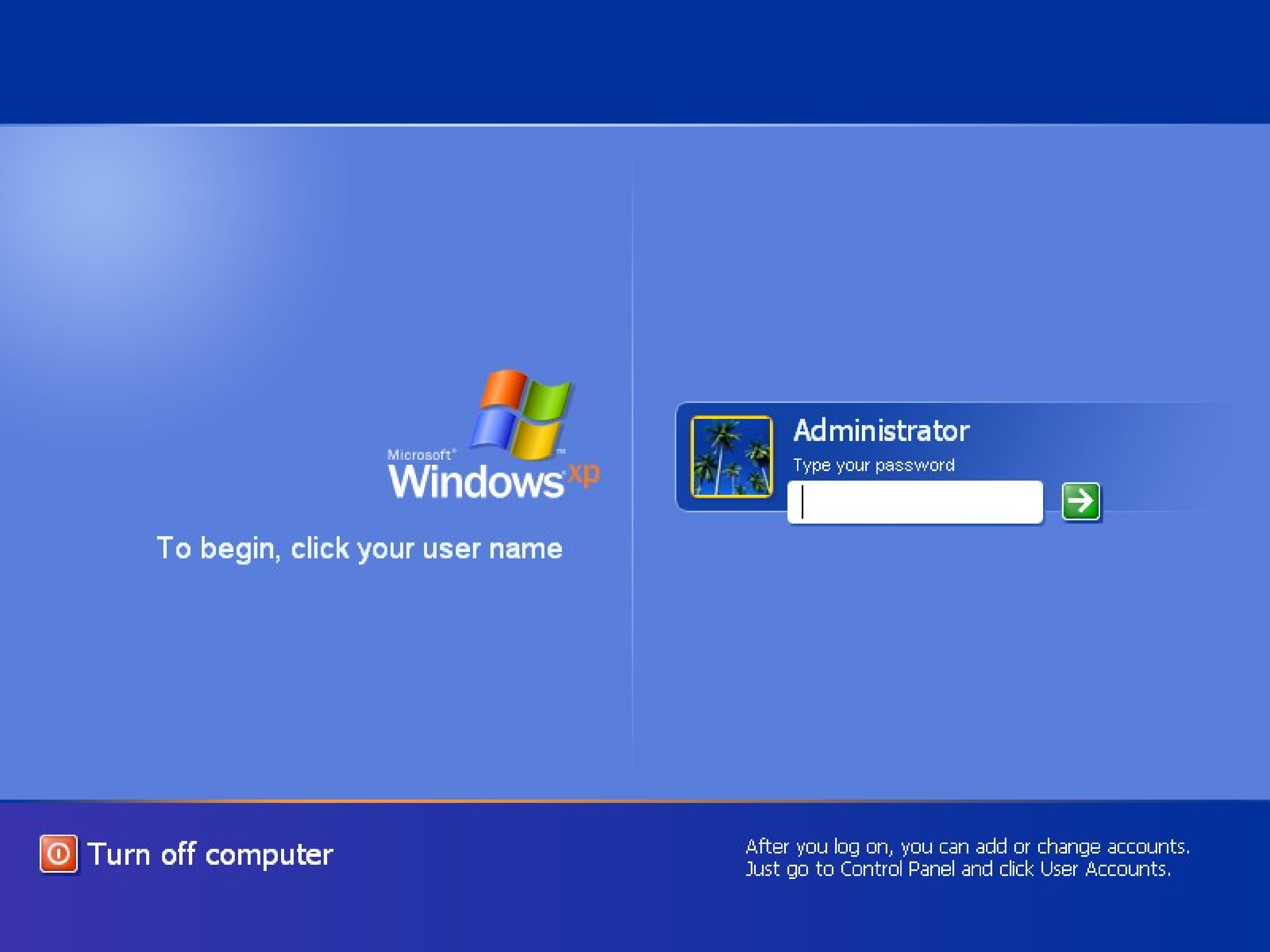
CS 5600
Computer Systems

Lecture 12: Authorization and Access Control

- Authentication
- Access Control
- Mandatory Access Control



To begin, click your user name

After you log on, you can add or change accounts.
Just go to Control Panel and click User Accounts.

Authentication

- Authentication is the process of verifying an actor's identity
- Critical for security of systems
  - Permissions, capabilities, and access control are all contingent upon knowing the identity of the actor
- Typically parameterized as a username and a secret
  - The secret attempts to limit unauthorized access

Types of Secrets

- Actors provide their secret to log-in to a system
- Three classes of secrets:
  1. Something you know
     - Example: a password
  2. Something you have
     - Examples: a smart card or smart phone
  3. Something you are
     - Examples: fingerprint, voice scan, iris scan

Checking Passwords

- The system must validate passwords provided by users
- Thus, passwords must be stored somewhere
- Basic storage: plain text

password.txt

cbw	p4ssw0rd
sandi	i heart doggies
amislove	93Gd9#jv*0x3N
bob	security

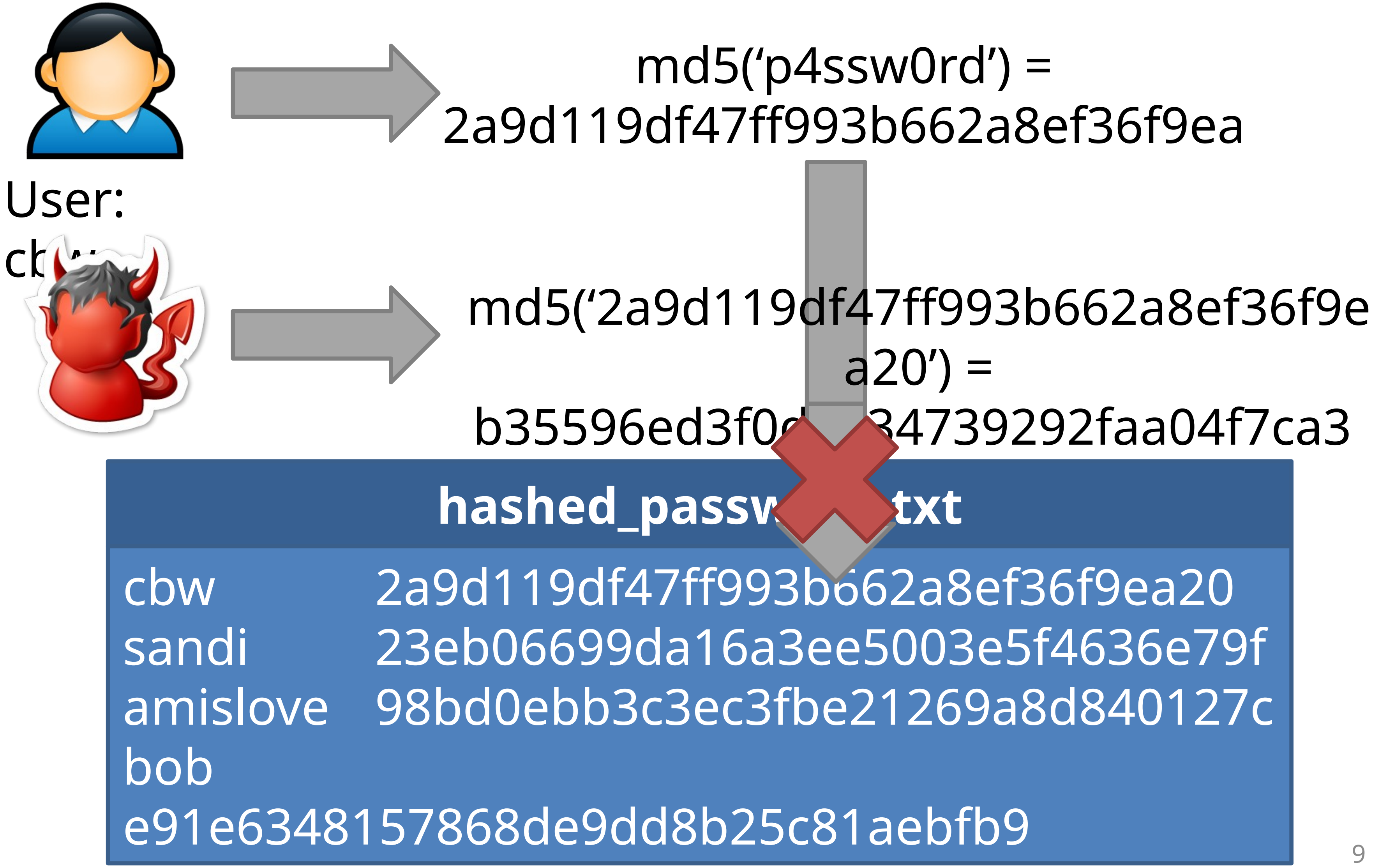
Problem: Password File Theft

- Attackers often compromise systems
- They may be able to steal the password file
  - Linux: /etc/shadow
  - Windows: c:\windows\system32\config\sam
- If the passwords are plain text, what happens?
  - The attacker can now log-in as any user, including root/administrator
- **Passwords should never be stored in plain text**

Hashed Passwords

- Key idea: store encrypted versions of passwords
  - Use one-way cryptographic hash functions
  - Examples: md5, sha1, sha256, sha512
- Cryptographic hash function transform input data into scrambled output data
  - Deterministic: hash(A) = hash(A)
  - High entropy:
    - md5(‘security’) = e91e6348157868de9dd8b25c81aebfb9
    - md5(‘security1’) = 8632c375e9eba096df51844a5a43ae93
    - md5(‘Security’) = 2fae32629d4ef4fc6341f1751b405e45
  - Collision resistant
    - Locating A’ such that hash(A) = hash(A’) takes a long time
    - Example: 2^{21} tries for md5

Hashed Password Example



md5('p4ssw0rd') =
2a9d119df47ff993b662a8ef36f9ea

User:
cb

md5('2a9d119df47ff993b662a8ef36f9e
a20') =
b35596ed3f0d 34739292faa04f7ca3

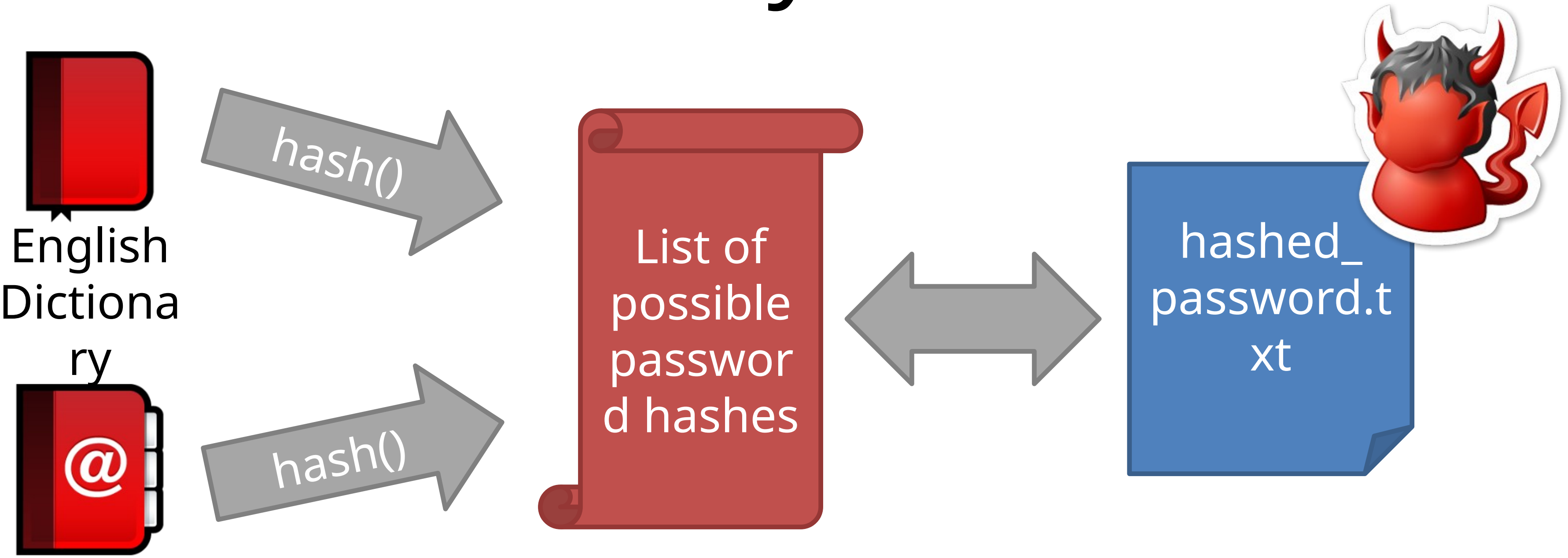
hashed_passw txt

cbw	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
amislove	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Attacking Password Hashes

- Recall: cryptographic hashes are collision resistant
  - Locating A’ such that hash(A) = hash(A’) takes a very long time
- Are hashed password secure from cracking?
  - **No!**
- Problem: users choose poor passwords
  - Most common passwords: 123456, password
  - Username: cbw, Password: cbw

Dictionary Attacks



English Dictionary

hash()

List of possible password hashes

hashed_password.txt

hash()

Common Passwords

- Common for 60-70% of hashed passwords to be cracked in <24 hours

Hardening Password Hashes

- Key problem: cryptographic hashes are deterministic
  - hash(‘p4ssw0rd’) = hash(‘p4ssw0rd’)
  - This enables attackers to build lists of hashes
- Solution: make each password hash unique
  - Add a salt to each password before hashing
  - hash(salt + password) = password hash
  - Each user has a unique, random salt
  - Salts can be stores in plain text

Example Salted Hashes

hashed_password.txt

cbw	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
amislove	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

hashed_and_salted_password.txt

		af19c842f0c781ad726de7aba439b033
sandi	0X	67710c2c2797441efb8501f063d42fb6
amislove	hz	9d03e1f28d39ab373c59c7bb338d0095
bob	K@	

Password Storage on Linux

/etc/passwd

username:x:UID:GID:full_name:home_directory:shell

cbw:x:1001:1000:Christo Wilson:/home/cbw/:/bin/bash
amislove:1002:2000:Alan Mislove:/home/amislove/:/bin/sh

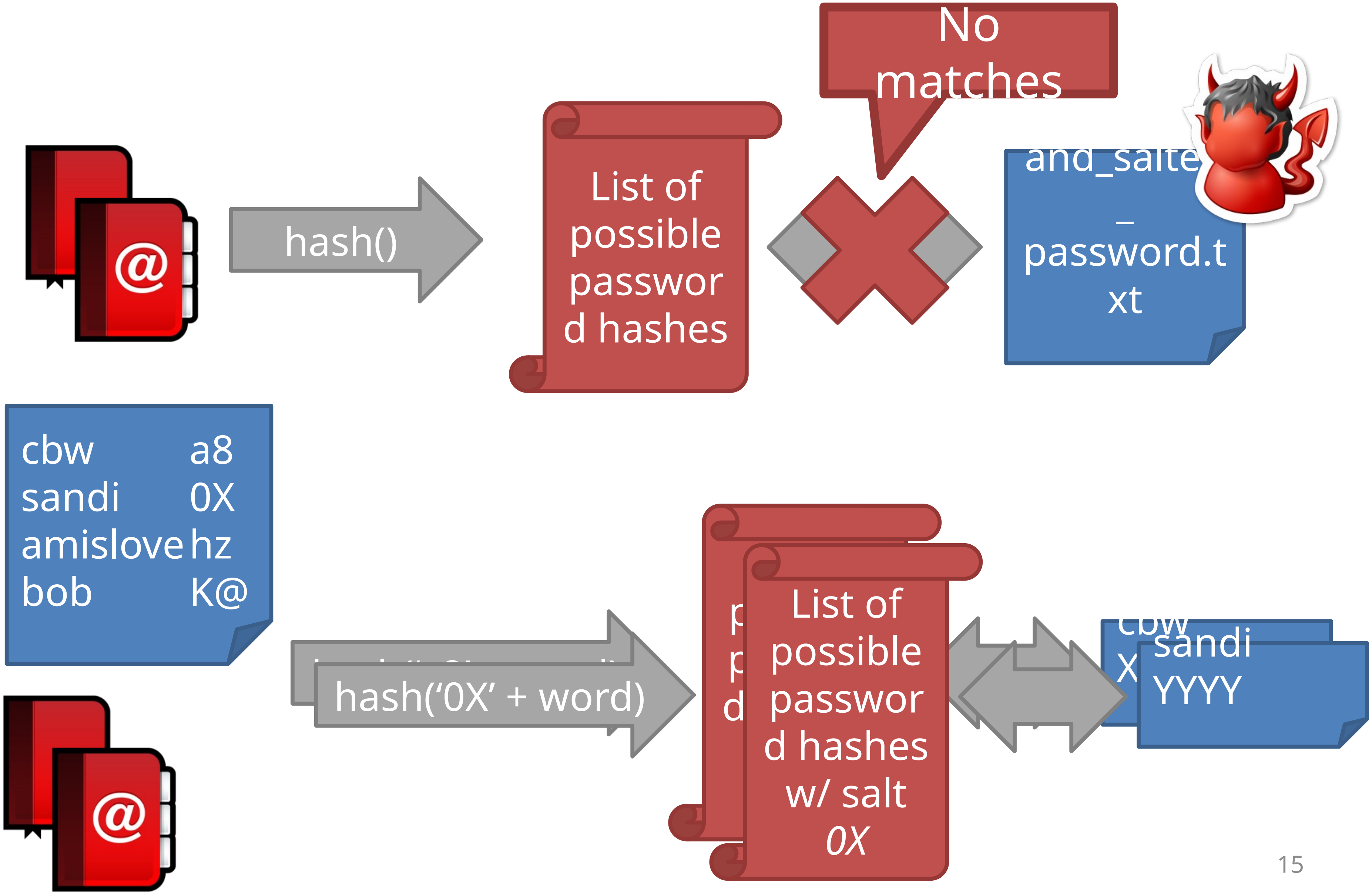
First two characters are the salt

/etc/shadow

username:password:last:may:must:warn:expire:disable:reserved

cbw:a8ge08pfz4wuk:9479:0:10000::::
amislove:hz560s9vnalh1:8172:0:10000::::

Attacking Salted Passwords



No matches
hash()
List of possible password hashes
and_salte _ password.txt
hash('0X' + word)
p p d
List of possible password hashes w/ salt *0X*
cbw X
sandi YYYY

cbw	a8
sandi	0X
amislove	hz
bob	K@

Breaking Hashed Passwords

- **Stored passwords should always be salted**
  - Forces the attacker to brute-force each password individually
- Problem: it is now possible to compute cryptographic hashes very quickly
  - GPU computing: hundreds of small CPU cores
  - nVidia GeForce GTX Titan Z: 5,760 cores
  - GPUs can be rented from the cloud very cheaply
    - 2x GPUs for $0.65 per hour (2014 prices)

Examples of Hashing Speed

- A modern x86 server can hash all possible 6 character long passwords in 3.5 hours
  - Upper and lowercase letters, numbers, symbols
  - $(26+26+10+32)^6$ = 690 billion combinations
- A modern GPU can do the same thing in 16 minutes
- Most users use (slightly permuted) dictionary words, no symbols
  - Predictability makes cracking much faster
  - Lowercase + numbers → $(26+10)^6$ = 2B combinations

Hardening Salted Passwords

- Problem: typical hashing algorithms are too fast
  - Enables GPUs to brute-force passwords
- Solution: use hash functions that are designed to be **slow**
  - Examples: bcrypt, scrypt, PBKDF2
  - These algorithms include a work factor that increases the time complexity of the calculation
  - scrypt also requires a large amount of memory to compute, further complicating

bcrypt Example

- Python example; install the *bcrypt* package

```
[cbw@ativ9 ~] python
>>> import bcrypt
>>> password = “my super secret password”
>>> fast_hashed = bcrypt.hashpw(password, bcrypt.gensalt(0))
>>> slow_hashed = bcrypt.hashpw(password, bcrypt.gensalt(12))
>>> pw_from_user = raw_input(“Enter your password:”)
>>> if bcrypt.hashpw(pw_from_user, slow_hashed) == slow_hashed:
…        print “It matches! You may enter the system”
…   else:
…        print “No match. You may not proceed”
```

Work factor

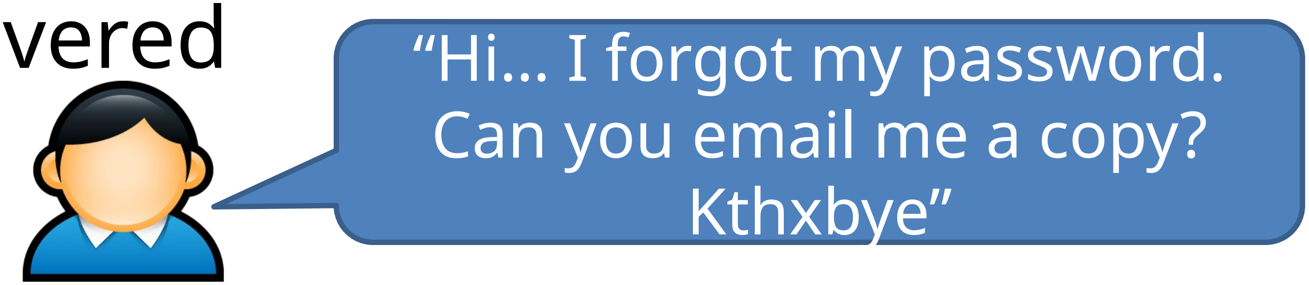
Password Storage Summary

1. **Never store passwords in plain text**
2. **Always salt and hash passwords before storing them**
3. **Use hash functions with a high work factor**

- These rules apply to any system that needs to authenticate users
  - Operating systems, websites, etc.

Password Recovery/Reset

- Problem: hashed passwords cannot be recovered



"Hi… I forgot my password. Can you email me a copy? Kthxbye"

- This is why systems typically implement password reset
  - Use out-of-band info to authenticate the user
  - Overwrite hash(old_pw) with hash(new_pw)
- Be careful: its possible to crack password reset

- Authentication
- Access Control
- Mandatory Access Control

Status Check

- At this point, we can authenticate users
  - And we are securely storing their password
- How do we control what users can do, and what they can access?

Simple Access Control

- Basic security in an OS is based on access control
- Simple policies can be written as an access control matrix
  - Specifies actions that actors can take on objects
  - Unix actions: read, write and execute
    - For directories, x → traverse

	file 1	file 2	dir 1	file 3
user 1	---	r--	---	rw-
user 2	r--	r--	rwx	r--

Users and Groups on Unix

- Actors are users, each user has a unique ID

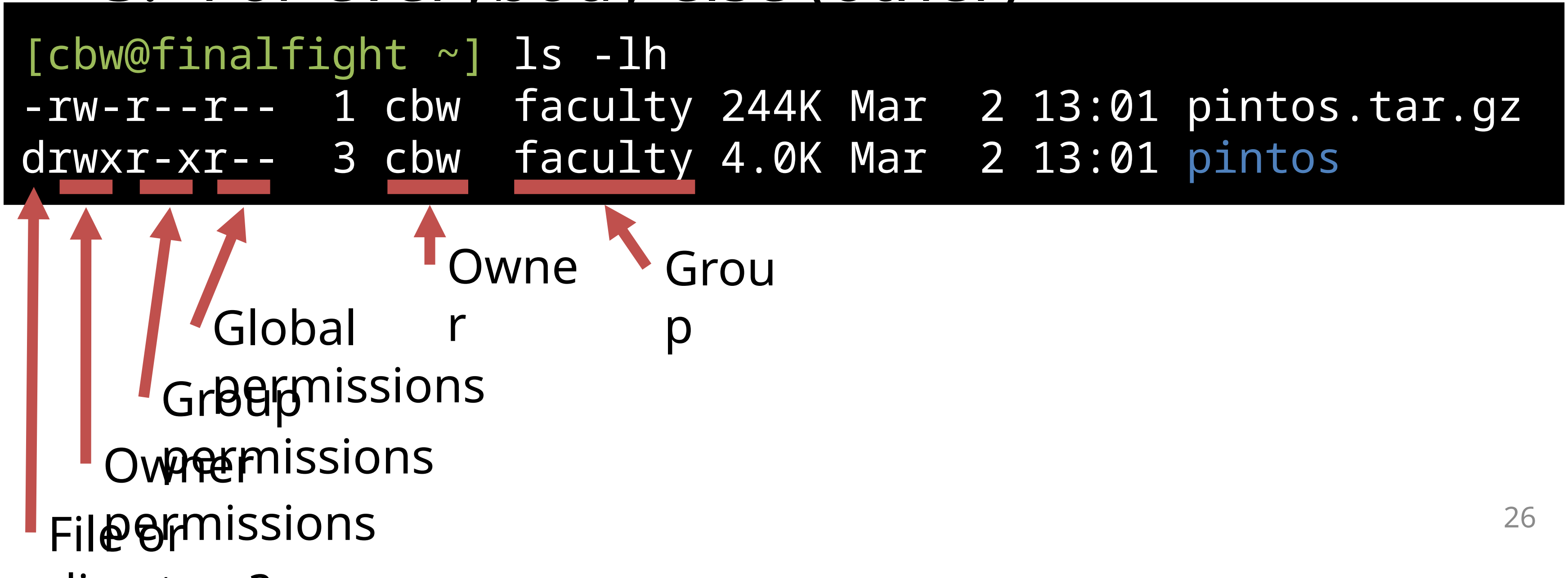
/etc/passwd

```
cbw:x:13273:65100:Christo Wilson:/home/cbw/:/bin/bash
```

```
[cbw@finalfight ~] id cbw
uid=13273(cbw) gid=65100(faculty) groups=65100(faculty),
1314(cs5700f13),1316(cs5750f13),1328(cs5600sp13)
```

File Permissions on Unix

- Files and directories have an owner and a group
- Three sets of permissions:
  1. For the owner
  2. For members of the group
  3. For everybody else (other)



```
[cbw@finalfight ~] ls -lh
-rw-r--r--   1 cbw   faculty 244K Mar   2 13:01 pintos.tar.gz
drwxr-xr--   3 cbw   faculty 4.0K Mar   2 13:01 pintos
```

Owner

Group

Global permissions

Group permissions

Owner permissions

File or directory?

Permission Examples

```
[cbw@finalfight ~] ls -lh
-rw-r--r--  1 cbw  faculty 244K Mar  2 13:01 pintos.tar.gz
drwxr-xr--  3 cbw  faculty 4.0K Mar  2 13:01 pintos
```

cbw:faculty

- May read both objects
- May modify the file
- May not execute the file
- May enter the directory
- May add files to the directory
- May modify the permissions of both objects

amislove:faculty

- May read both objects
- May not modify the file
- May not execute the file
- May enter the directory
- May not add files to the directory
- May not modify permissions

bob:student

- May read both objects
- May not modify the file
- May not execute the file
- May not enter the directory
- May not add files to the directory
- May not modify permissions

Encoding the Access Control Matrix

	file 1	file 2	dir 1	file 3
user 1	---	r--	---	rw-
user 2	r--	r--	rwx	r--
user 3	r--	r--	---	---
user 4	rw-	rwx	---	---

group 1 = {user 2, user 3, user 4}

group 2 = {user 1, user 2}

-, group = group 1,

-, group = group 1,

rwx---r--

- dir 1 – owner = user 2, group = group 1, rwx------

Modifying Permissions

- U y e ns
  of files hey o

u – user
g – group
o - other

+ add permissions
- remove permissions
= set permissions

r – read
w – write
x - executable

```
[cbw@finalfight ~] ls -lh
-rw------- 1 amislove faculty 5.1K Jan 23 11:25 alans_file
-rw------- 4 cbw      faculty 3.5K Jan 23 11:25 christos_file
[cbw@finalfight ~] chmod ugo+rw alans_file
chmod: changing permissions of `alans_file': Operation not permitted
[cbw@finalfight ~] chmod go+r christos_file
[cbw@finalfight ~] chmod u+w christos_file
[cbw@finalfight ~] chmod u-r christos_file
[cbw@finalfight ~] ls -lh
-rw------- 1 amislove faculty 5.1K Jan 23 11:25 alans_file
--wxr--r-- 4 cbw      faculty 3.5K Jan 23 11:25 christos_file
```

Advanced chmod

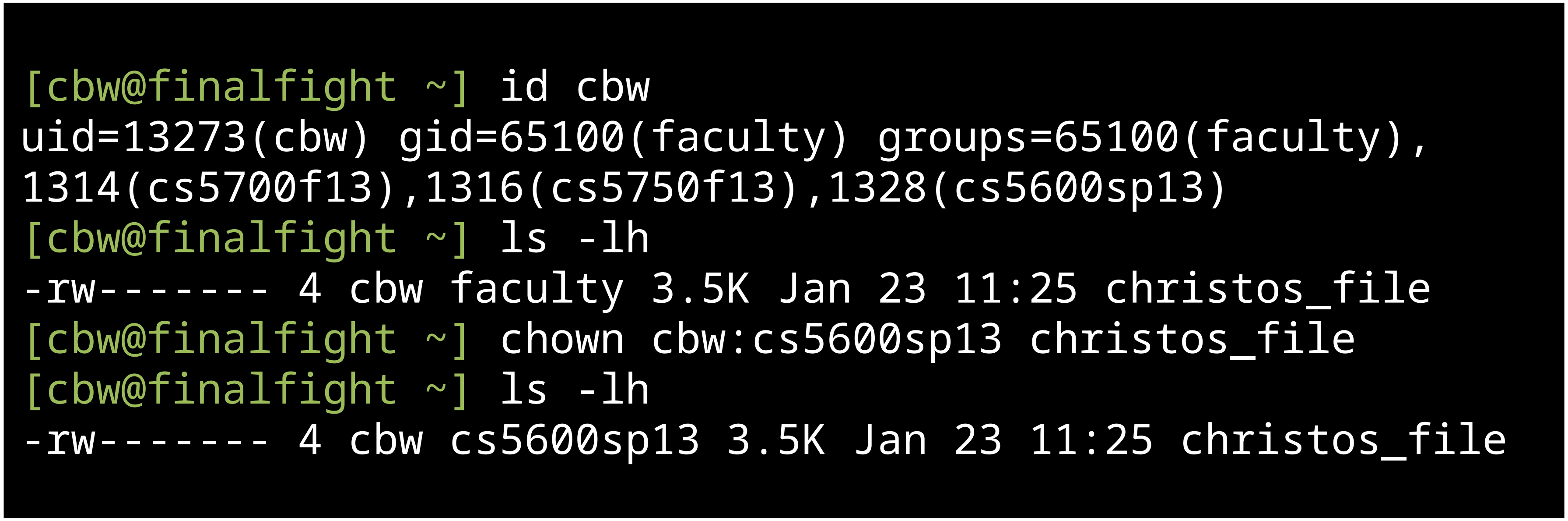
- Sometimes you'll see chmod commands in numerical format

```
[cbw@finalfight ~] chmod 755 christos_file
```

- What do the numbers mean?
  - Permissions (rwx) are stored in binary (000)
  - Example: rw- is 110, or 6
  - Three permission groups, hence three number
- Examples:

777	
755	

Modifying Users and Groups



```
[cbw@finalfight ~] id cbw
uid=13273(cbw) gid=65100(faculty) groups=65100(faculty),
1314(cs5700f13),1316(cs5750f13),1328(cs5600sp13)
[cbw@finalfight ~] ls -lh
-rw------- 4 cbw faculty 3.5K Jan 23 11:25 christos_file
[cbw@finalfight ~] chown cbw:cs5600sp13 christos_file
[cbw@finalfight ~] ls -lh
-rw------- 4 cbw cs5600sp13 3.5K Jan 23 11:25 christos_file
```

- Users may not change the owner of a file*
  - Even if they own it
- Users may only change to a group they belong to

* unless you are root

Permissions of Processes

- Processes also have permissions
  - They have to, since they read files, etc.
- What is the user:group of a process?
  1. ~~The user:group of the executable~~* file?
  2. The user:group of the user running the process?
- Processes inherit the credentials of the user who runs them
  - Child processes inherit their parents credentials

* except when the program is setuid

Privileged Operations

- Other aspects of the OS may also require special privileges
- Fortunately, on Unix most aspects of the system are represented as files
  - E.g. /dev contains devices like disks
  - Formatting a disk requires permissions to /dev/sd*
- Processes may only signal other processes with the same user ID*
  - Otherwise, you could send SIGKILL to other user's processes

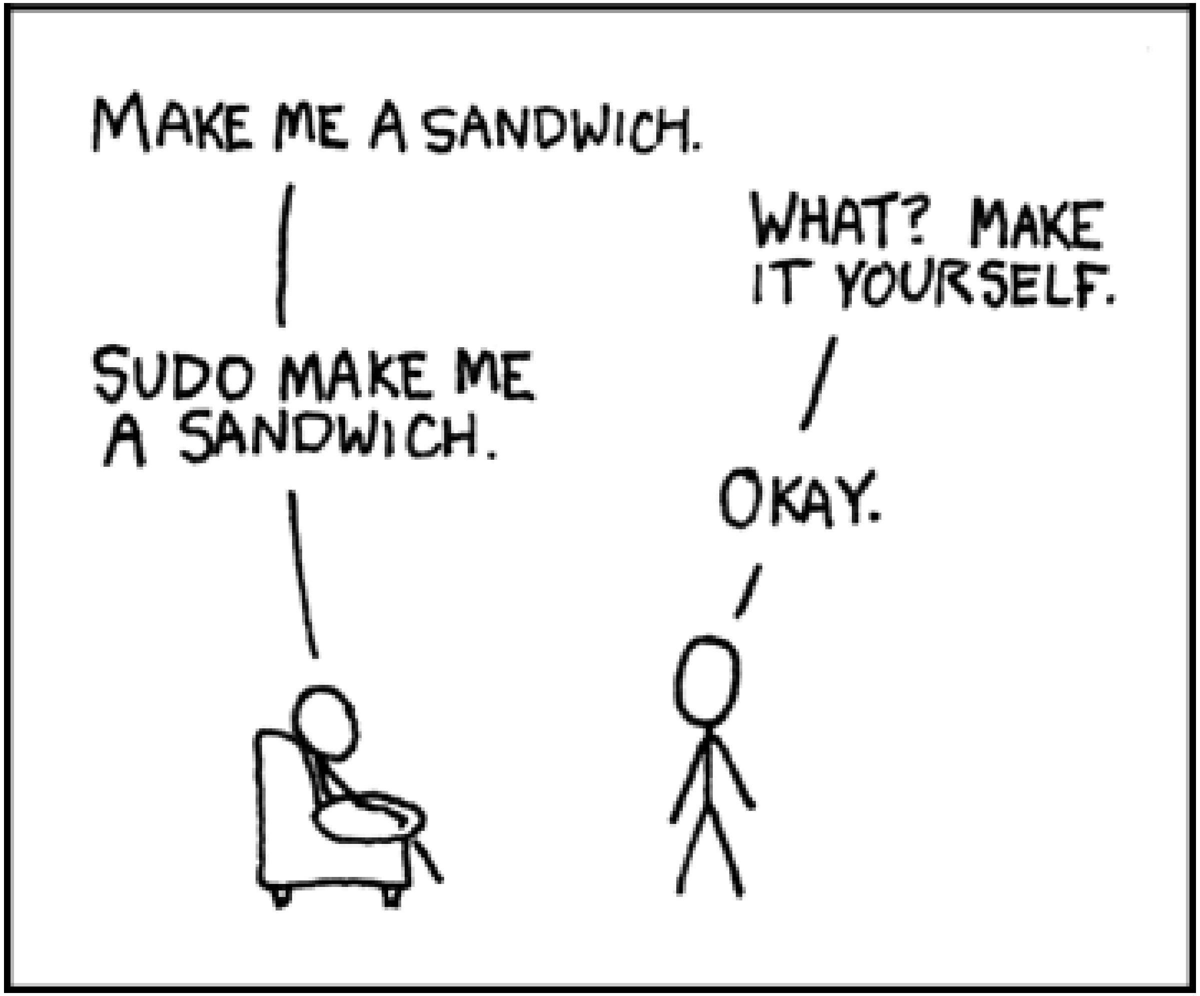
* unless the process is root

The Exception to Every Rule

- On Unix, the root user (ID=0) can do whatever it wants
  - Access any file
  - Change any permission
- On Windows, called the Administrator account
- **Your everyday user account should never be Admin/root**

Ways to Access Root

- Suppose you need to run a privileged command
  - Example: $ *apt-get install python*
- How can you get root privileges?
  1. Log in as root
     - $ ssh root@mymachine.ccs.neu.edu
  2. The Switch User command (su)
     - $ su
     - Opens a new shell with as root:root
  3. The Switch User Do Command (sudo)
     - $ sudo apt-get install python
     - Runs the given command as root:root

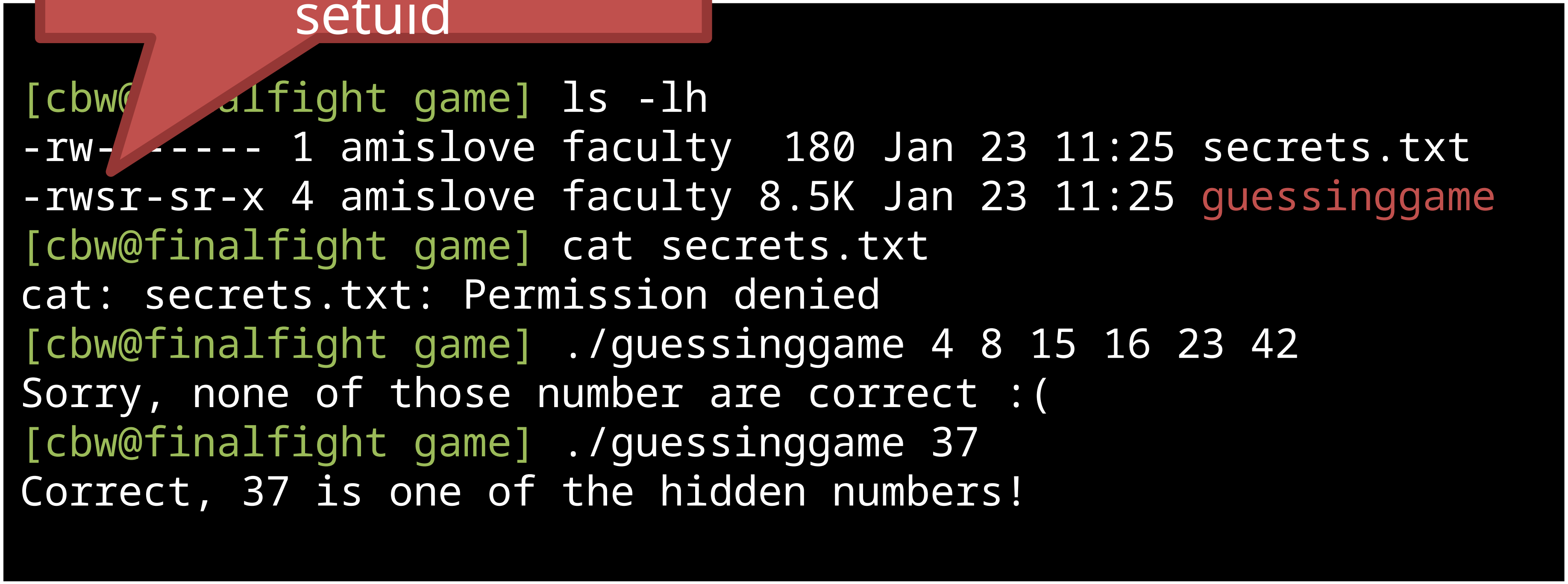

MAKE ME A SANDWICH.
WHAT? MAKE IT YOURSELF.
SUDO MAKE ME A SANDWICH.
OKAY.

Set Effective User ID

- In some cases, you may need a program to run as the file owner, not the invoking user
- Imagine a command-line guessing game
  - Users may input numbers as guesses
    - The user should not be able to read the file with the correct answers
  - Program must check if guesses are correct
    - The program must be able to read the file with correct answers

setuid example

Game executable is setuid



```
[cbw@finalfight game] ls -lh
-rw------- 1 amislove faculty  180 Jan 23 11:25 secrets.txt
-rwsr-sr-x 4 amislove faculty 8.5K Jan 23 11:25 guessinggame
[cbw@finalfight game] cat secrets.txt
cat: secrets.txt: Permission denied
[cbw@finalfight game] ./guessinggame 4 8 15 16 23 42
Sorry, none of those number are correct :(
[cbw@finalfight game] ./guessinggame 37
Correct, 37 is one of the hidden numbers!
```

How to setuid

```
[cbw@finalfight tmp] gcc -o my_program my_program.c
[cbw@finalfight tmp] ls -lh
-rwxr-xr-x 1 cbw faculty 2.3K Jan 23 11:25 my_program
[cbw@finalfight tmp] chmod u+s my_program
[cbw@finalfight tmp] ls -lh
-rwsr-xr-x 1 cbw faculty 2.3K Jan 23 11:25 my_program
```

- **Be very careful with setuid**
  - You are giving other users the ability to run a program as **you**, with **your privileges**
- Programs that are setuid=root should drop privileges
  - Google “setuid demystified” for more info

setuid and scripts

```
[cb
-rw                                             1:25 server.py
[cbw@finalfight tmp] ./s                   y
```

This is known as a TOCTOU vulnerability:
Time-Of-Check, Time-of-Use

- Steps to run a setuid script
  1. Kernel checks setuid bit of the script
  2. Kernel loads the interpreter (i.e. python) with setuid permissions
  3. Interpreter executes the script
- **Never set a script as setuid**

Replace server.py with modified, evil script

Limitations of the Unix Model

- The Unix model is very simple
  - Users and groups, read/write/execute
- Not all possible policies can be encoded

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rw-	rwx

- file 1: two users have high privileges
  - If user 3 and user 4 are in a group, how to give user 2 read and user 1 nothing?
- file 2: four distinct privilege levels
  - Maximum of three levels (user, group, other)

Access Control Lists

- ACLs are explicit rules that grant or deny permissions to users and groups
  - Typically associated with files as meta-data

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rw-	rwx
user	rw-	---

- file 1: owner = user 4, group = {user 4, user 3}

  owner: rw- group: rw-

  user 2: r-- other: ---

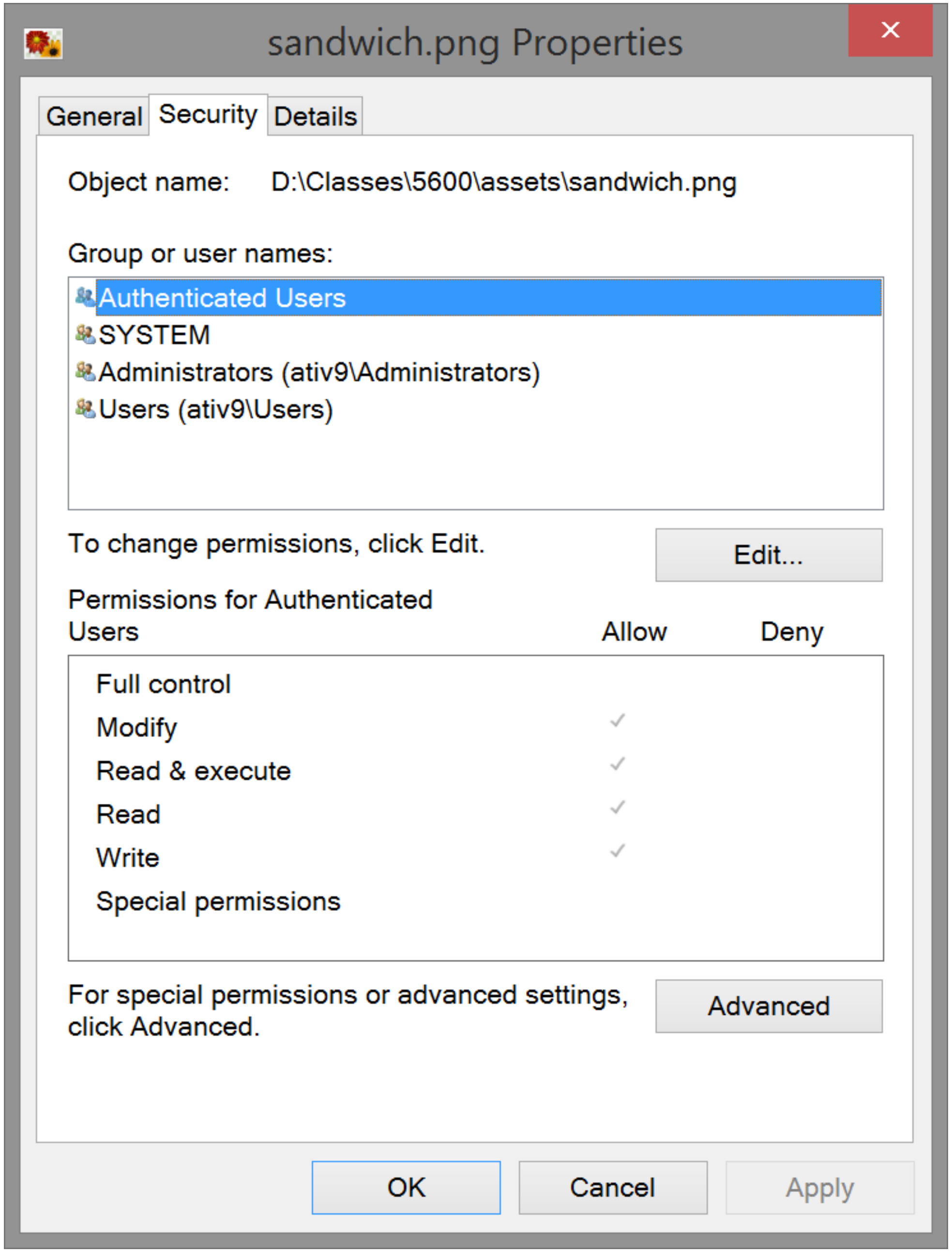
- file 2: owner = user 3, group = {user 3, user 1}

  owner: rwx group: rw-

  user 2: r-- other: ---

More ACLs

- OSX and some versions of Linux also support ACLs



Object name: D:\Classes\5600\assets\sandwich.png

Group or user names:

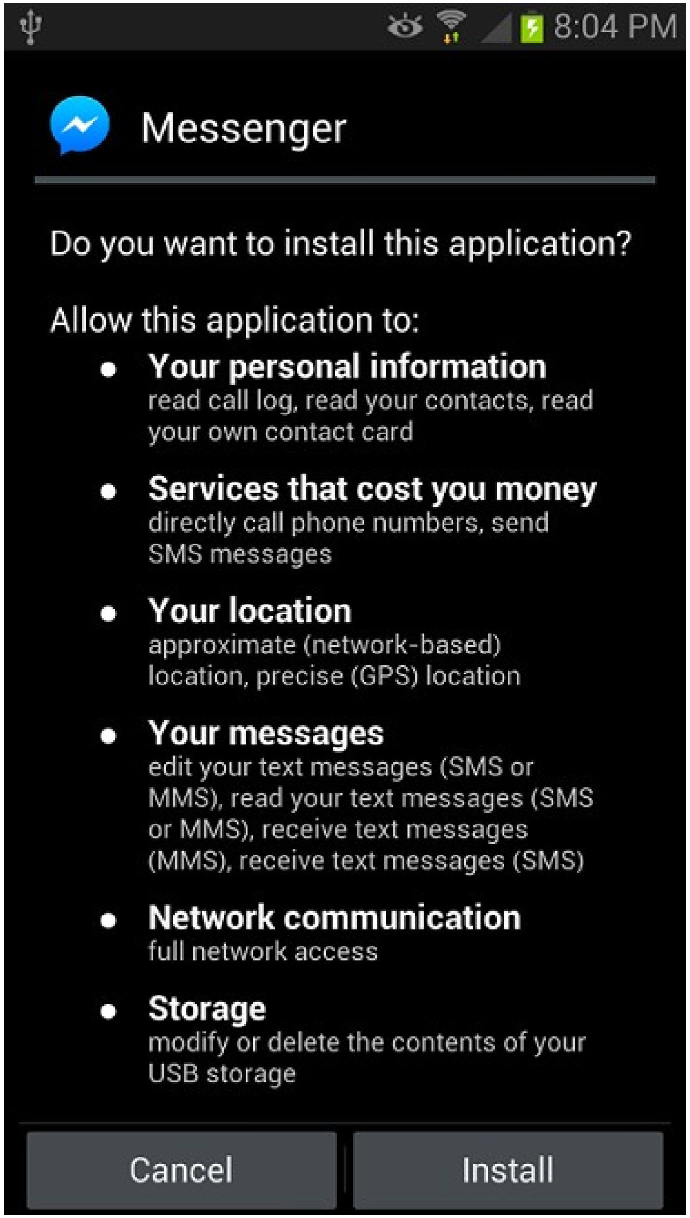
- Authenticated Users
- SYSTEM
- Administrators (ativ9\Administrators)
- Users (ativ9\Users)

To change permissions, click Edit.

Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

API Permissions



8:04 PM

Messenger

Do you want to install this application?

Allow this application to:

- **Your personal information**
  read call log, read your contacts, read your own contact card
- **Services that cost you money**
  directly call phone numbers, send SMS messages
- **Your location**
  approximate (network-based) location, precise (GPS) location
- **Your messages**
  edit your text messages (SMS or MMS), read your text messages (SMS or MMS), receive text messages (MMS), receive text messages (SMS)
- **Network communication**
  full network access
- **Storage**
  modify or delete the contents of your USB storage

- On Android, apps need permission to access some sensitive API calls
- Android is based on Linux
- Behind the scenes, each app is given its own user and group
- Kernel enforces permission checks when system calls are made

- Authentication
- Access Control
- Mandatory Access Control