 Recaps

Yesterday we looked at how we typically authenticate users:

- Passwords
- MFA
- SSO

But how do we actually use that?

Let’s recap by writing some auth.

Demo: Actually implementing a login page.

HTTP is Stateless

Websites send a lot of HTTP requests.

Each new request **could** be on a new connection and/or from a new IP - maybe you changed WiFi networks or cell tower.

HTTP itself has no notion of a ‘connection’ - just a request and response.

Sessions

We want to maintain a ‘session’ - we want an easy way to indicate to the server that we have authenticated at some point in the past.

But we also need it to be easy for the browser to know what to store and send alongside all our requests.

So how do we send this session information to the server?

Just chuck it in a header:

```
GET / HTTP/1.1
Host: localhost
Logged-In: Yes
```

These ‘cookies’ are the standard way of sending information alongside every request your browser sends.

So how do we send this session information to the server?

Just chuck it in a header:

```
GET / HTTP/1.1
Host: localhost
Cookies: session=???
```

These ‘cookies’ are the standard way of sending information alongside every request your browser sends.

Where do these cookies come from?

They are sent with every request - the client could set them manually.

But typically they are set by the server in a response:

Where do these cookies come from?

They are sent with every request - the client could set them manually.

But typically they are set by the server in a response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 3853
Set-Cookie: session=???

<html> ... </html>
```

Where do these cookies come from?

They are sent with every request - the client could set them manually.

But typically they are set by the server in a response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 3853
Set-Cookie: session=???; Domain=quoccacorp.com; Max-Age=3600; Secure

<html> ... </html>
```

What do we put in cookies?

I've been writing `???` as the value of our cookies.

What do server typically put there?

How about this?

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 3853
Set-Cookie: session=someuser

<html> ... </html>
```

Cookies are Client-Side Input

You can edit cookies in a browser. This includes those extra properties we saw earlier (Domain, Max-Age, Secure, etc).

If someone other than the server can generate a cookie value that corresponds to a user, your authentication is insecure.

Tokens

A common way to represent/authenticate a session is the use of a ‘token’.

These are most commonly sent to the server via a cookie, but they may also be sent in request headers (e.g. when authenticating requests to an API, that isn’t going to be viewed with a browser).

Opaque/Server-Side Tokens

All you need for a good token is a way for the server to match it to a user, without a user being able to guess a token for another user.

Easiest option: a list of random tokens.

Opaque/Server-Side Tokens

All you need for a good token is a way for the server to match it to a user, without a user being able to guess a token for another user.

Easiest option: a list of random tokens.

Token	Username
qXN0TVAddYDKChxkrz94WCDuaOA	admin
ueChoCpy9Mke1XJW5OqPdkgjEzY	HamishWHC
mVB3DygFnsvHck5YOAxnzmFc8u0	melon
LE8oUVIKP9ynGA4mcCC6vvzbkkY	TrashPanda
8ISe586s8ralOz7YckFWjRN1ch4	ssparrow
...	...

Opaque/Server-Side Tokens

All you need for a good token is a way for the server to match it to a user, without a user being able to guess a token for another user.

Easiest option: a list of random tokens.

Token	Username	Expires At
qXN0TVAddYDKChxkrz94WCDuaOA	admin	2025-02-26
ueChoCpy9Mke1XJW5OqPdkgjEzY	HamishWHC	2025-02-28
mVB3DygFnsvHck5YOAxnzmFc8u0	melon	2025-02-27
LE8oUVIKP9ynGA4mcCC6vvzbkkY	TrashPanda	2025-03-02
8ISe586s8ralOz7YckFWjRN1ch4	ssparrow	2025-02-30
...	...	...

Opaque/Server-Side Tokens

All you need for a good token is a way for the server to match it to a user, without a user being able to guess a token for another user.

Easiest option: a list of random tokens.

Token	Username	Revoked At
qXN0TVAddYDKChxkrz94WCDuaOA	admin	2025-02-23
ueChoCpy9Mke1XJW5OqPdkgjEzY	HamishWHC	NULL
mVB3DygFnsvHck5YOAxnzmFc8u0	melon	2025-02-24
LE8oUVIKP9ynGA4mcCC6vvzbkkY	TrashPanda	2025-02-14
8ISe586s8ralOz7YckFWjRN1ch4	ssparrow	NULL
...	...	...

Signed Tokens

Don’t have a database or storage option of any kind, or don’t want the performance hit of a database lookup for every request?

Well all we need for a good token is a way to ensure someone who has it, can’t edit it...

Just sign it.

Signed Tokens

Don’t have a database or storage option of any kind, or don’t want the performance hit of a database lookup for every request?

Well all we need for a good token is a way to ensure someone who has it, can’t edit it...

Just sign it.

Signed Tokens

Don’t have a database or storage option of any kind, or don’t want the performance hit of a database lookup for every request?

Well all we need for a good token is a way to ensure someone who has it, can’t edit it...

Just sign and verify it.

Flask Tokens

Flask offers a simple `session` interface that allows you to add data to a session that is then signed and saved as a cookie.

```
@app.route("/login", method=["POST"])
def login():
    # verify username and password ...

    session["user_id"] = user.id

    return redirect("/")
```

It's basically a dictionary you can manipulate as you like.

Flask Tokens

A Flask token is made up of three Base64 sections separated by dots.

```
eyJ1c2VybmFtZSI6ImFkbWluIn0.Z71j0g.LMQK7W3n58NpT1tD6at7bfWC39M
```

No, I don’t know what the signature stuff is :)

Flask Tokens

A Flask token is made up of three Base64 sections separated by dots.

`base64('{"username": "admin"}').<signature-stuff>.<signature-stuff>`

No, I don’t know what the signature stuff is :)

Flask Tokens

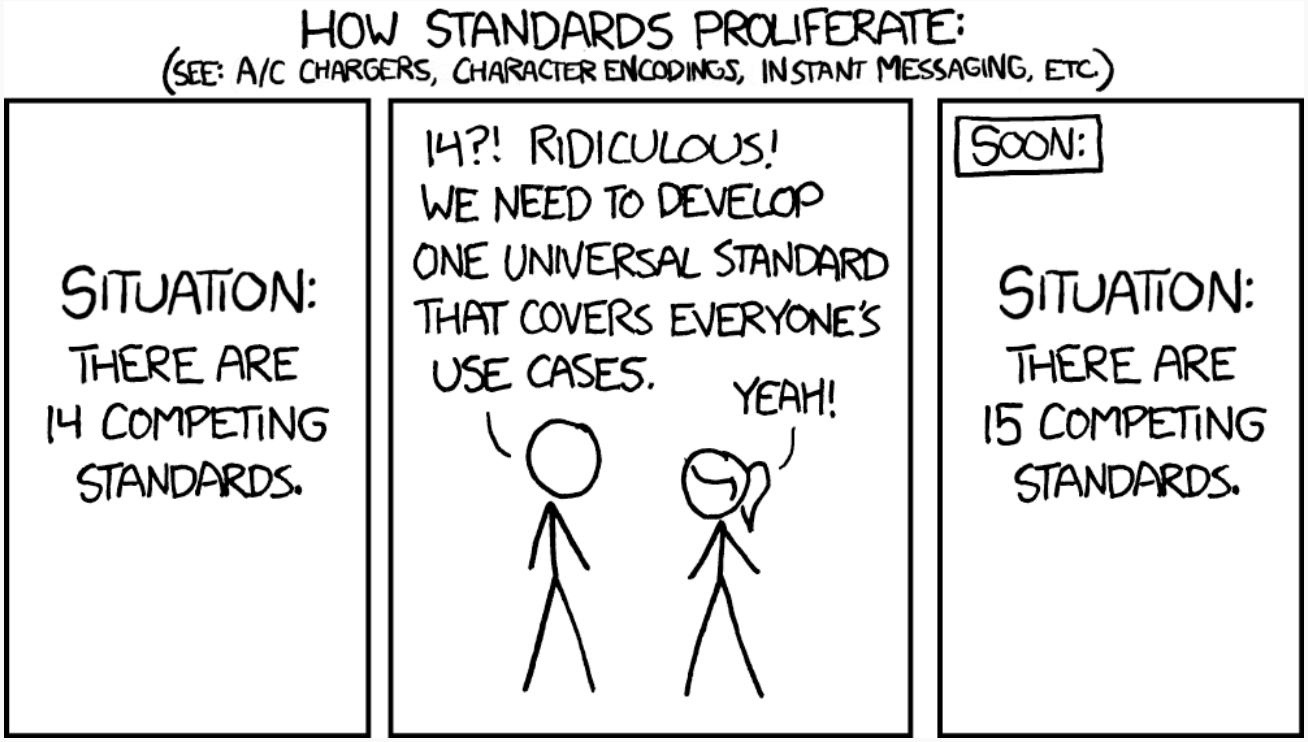
A Flask token is made up of three Base64 sections separated by dots.

`base64('{"username": "admin"}').<signature-stuff>.<signature-stuff>`

No, I don’t know what the signature stuff is :)

JSON Web Tokens

Another form of signed token, but this one has a standard!



JSON Web Tokens

A JWT is made up of three Base64 sections separated by dots.

JSON Web Tokens

A JWT is made up of three Base64 sections separated by dots.

Sound familiar?

JSON Web Tokens

A JWT is made up of three Base64 sections separated by dots.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTc0MDQ2Nzc0Mn0.Cumjo-
Y791DtTd29TWCaQruQBINoO-NSJkCthBDcXqk
```

JSON Web Tokens

A JWT is made up of three Base64 sections separated by dots.

```
base64('{"alg": "HS256", "typ": "JWT"}').base64('{"sub": "admin",
"exp": 1740467742}').<signature>
```

Signed Token Drawbacks

An important note about these signed tokens is that the data is just base64 encoded: **this can be decoded by the client!**

You can opt to encrypt the token's data rather than just sign it. The JavaScript Object Signing and Encryption framework (which standardises JWTs) also includes a spec for JWEs (plus JWKs and JWSs).

The JOSE framework also allows you to sign tokens in a way that can be verified by other servers.

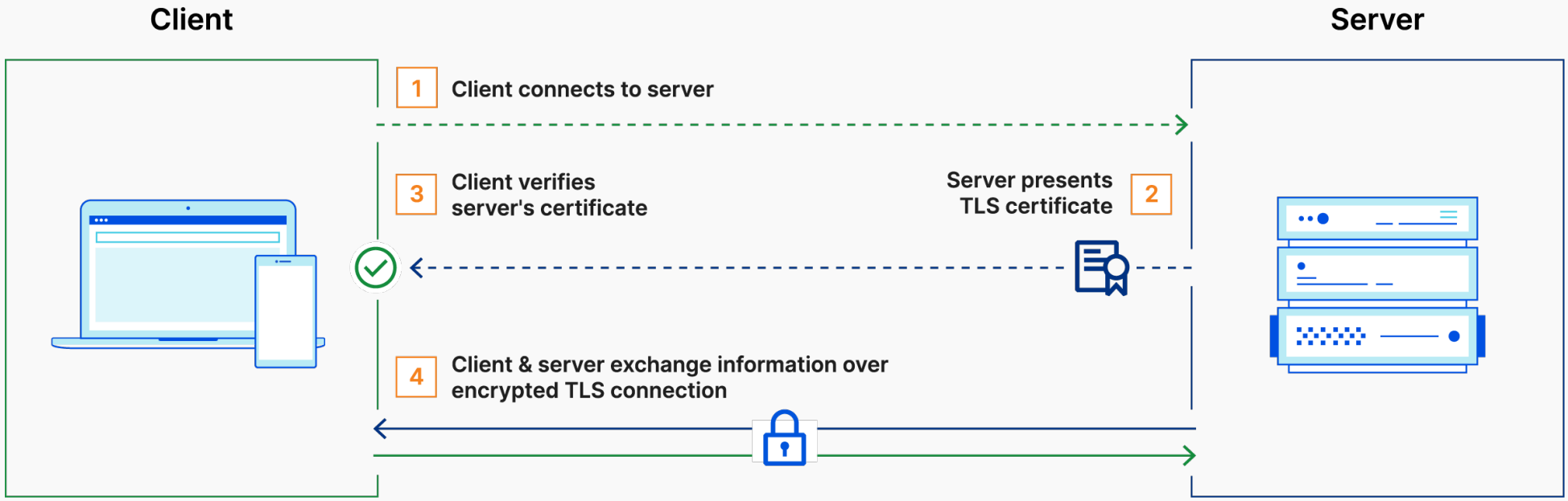
However, the thing you can't do (without a database query), is revoke them.

Opaque vs Signed

	Opaque	Signed
Best Performance	✗	✓
Expirable	✓	✓
Revokable	✓	✗
Store Sensitive Information	✓	~
Verifiable by Others	✗	~

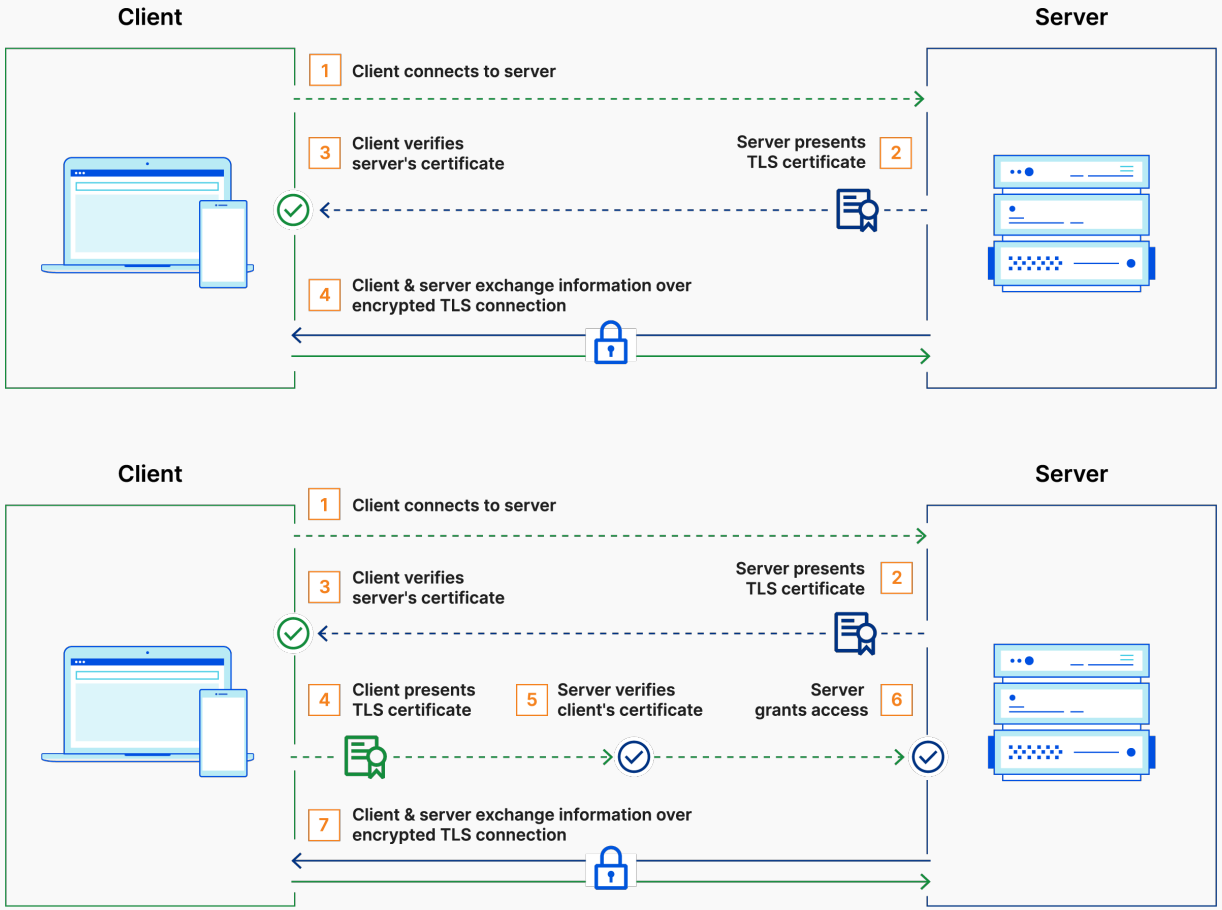
We **can** bypass these limitations by combining these concepts: some websites will use both a short-lived access token and a longer-lived refresh token.

TLS, mTLS and Certificates



From https://www.cloudflare.com/en-gb/learning/access-management/what-is-mutual-tls/

TLS, mTLS and Certificates



From https://www.cloudflare.com/en-gb/learning/access-management/what-is-mutual-tls/

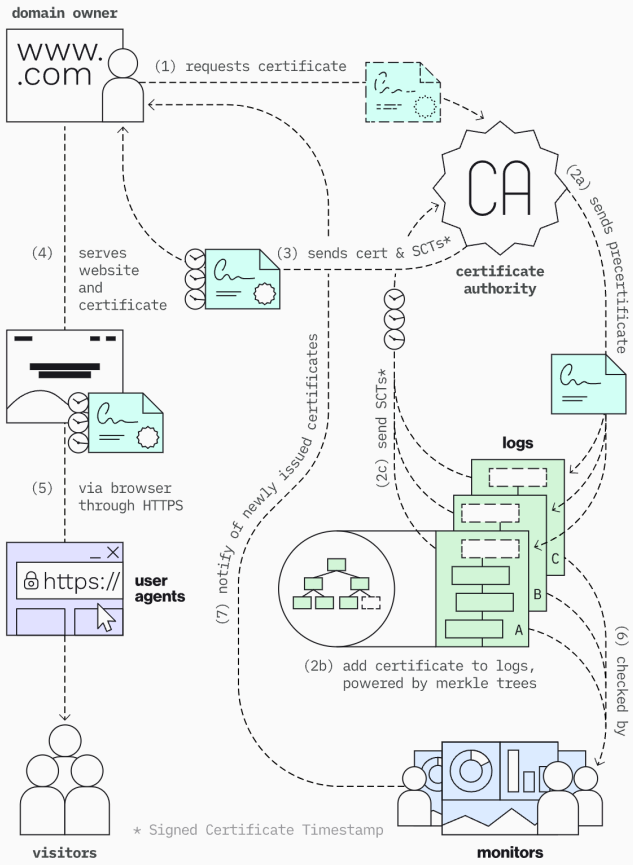
HSTS and HSTS Preloading

HTTP Strict Transport Security (HSTS) tells browsers to **only** load a website over HTTPS.

```
Strict-Transport-Security: max-age=<expire-time>
```

This can also be enforced via ‘preloading’ where this information is baked into users’ browsers to avoid even the first insecure connection: https://hstspreload.org

Certificate Transparency



From https://certificate.transparency.dev/howctworks/