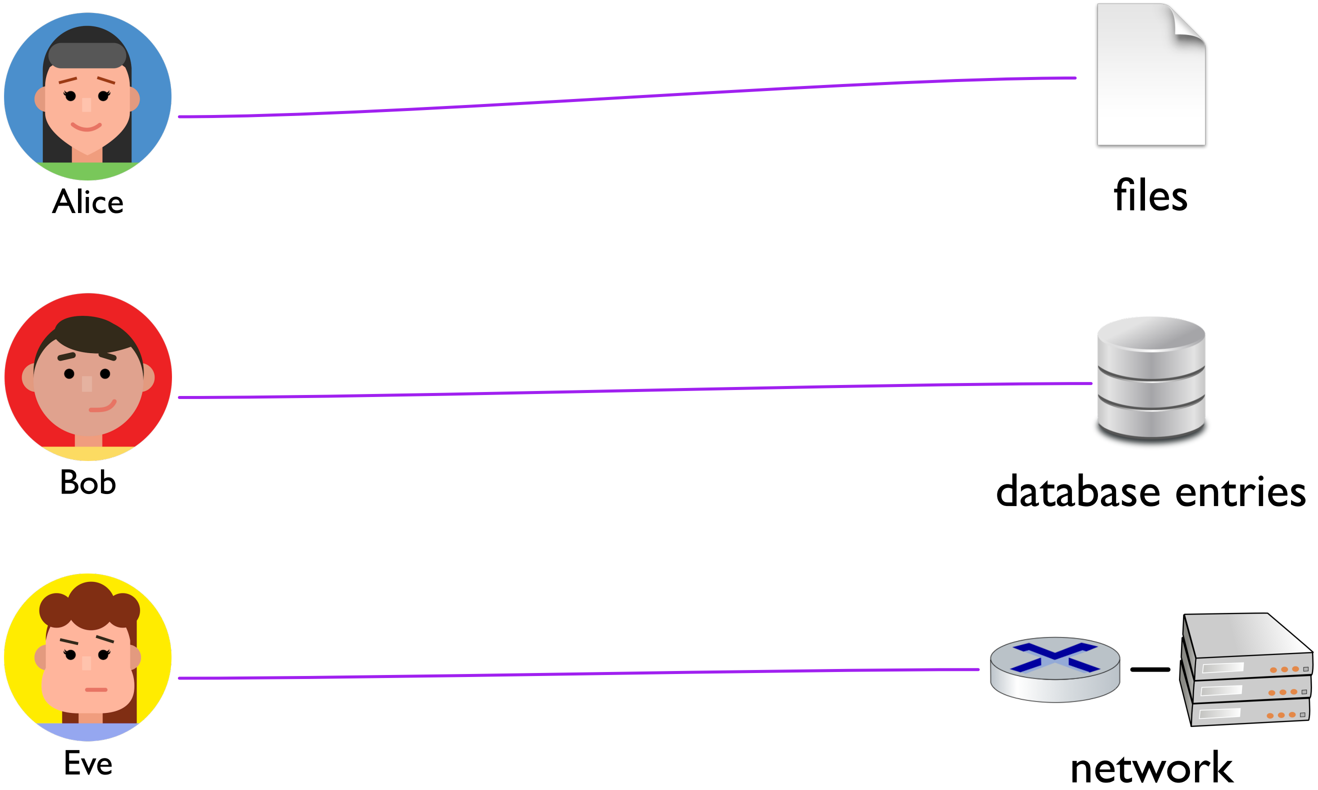
Users and Resources



Alice

Bob

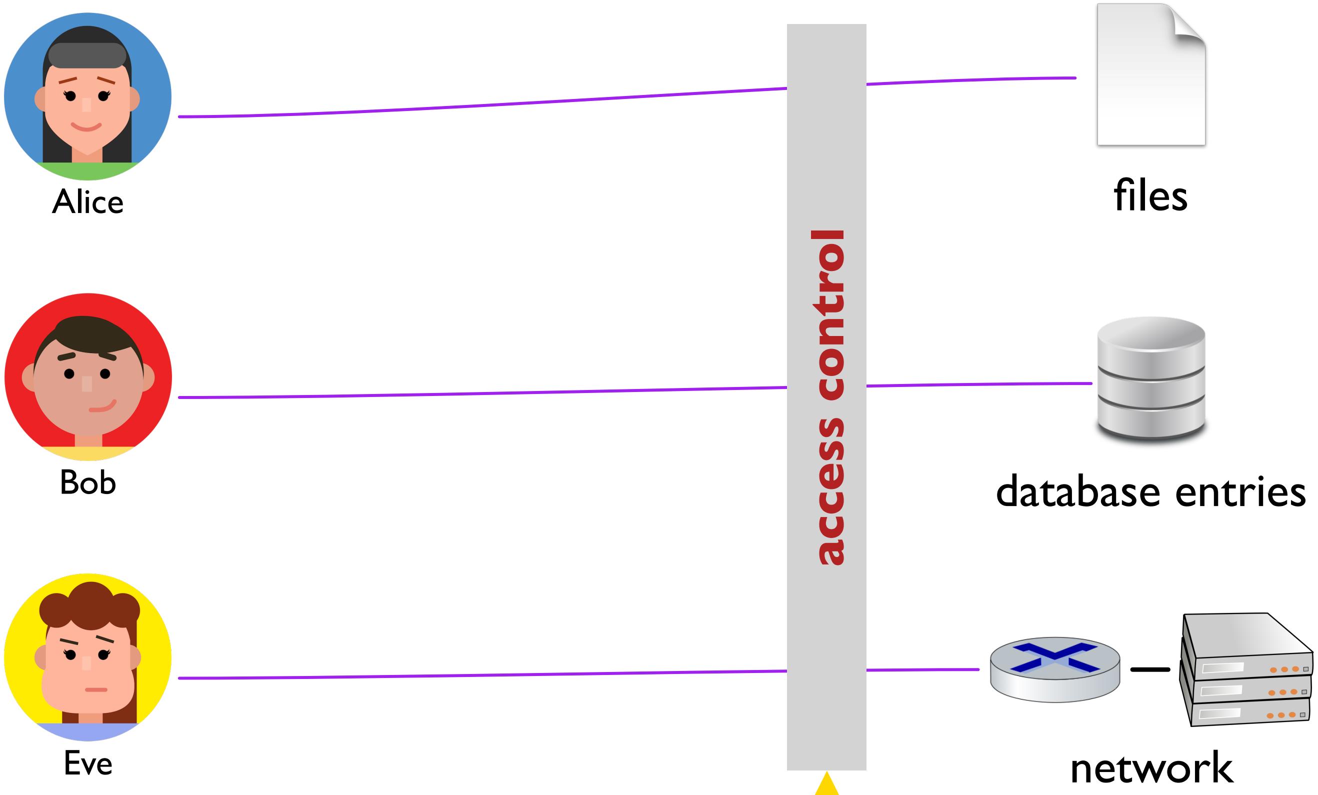
Eve

files

database entries

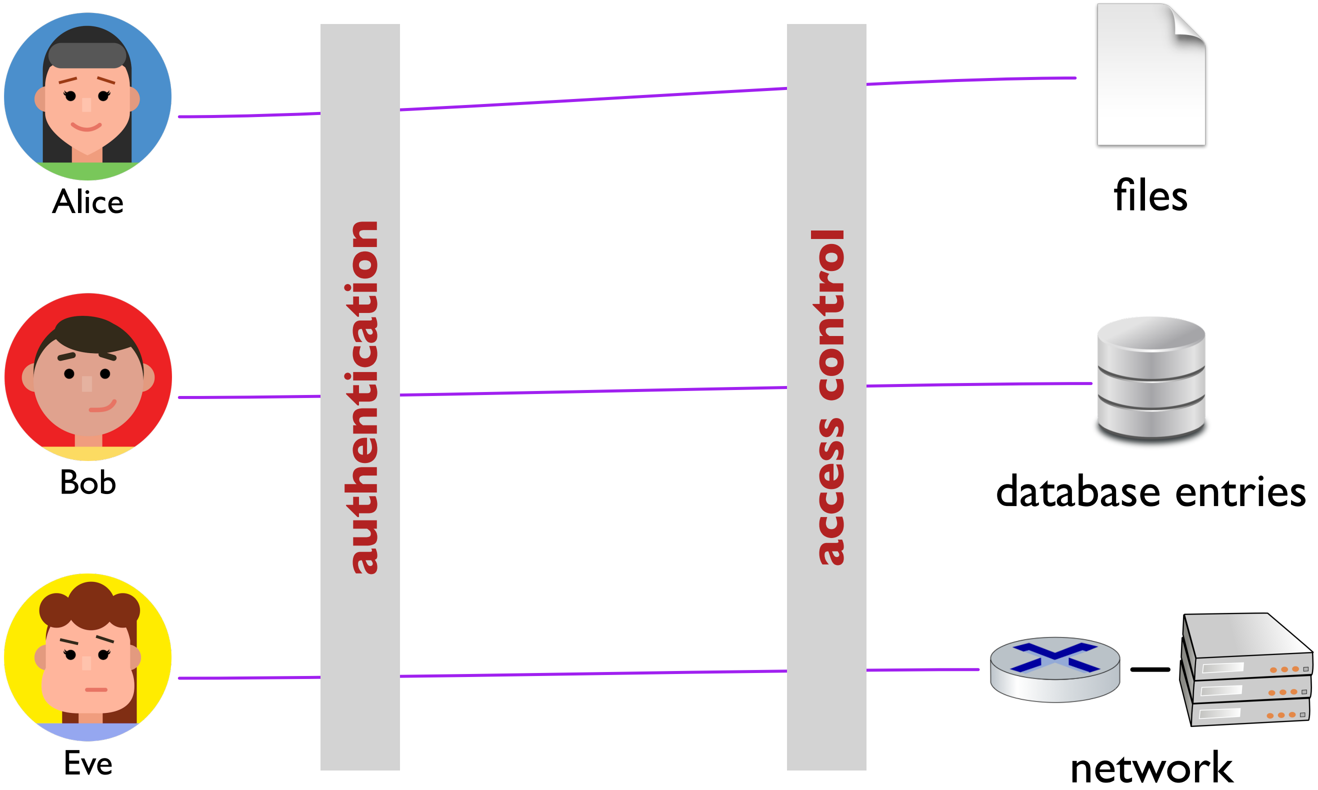
network

Users and Resources

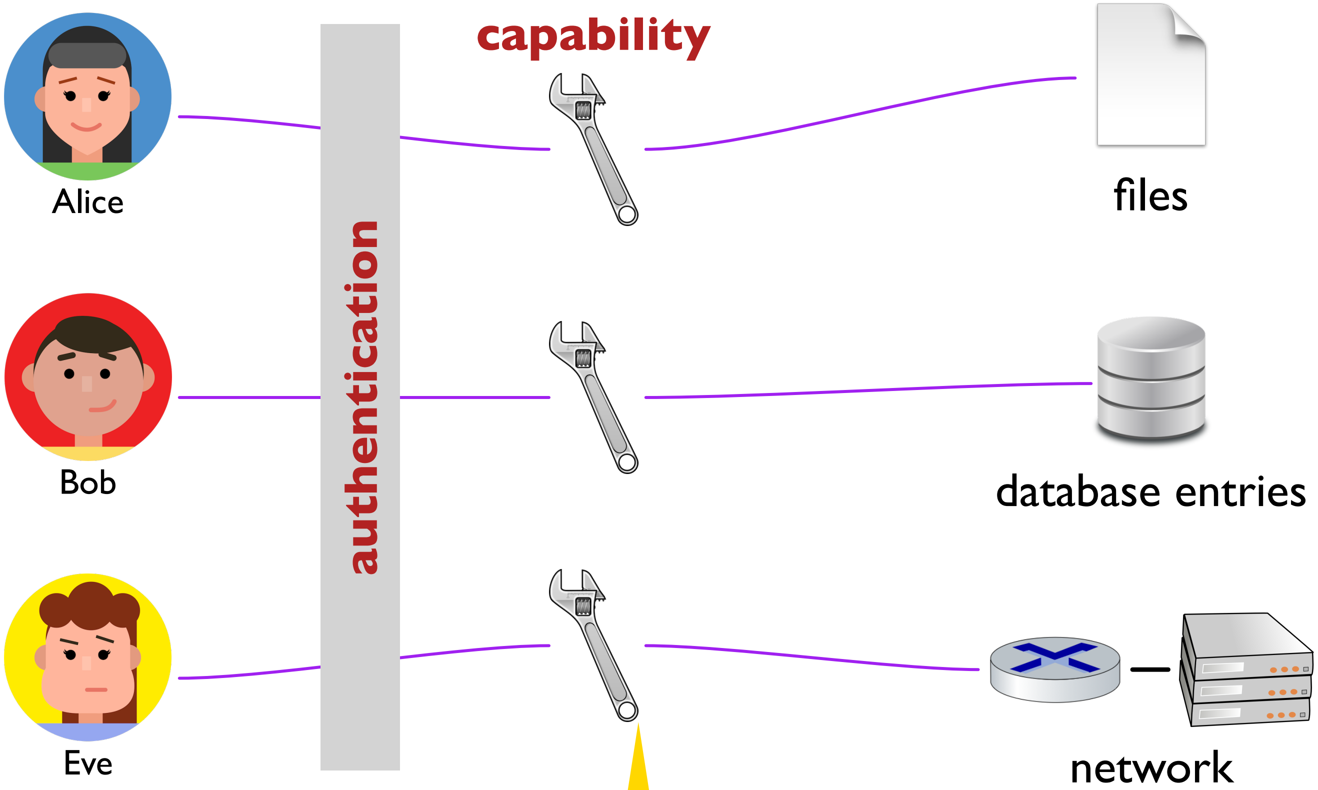


Access control determines who/what can use a given resource

Users and Resources

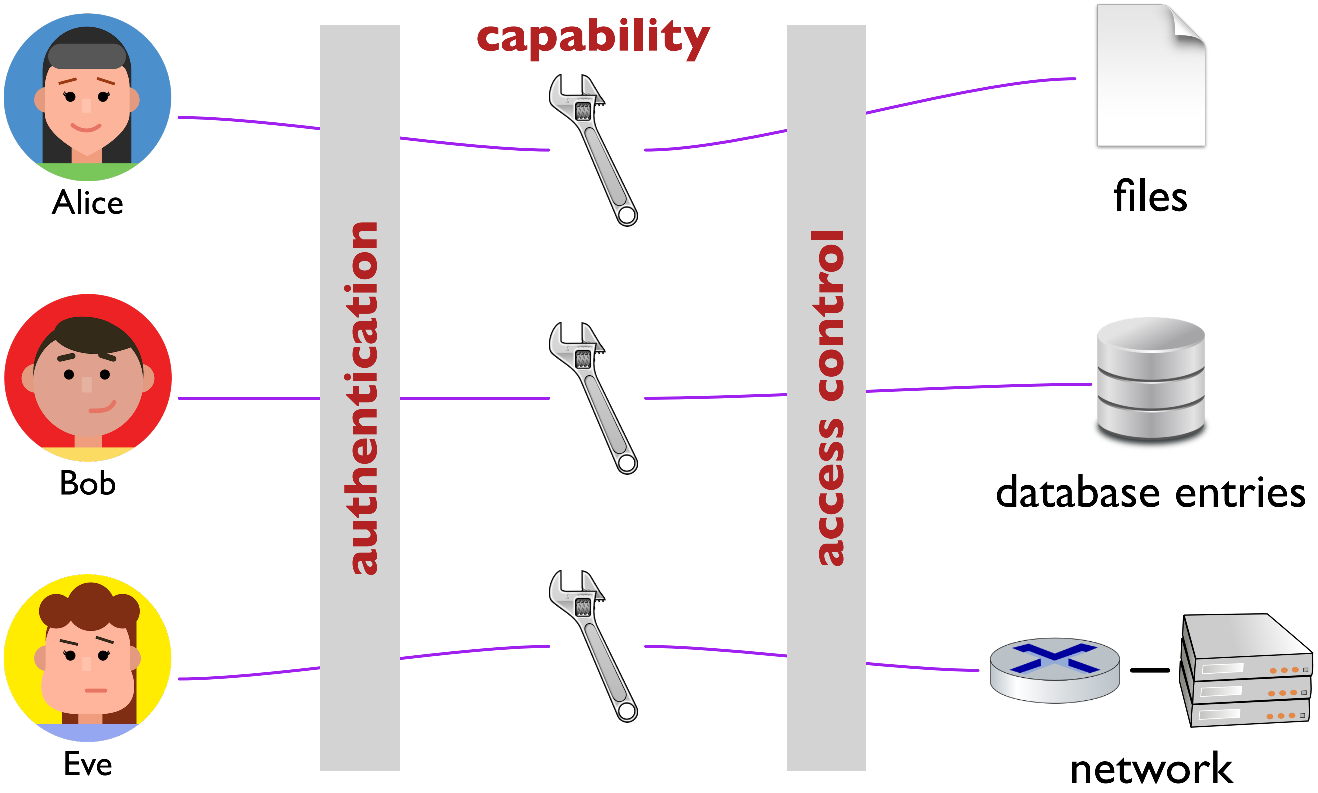


Users and Resources



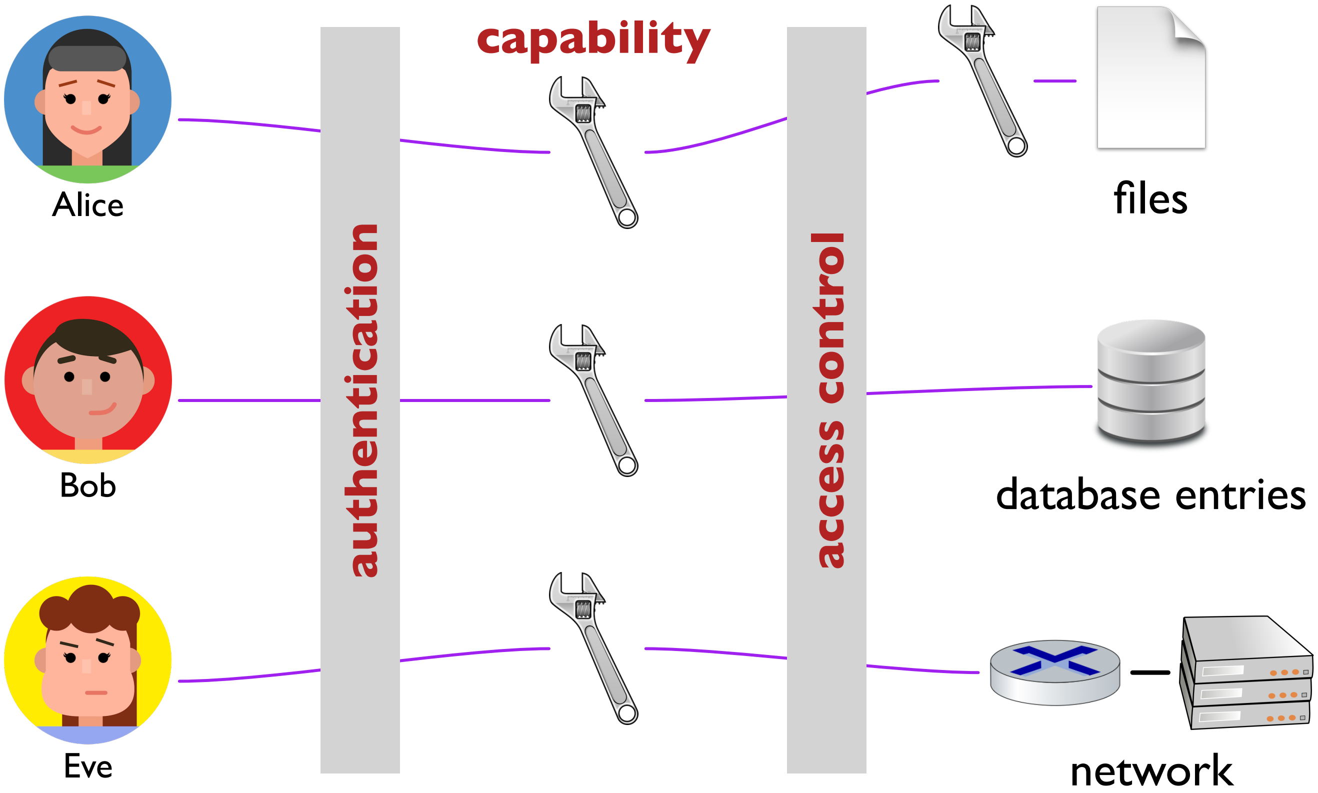
A **capability** says what resources can be used

Users and Resources



A typical system uses a mixture of capabilities and access control

Users and Resources



A typical system uses a mixture of capabilities and access control

Aside: Objects versus Abstract Datatypes

object-oriented

```
interface Shape {
    int area();
    int perimeter();
}

class Circle implements Shape {
    int area() { .... }
    int perimeter() { .... }
}

class Square implements Shape {
    int area() { .... }
    int perimeter() { .... }
}
```

New `Shape` operation needs change

Add new `Shape` variant without changing old code

datatype-oriented

```
type Shape
| Circle
| Square

fun area(s :: Shape):
  match s
  | Circle: ....
  | Square: ....

fun perimeter(s :: Shape):
  match s
  | Circle: ....
  | Square: ....
```

New `Shape` variant needs change

Add new `Shape` operation without changing old code

Aside: Objects versus Abstract Datatypes

object-oriented

```
interface Shape {
   int area();
   int perimeter();
}

class Circle im
   int area() {
   int perimete
}

class Square im
   int area() {
   int perimeter() { .... }
}
```

New Shape operation needs change

Add new Shape variant without changing old code

datatype-oriented

```
type Shape
| Circle
| Square

                 :: Shape):

                    ....
                    ....

                er(s :: Shape):

                    ....
   | Square:  ....
```

New Shape variant needs change

Add new Shape operation without changing old code

Each style makes some things easier, but both can get to the same place

Capability versus **access control** is a similar trade-off

Authentication Approaches

Authentication relies on something the user...

knows

password

`"superSecret"`

personal info

mother's maiden name is Smith

has

private key

phone app

RFID

email

SMS

is

face

fingerprint

retinal scan

signature

Santa Claus

voice

Authentication Approaches

Authentication relies on something the user...

knows

+ Easy input

– Hard to remember

password

"superSecret"

personal info

mother's maiden name is Smith

has

+ Likely on hand

– Can be stolen

private key

phone app

RFID

email

SMS

Two-factor authentication (2FA)

uses two of these

is

+ Always on hand

– Immutable and imitable

face

fingerprint

retinal scan

signature

Santa Claus

voice

Authentication Result

The result of authentication is a capability 🔧

Authentication Result

system	*capability* 🔧
operating system login	each started process has user ID supplied by parent process

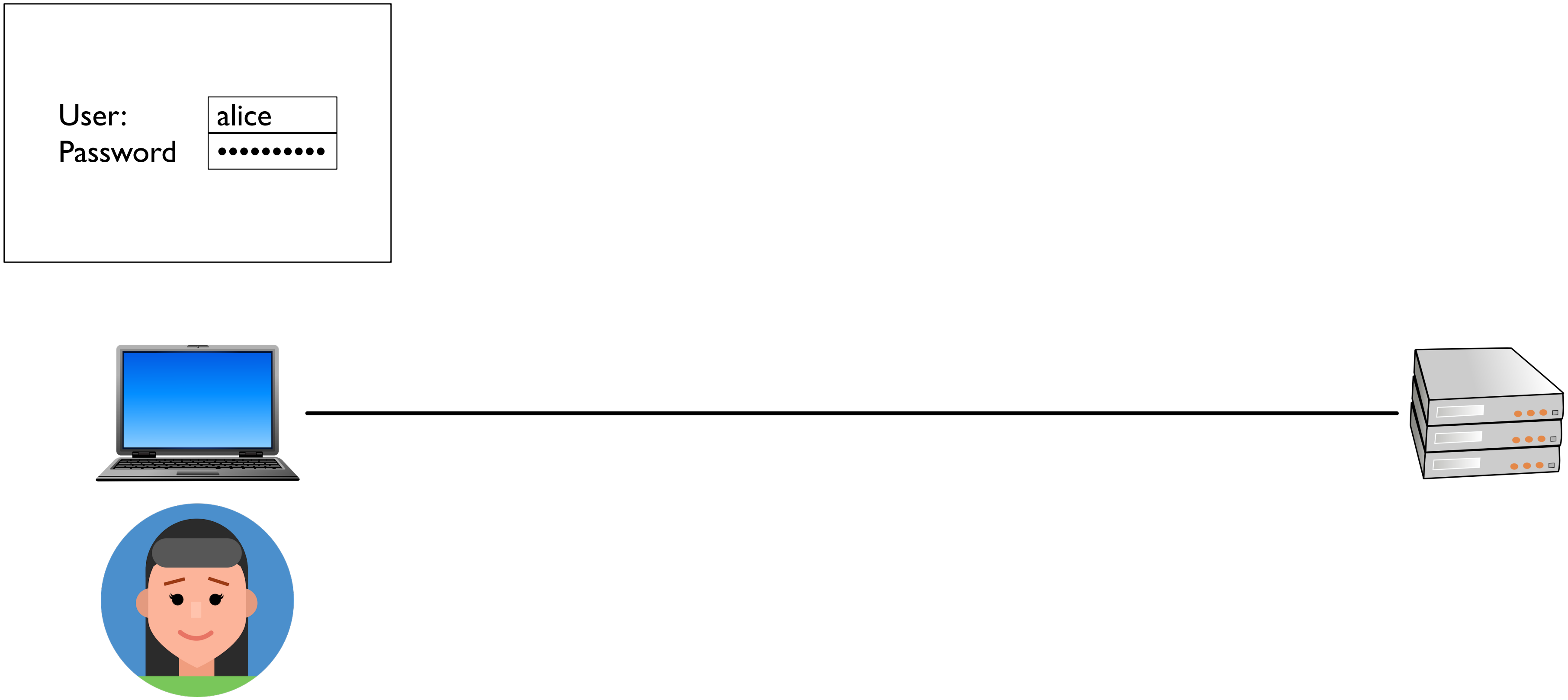
Authentication Result

system	*capability*
operating system login	each started process has user ID supplied by parent process
simple network service	TCP connection implies user

Authentication Result

system	*capability*
operating system login	each started process has user ID supplied by parent process
simple network service	TCP connection implies user
web service	login supplies a **cookie**, which is sent back with each request

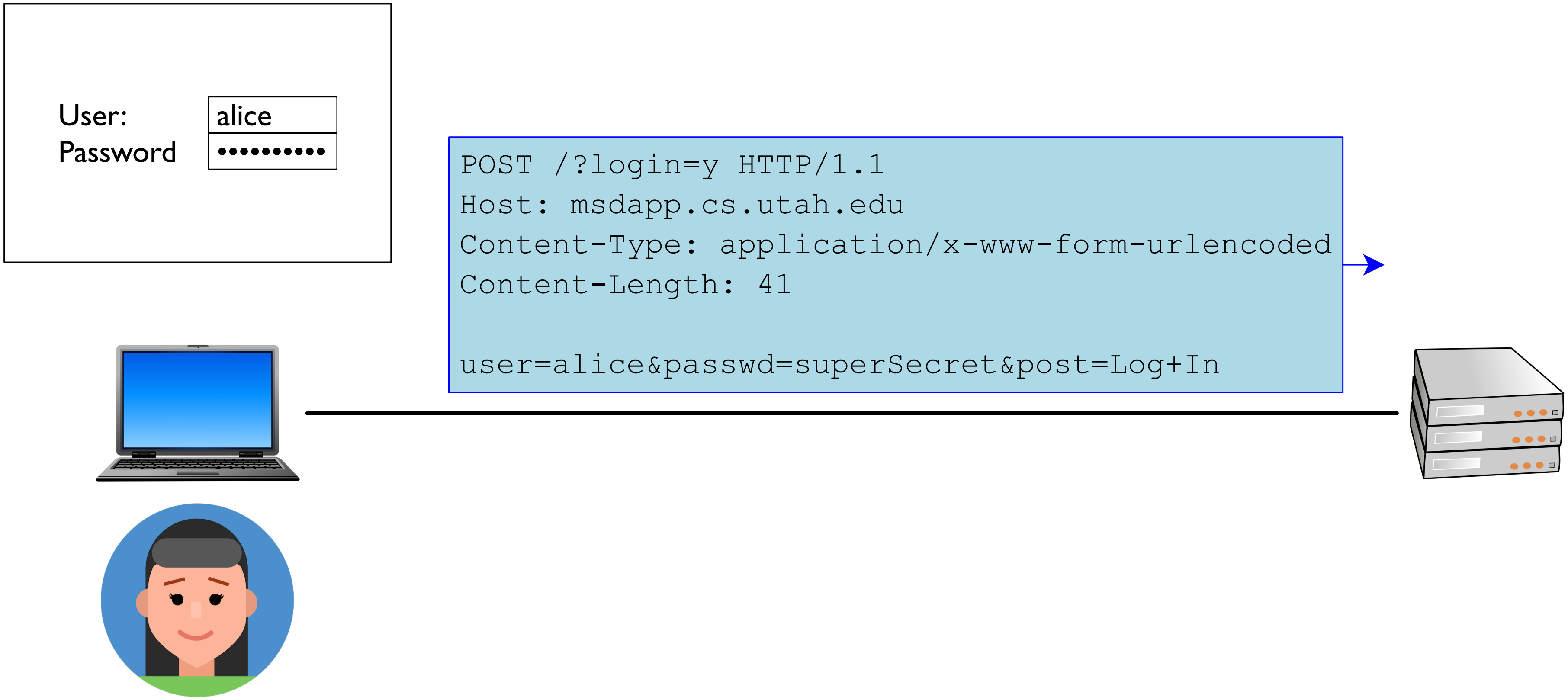
Authentication Tokens as Cookies



User: alice

Password ••••••••••

Authentication Tokens as Cookies

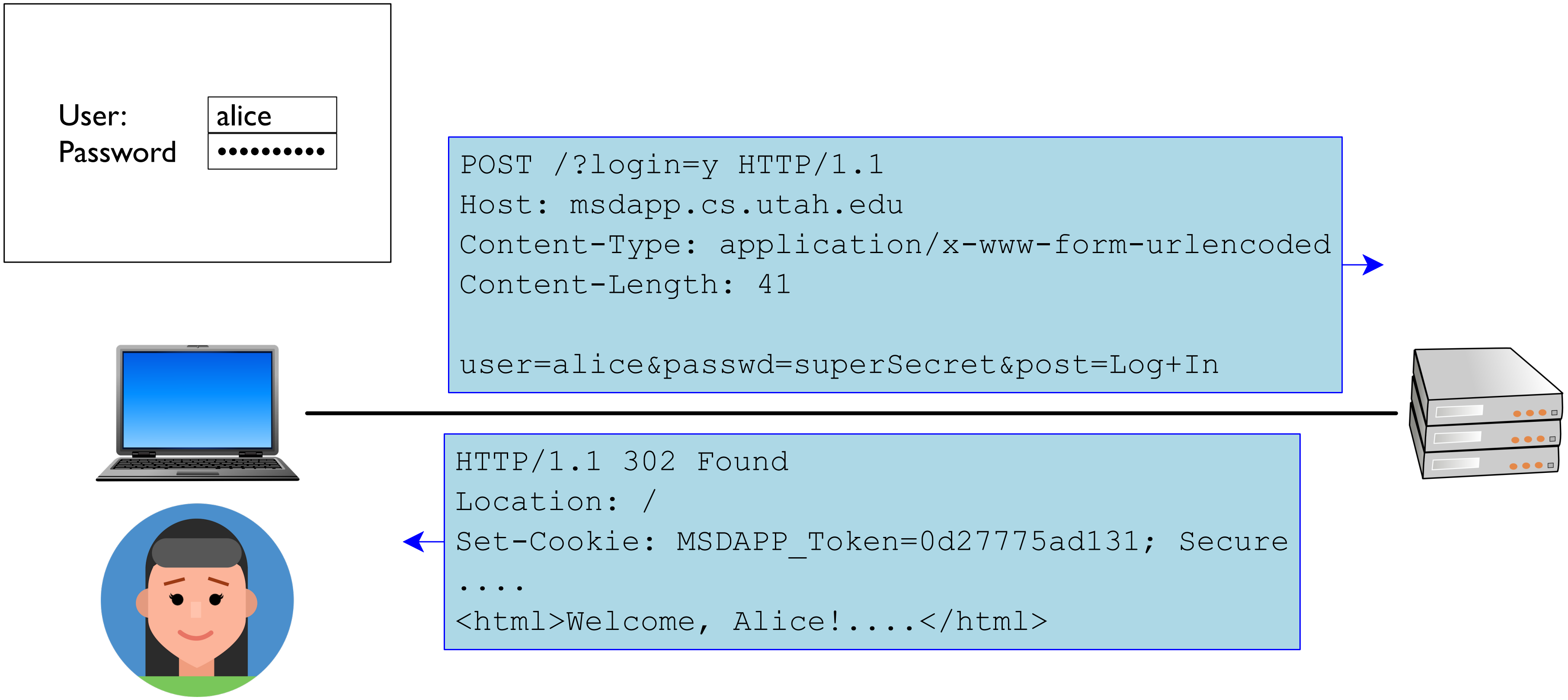


User: alice
Password ••••••••••

```
POST /?login=y HTTP/1.1
Host: msdapp.cs.utah.edu
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

user=alice&passwd=superSecret&post=Log+In
```

Authentication Tokens as Cookies



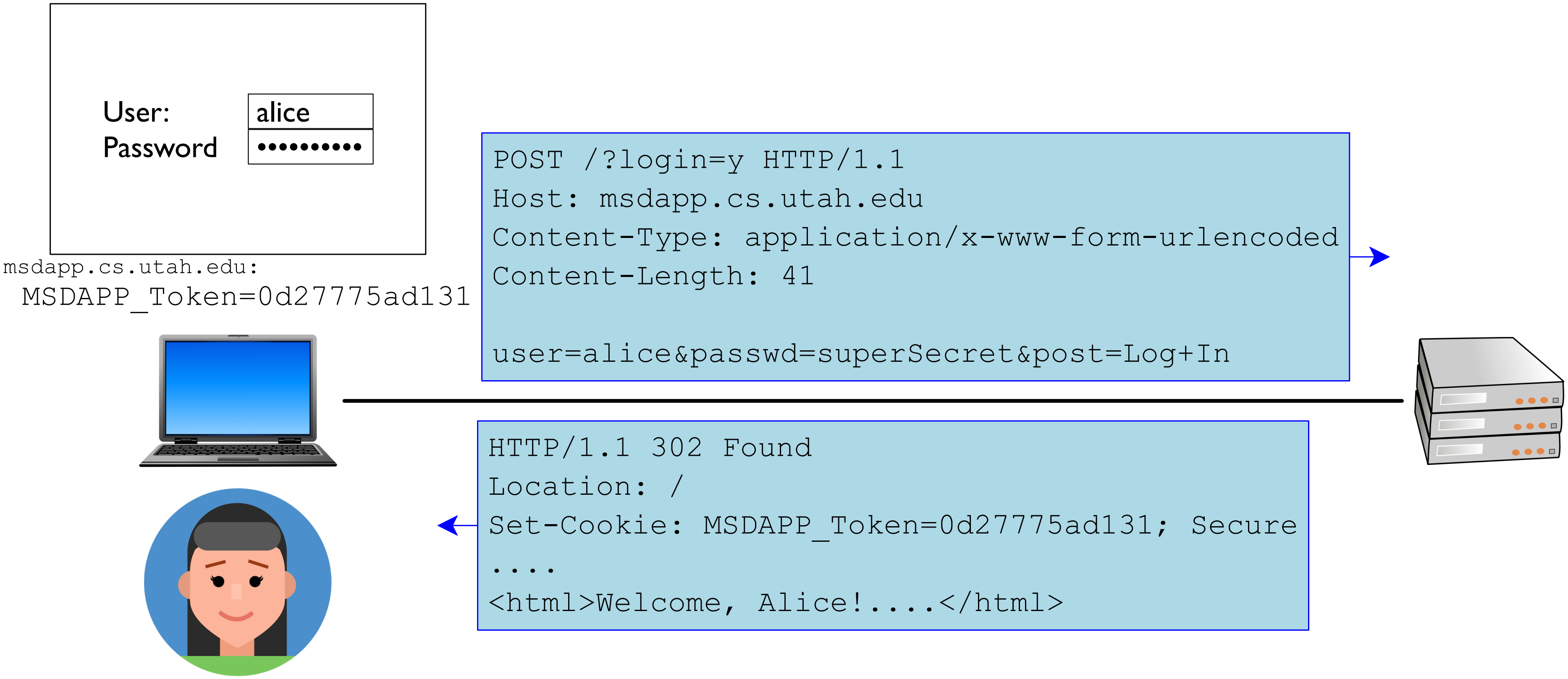
User: alice
Password ••••••••••

```
POST /?login=y HTTP/1.1
Host: msdapp.cs.utah.edu
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

user=alice&passwd=superSecret&post=Log+In
```

```
HTTP/1.1 302 Found
Location: /
Set-Cookie: MSDAPP_Token=0d27775ad131; Secure
....
<html>Welcome, Alice!....</html>
```

Authentication Tokens as Cookies



User: alice
Password ••••••••••

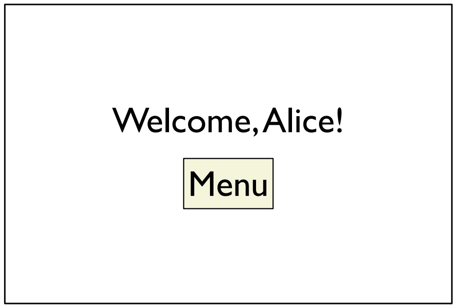
msdapp.cs.utah.edu:
MSDAPP_Token=0d27775ad131

```
POST /?login=y HTTP/1.1
Host: msdapp.cs.utah.edu
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

user=alice&passwd=superSecret&post=Log+In
```

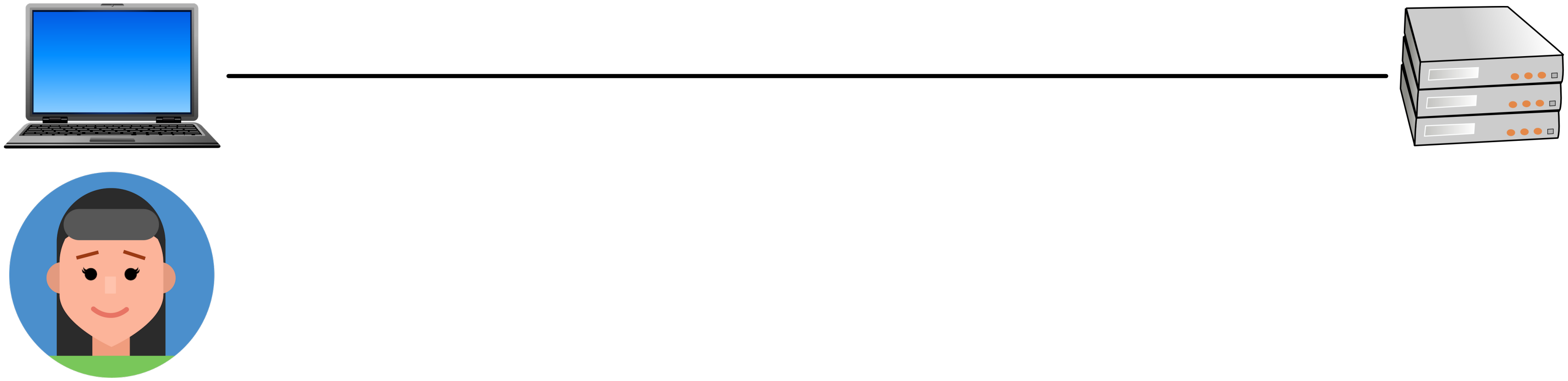
```
HTTP/1.1 302 Found
Location: /
Set-Cookie: MSDAPP_Token=0d27775ad131; Secure
....
<html>Welcome, Alice!....</html>
```

Authentication Tokens as Cookies

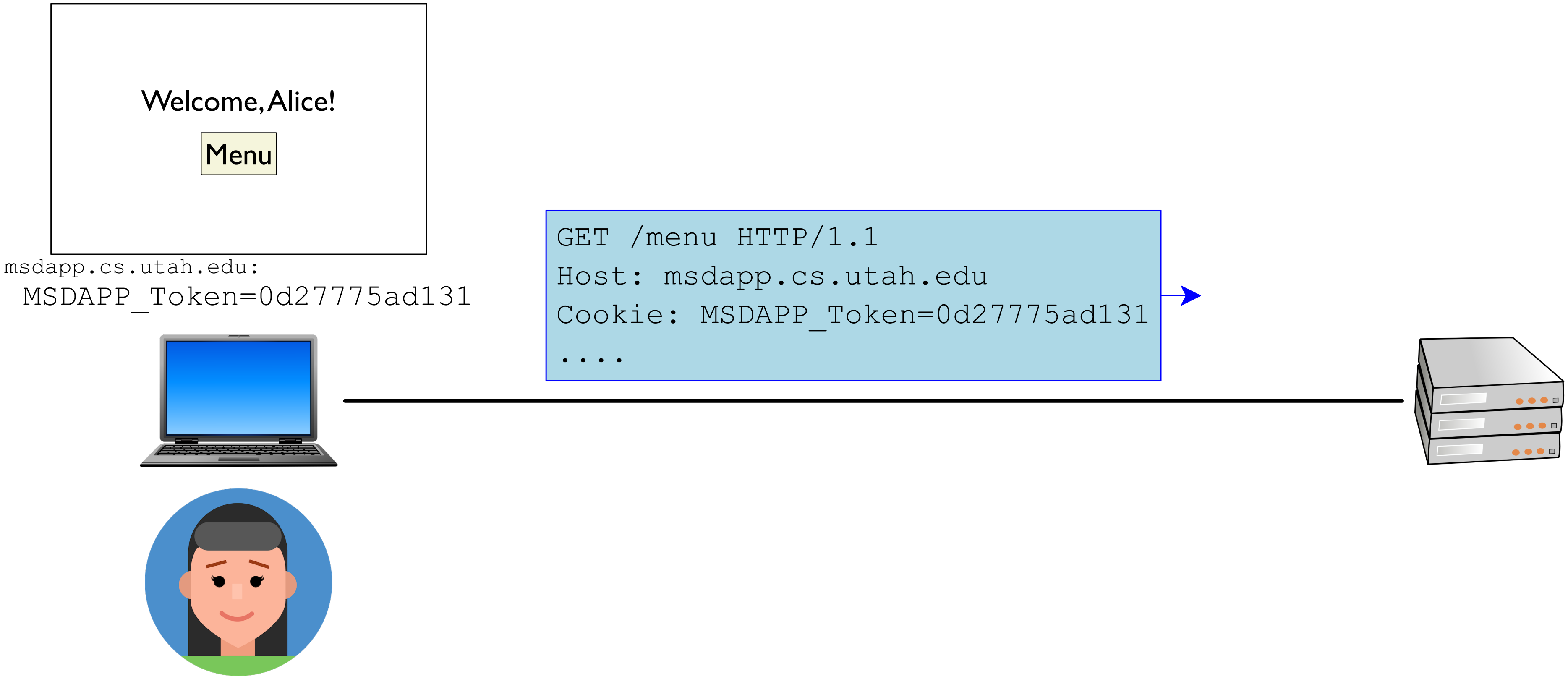


Welcome, Alice!

```
msdapp.cs.utah.edu:
 MSDAPP_Token=0d27775ad131
```



Authentication Tokens as Cookies



msdapp.cs.utah.edu:
MSDAPP_Token=0d27775ad131

```
GET /menu HTTP/1.1
Host: msdapp.cs.utah.edu
Cookie: MSDAPP_Token=0d27775ad131
....
```

Access Control

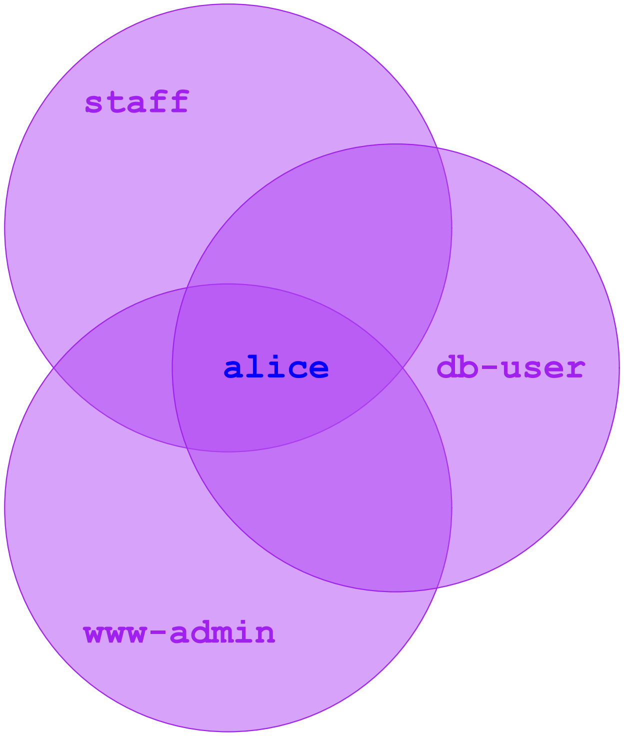
Given a current capabilty, such as the current user, **access control** determines whether to allow use of a specific resource

Simple access control: Unix file permissions

More flexible: **access-control list (ACL)**

Unix Users and File Permissions

Every *user* belongs to one or more *groups*



Unix Users and File Permissions

Every file has an owning *user* plus *group* and a table:

	read	write	execute
user	✓/✗	✓/✗	✓/✗
group	✓/✗	✓/✗	✓/✗
others	✓/✗	✓/✗	✓/✗

Unix Users and File Permissions

Every file has an owning *user* plus *group* and a table:

	`read`	`write`	`execute`
user	✓=1	✓=1	✓=1
group	✓=1	✗=0	✓=1
others	✗=0	✗=0	✗=0

`-rwxr-x---`

`111101000`

= `0750` octal

For a directory:

- `read` ⇒ `ls`
- `write` ⇒ create file or subdirectory
- `execute` ⇒ `cd`

Unix Users and File Permissions

Every file has an owning *user* plus *group* and a table:

	`read`	`write`	`execute`
user	✓=1	✓=1	✓=1
group	✓=1	✗=0	✓=1
others	✗=0	✗=0	✗=0

`-rwxr-x---`

`111101000`

= `0750` octal

Every process has a current *user* and *group*

- login or `su` changes current *user*
- login or `newgrp` changes current *group*

Unix Users and File Permissions

Every file has an owning *user* plus *group* and a table:

	`read`	`write`	`execute`
user	✓=`1`	✓=`1`	✓=`1`
group	✓=`1`	✗=`0`	✓=`1`
others	✗=`0`	✗=`0`	✗=`0`

`-rwxr-x---`

`111101000`

= `0750` octal

Every process has a current *user* and *group*

On file access, check permissions relative to *user* (and its *group*s)

On file creation, assign current *user* and *group* as owners

Unix Users and File Permissions

```
#include <unistd.h>
#include <fcntl.h>

int main() {
  close(open("the_new_file",
             O_RDWR | O_CREAT,
             0666));
  return 0;
}
```

Unix Users and File Permissions

```
#include <unistd.h>
#include <fcntl.h>

int main() {
  close(open("the_new_file",
             O_RDWR | O_CREAT,
             0666));
  return 0;
}
```

Specifies permissions for new file

Unix Users and File Permissions

```
#include <unistd.h>
#include <fcntl.h>

int main() {
  close(open("the_new_file",
             O_RDWR | O_CREAT,
             0666));
  return 0;
}
```

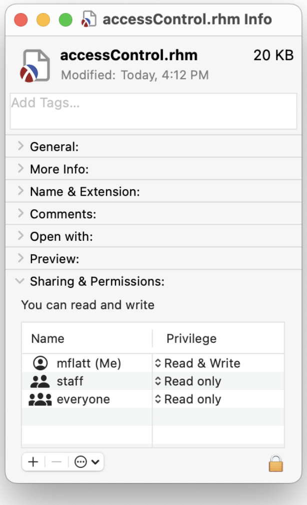
Specifies permissions for new file ... removing bits set in `umask`, which is also a process property

Access Control Lists

Unix traditional file permissions are specific to just one user and one group

A file can have a more general **access control list (ACL)** with specific permissions for multiple users and groups

Per-user permissions might be `R`/`W`/`X`, or permissions might be more general, depending on the OS and filesystem



accessControl.rhm Info

accessControl.rhm 20 KB

Modified: Today, 4:12 PM

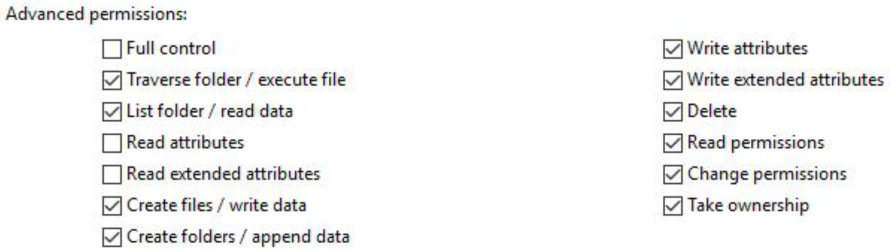
Add Tags...

- General:
- More Info:
- Name & Extension:
- Comments:
- Open with:
- Preview:
- Sharing & Permissions:

You can read and write

Name	Privilege
mflatt (Me)	Read & Write
staff	Read only
everyone	Read only

Windows



Advanced permissions:

☐ Full control
☑ Traverse folder / execute file
☑ List folder / read data
☐ Read attributes
☐ Read extended attributes
☑ Create files / write data
☑ Create folders / append data
☑ Write attributes
☑ Write extended attributes
☑ Delete
☑ Read permissions
☑ Change permissions
☑ Take ownership

Role-Based Access Control (RBAC)

In a setting with many kinds of actions (e.g., Amazon AWS), permissions can be grouped into **roles**

To allow a user/service to perform a set of actions, give them the relevant role

A role is a kind of capability 🔧

associated to a user, not a resource

Capabilities Instead of Access Control

In a pure capability-oriented view, all access control is through a capability

- There’s no way to even talk about an action without having that capability
- Capabilities include the possibility of generating and delegating capabilities

For example, a JavaScript program can manipulate a web page, but only through DOM methods, and there’s no way to perform an action that doesn’t have a method

Recovocation

Revocation of a capability removes its access

For example, a token/cookie for a network login is revoked when it expires

When an capability is represented by object, actions on the object may revoke its capabilities

For example, closing a file object revokes its ability to read a file

Support for revocation is a key issue in the design of a capability system

Summary

Authentication is only a first step:

- authenticated identity can be considered a **capability** that represents allowed actions
- this identity/capability might allow use of a resource pending **access control** checks

Capabilities and access control represent two sides of the same coin, but differ in whether they're associated with an actor or a resource