Authentication & Authorization

Who are you and what can you do

Authentication & Authorization

Authentication

- Authentication refers to establishing an actor's identity sufficiently
  - Driver's license
  - University CatCard
  - Username and Password
- Only establishes Identity
- Can be satisfied within the service, or through an external Identity Provider

Authentication & Authorization

Authorization

- Authorization refers to establishing what actions a verified actor can perform
  - Depends on Authentication
- Many strategies
  - Groups
  - Roles
  - Usually dependent on the service / application to determine

Authentication

Methods and Use Cases

- There are usually different strategies for Authentication depending on if you are Authenticating a person, or some sort of other actor, like application code.
  - When you log in to D2L we use a different strategy (NetID+Password+DUO) than if you were authenticating to make certain API calls (Access Keys or Certificates

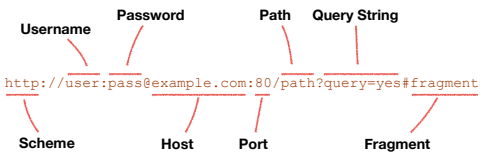
Authentication

Person Authentication

	Memorized Passwords	Password Manager	Password + MFA	Security Keys	Passkeys
Easy to use	✓	✓	✓	!	✓
Always with you	✓	✓	!	!	✓
Widely used	✓	✓	✓	!	!
Security Level	✗	!	!	✓	✓
Recoverable	✗	!	!	✗	!
Phishing resistant	✗	✗	!	✓	✓
Doesn't require shared secrets	✗	✗	✗	✓	✓

Authentication

HTTP Requests



Authentication
HTTP Basic Authentication

- The username:password portion of a URL is translated into Basic Authentication by user agents (browsers, curl, etc)

```
http://user:pass@example.com/index.html
```

```
GET /index.html HTTP/1.1
Host: example.com
Authentication: Basic dXNlcjpwYXNz
```


Authentication
HTTP Basic Authentication

- Basic Auth must only ever be used with TLS encrypted connections: HTTPS


```
https://user:pass@example.com/index.html
```


Authentication
HTTP Basic Authentication

- Not encrypted, just base64 encoded

```python
import base64

username = "mark"
password = "aReallyGr8PasswordNoOneWillGuess"

authString = f"{username}:{password}"
binaryAuthString = authString.encode("UTF-8")
b64AuthString = base64.b64encode(binaryAuthString).decode("UTF-8")
authHeader = f"Authentication: Basic {b64AuthString}"

print(authHeader)

# Prints the following
# Authentication: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
```

Authentication

HTTP Basic Authentication

- Libraries and tools make this really easy

```
import requests

url = "https://example.com/index.html"
username = "mark"
password = "aReallyGr8PasswordNoOneWillGuess"

response = requests.get(url, auth=(username, password))
headers = "\r\n".join(f"{k}: {v}" for k, v in response.request.headers.items())
print(headers)

# User-Agent: python-requests/2.28.1
# Accept-Encoding: gzip, deflate
# Accept: */*
# Connection: keep-alive
# Authorization: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
```

10

Authentication

HTTP Basic Authentication

- Libraries and tools make this really easy

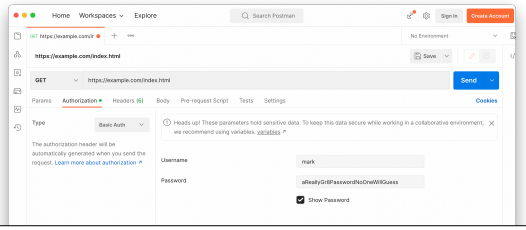
```
~ $ curl -v --user "mark:aReallyGr8PasswordNoOneWillGuess" https://example.com/index.html
*   Trying 93.184.216.34:443...
* Connected to example.com (93.184.216.34) port 443 (#0)
* Server auth using Basic with user 'mark'
> GET /index.html HTTP/2
> Host: example.com
> authorization: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
> user-agent: curl/7.79.1
> accept: */*
>
```

11

Authentication

HTTP Basic Authentication

- Libraries and tools make this really easy



12

Authentication

Storing Usernames and Passwords

- How do you securely store passwords?
- Naive way is to just store the plaintext username and password in a data store. When someone logs in, you compare the password they entered with the one you stored.
- Advantages:
  - You can see their passwords if they need to recover them
- Disadvantages:
  - If you can see their passwords, so can the baddies (there are so many baddies)

Authentication

Storing Usernames and Passwords

- Better way is to use a strong hash algorithm with a salt
- Hashes are one-way transformation. Easy to transform an input into an output, but very very difficult to go the other way around.
- Store the hashed value in your data store
- Re-hash each password attempt, and compare the hashes
- If a baddie steals your data store hashes, your passwords are still relatively protected
- A salt value helps protect against pre-computed hash tables

Authentication

Storing Usernames and Passwords

```
import hashlib

username = b"mark"
password = b"aReallyGr8PasswordNoOneWillGuess"

hashedPass = hashlib.sha3_512(password)
print(hashedPass.hexdigest())

# 07ef323985718aade0fa0e40e86d6f0cf429f6c8ce55dd4e7ec5f9ee0e3fcf533db...

hashedPass = hashlib.blake2b(password, salt=username)
print(hashedPass.hexdigest())

# 4fe792736fbc3d1366b3e63f1223e39abacd208de0378db03c1d27c4b3663b74b11c...
```

Authentication
Identity Providers

- Even better is to not get into the authentication business in the first place
- Use someone else's set of identities
  - Social IdPs: Google, Facebook, etc
  - Enterprise specific IdPs: University NetID
- Gets you off the hook for having to securely store authentication credentials

Authentication
Identity Providers

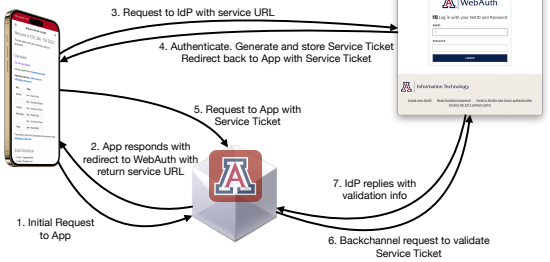
- Authentication Protocols
  - OAuth2
  - OpenID Connect (OIDC)
  - Security Assertion Markup Language (SAML)
  - Central Authentication Service (CAS)

Authentication
Central Authentication Service (CAS)

- CAS is pretty easy to implement ourselves
- Supported by the University's Shibboleth Identity Provider

Authentication

Central Authentication Service (CAS)



Demo