

10/5/26

# COMP 435:



# *Computer Security Concepts*

**Lecture 19:** Confused- Deputy Attack & Capabilities, Authentication



The University  
of North Carolina  
at Chapel Hill

<https://tinyurl.com/comp435-fa26>

# Logistics Update

---



- Lab 2 Released! Due 10/14
- All in-class Friday WA days posted on assignments tab
  - Only 2 left! Yay
- Details about final project posted

# Plan for Class Today

---



- 10:10-10:25 – Confused Deputy & Capabilities
- 10:35-11:00 – Authentication Approaches

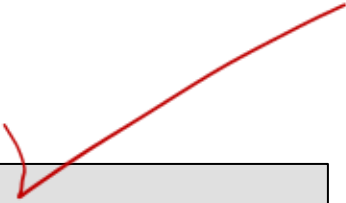
## Guiding Question 😊

How do we  
authenticate our  
systems?



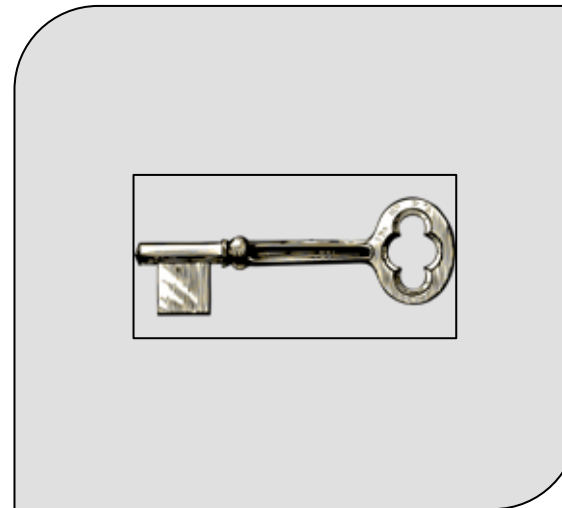
# Capabilities

# Mechanisms for Making Authorization Decisions



|           | BIBLIOG | TEMP | F   | HELPTXT | C_COMP | LINKER | SYS_CLOCK | PRINTER |
|-----------|---------|------|-----|---------|--------|--------|-----------|---------|
| USERA     | ORW     | ORW  | ORW | R       | X      | X      | R         | W       |
| USERB     | R       | -    | -   | R       | X      | X      | R         | W       |
| USERS     | RW      | -    | R   | R       | X      | X      | R         | W       |
| USERT     | -       | -    | -   | R       | X      | X      | R         | W       |
| SYS_MGR   | -       | -    | -   | RW      | OX     | OX     | ORW       | O       |
| USER_SVCS | -       | -    | -   | O       | X      | X      | R         | W       |

Access  
Control



Capabilities

Is this process  
authorized to access  
this resource?



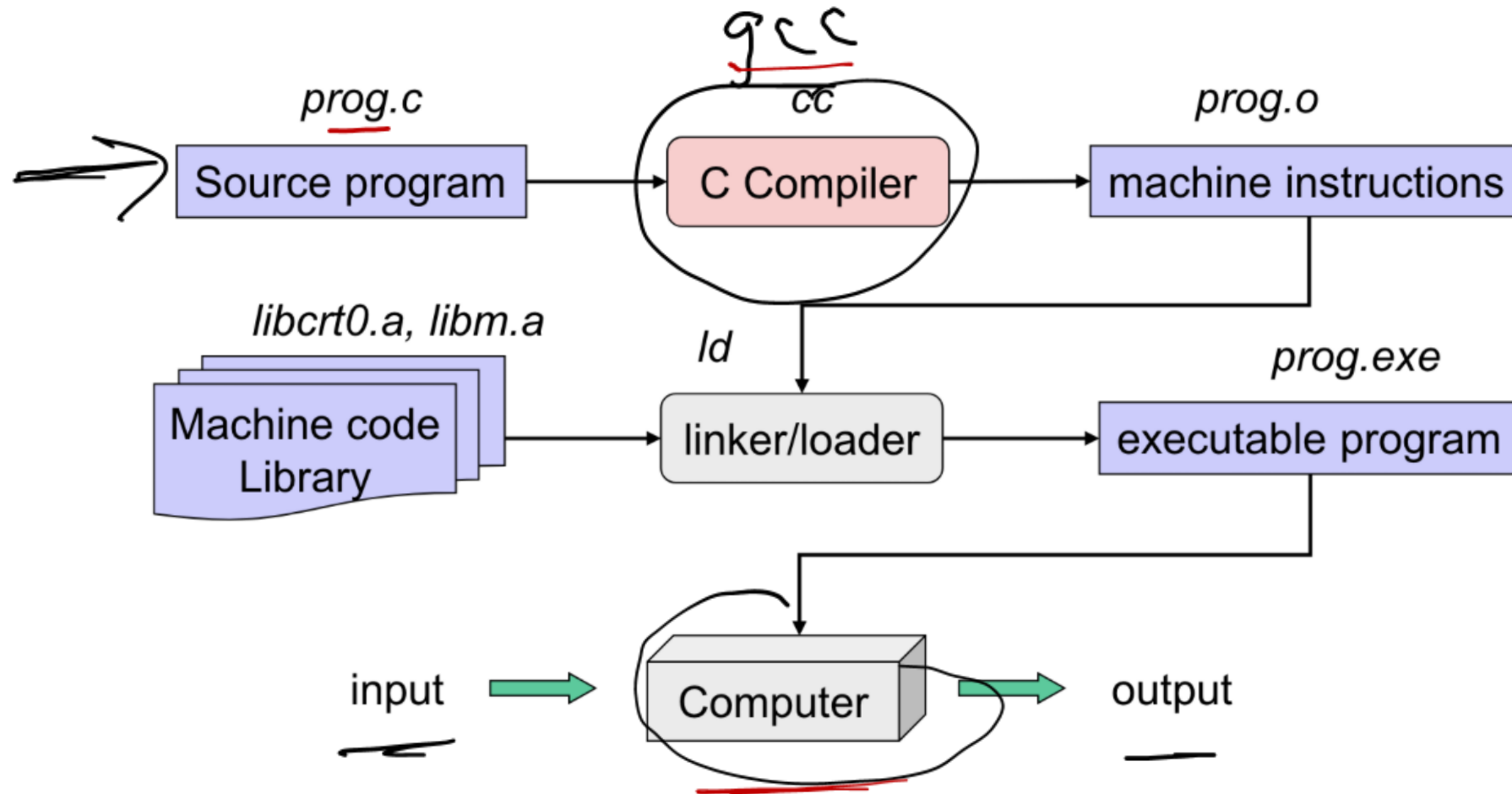
# Motivating Capabilities: *The Confused Deputy Attack*

# Quick Background: EUID



- Effective user ID (EUID): the user ID that the OS uses to determine **what permissions the program** has when it runs
- There is also Real user ID (RUID): ID of user who started the process.
- Most of the time EUID == RUID. ★
- When the `setuid` bit is set and a user runs that program, EUID becomes the file owner's UID (usually root), even though the RUID remains the users
  - This lets certain programs perform privileged ops on behalf of ordinary users

# Background: The Compiler



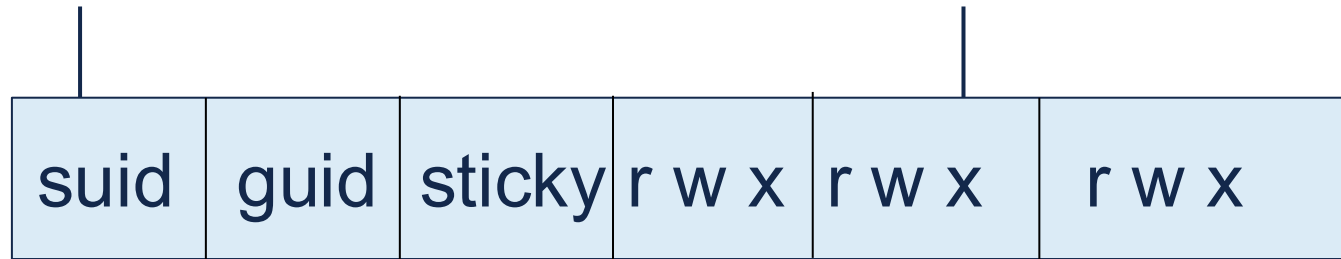
# SetUID

When the setUID bit is set on an executable file and a user executes the file → the effective UID of that user switches to that of the creator of the executable.



SetUID

Group



Owner

Users

myFile.exe



```
(1) int main(int argc, char *argv[]) {  
    /*compile user's source code*/
```

```
    // write out binary executable
```

```
(2) FILE *fp = fopen(argv[2], "w");  
    /*write compiled program to fp*/
```

```
    // append compiler usage stats to protected system file
```

```
(3) fp = fopen("/etc/compiler_stats", "a");  
    /*write usage stats to fp*/
```

```
}
```

Some pseudocode for  
the GCC compiler.



```
(1) int main(int argc, char *argv[]) {  
    /*compile user's source code*/
```

```
$ gcc prog.c -o prog
```

```
    // write out binary executable
```

```
(2) FILE *fp = fopen(argv[2], "w");
```

```
    /*write compiled program to fp*/
```

```
    // append compiler usage stats to protected system file
```

```
(3) fp = fopen("/etc/compiler_stats", "a");
```

```
    /*write usage stats to fp*/
```

```
}
```

```
(1) int main(int argc, char *argv[]) {  
    /*compile user's source code
```

```
$ gcc prog.c -o prog  
$ gcc prog.c -o "/etc/passwd"
```

```
    // write out binary executable
```

```
(2) FILE *fp = fopen(argv[2], "w");  
    /*write compiled program to fp*/
```

```
    // append compiler usage stats to protected system file
```

```
(3) fp = fopen("/etc/compiler_stats", "a");  
    /*write usage stats to fp*/
```

```
}
```

```
(1) int main(int argc, char *argv[]) {  
    /*compile code*/
```

```
$ gcc prog.c -o prog  
$ gcc prog.c -o "/etc/passwd"
```

```
// write
```

```
(2) FILE
```

```
/*write
```

```
// write
```

```
(3) fp
```

```
/*write to fp*/
```

```
}
```

This is a problem if the compiler is running with root privileges or as an administrator!

It will use its *own privileges* to open `/etc/passwd` for writing and overwrite the password file! Even if the user who ran the `gcc` command didn't have permission to do so!

# Ambient Authority

---



Def'n: all the extant authority of the current execution context

Problem: Authority can accrue as a program runs.

A given execution context will have multiple sets of permissions at any given time!

```
(1) int main(int argc, char *argv[]) {  
    /*compile user's source code
```

```
$ gcc prog.c -o prog  
$ gcc prog.c -o "/etc/passwd"
```

```
    // write out binary executable
```

```
(2) FILE *fp = fopen(argv[2], "w");  
    /*write compiled program to fp*/
```

```
    // append compiler usage stats to protected system file
```

```
(3) fp = fopen("/etc/compiler_stats", "a");  
    /*write usage stats to fp*/
```

```
}
```

# Confused Deputy Attack



A program running with multiple sets of permissions uses its ambient authority indiscriminately

In our example:

| Role        | Description                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Deputy      | The compiler, has authority to write to system files                                                                                 |
| Attacker    | Unprivileged user, who provides etc/passwd as an output file name                                                                    |
| Victim      | The system or admin who trusts the compiler                                                                                          |
| Confusion?? | The compiler doesn't realize it's acting on behalf of a user when it opens /etc/passwd for appending. It just uses its own privilege |

# Confused Deputy Attack

---



A program running with multiple sets of permissions uses its ambient authority indiscriminately

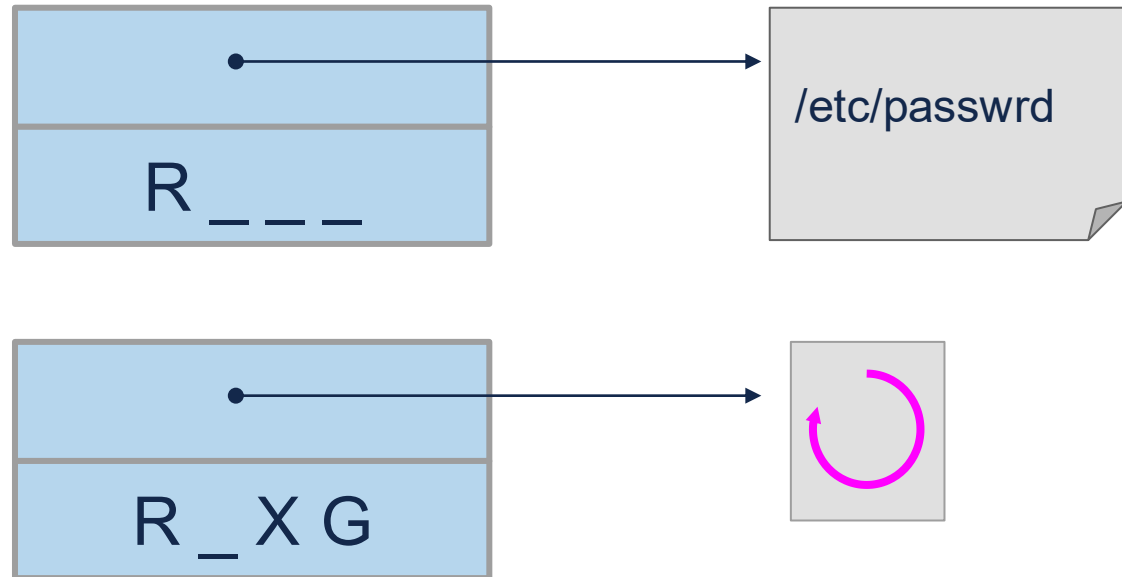
Access Control ***cannot***  
prevent the confused  
deputy attack

# Capabilities



- object reference + access right
- “fat pointer”
  - **Where** an object is AND
  - **What** you are allowed to do with it

READ, WRITE, EXECUTE,  
GRANT



```
(1) int main(int argc, char *argv[]) {  
    /*compile user's source code*/
```

```
    //write out binary executable
```

```
(2) cap *fcap = get_cap(argv[2], user_dir_cap);
```

Line 3: code uses the  
user's capabilities

```
(3) FILE *fp = cap_fopen(fcap);
```

```
    /*write compiled program to fp*/
```

```
// append compiler usage stats to protected system file
```

```
(4) fcap = get_cap("/etc/compiler_stats", sys_dir_cap);
```

```
(5) fp = cap_fopen(fcap);
```

Line 5: code uses the  
system's capabilities

```
    /*write usage stats to fp*/
```

```
}
```

```
(1) int main(int argc, char *argv[])
```

```
    /* compile user's source code
```

```
    //write out binary executable
```

```
(2) cap *fcap = get_cap(argv[2], user_dir_cap);
```

```
(3) FILE *fp = cap_fopen(fcap);
```

```
    /*write compiled program to fp
```

```
// append compiler usage stats to p
```

```
(4) fcap = get_cap("/etc/compiler_stats", sys_dir_cap);
```

```
(5) fp = cap_fopen(fcap);
```

```
    /*write usage stats to fp*/
```

```
}
```

```
$ gcc prog.c -o prog
```

```
$ gcc prog.c -o "/etc/passwd"
```

```
> ERROR: no capability for  
passwd file!
```

Fails at line 3. User capability  
does not provide access for the  
password file!

# Capabilities

---



## Properties:

- unforgeable token
- token directly ties access right to object
- no designation without authority

## Provides:

- no ambient authority
- supports principle of least privilege



# Another Confused Deputy Example: WS

---



**Chromium:** Open-source web browser

**Node.js:** a JavaScript runtime environment that enables developers to execute JavaScript code outside of a web browser

**Electron:** a framework for building apps w/ HTML, CSS, JS, packaged together with Chromium & Node.JS. Slack, VSCode & Discord are examples of apps built using Electron!



# User Authentication

# User Authentication



System



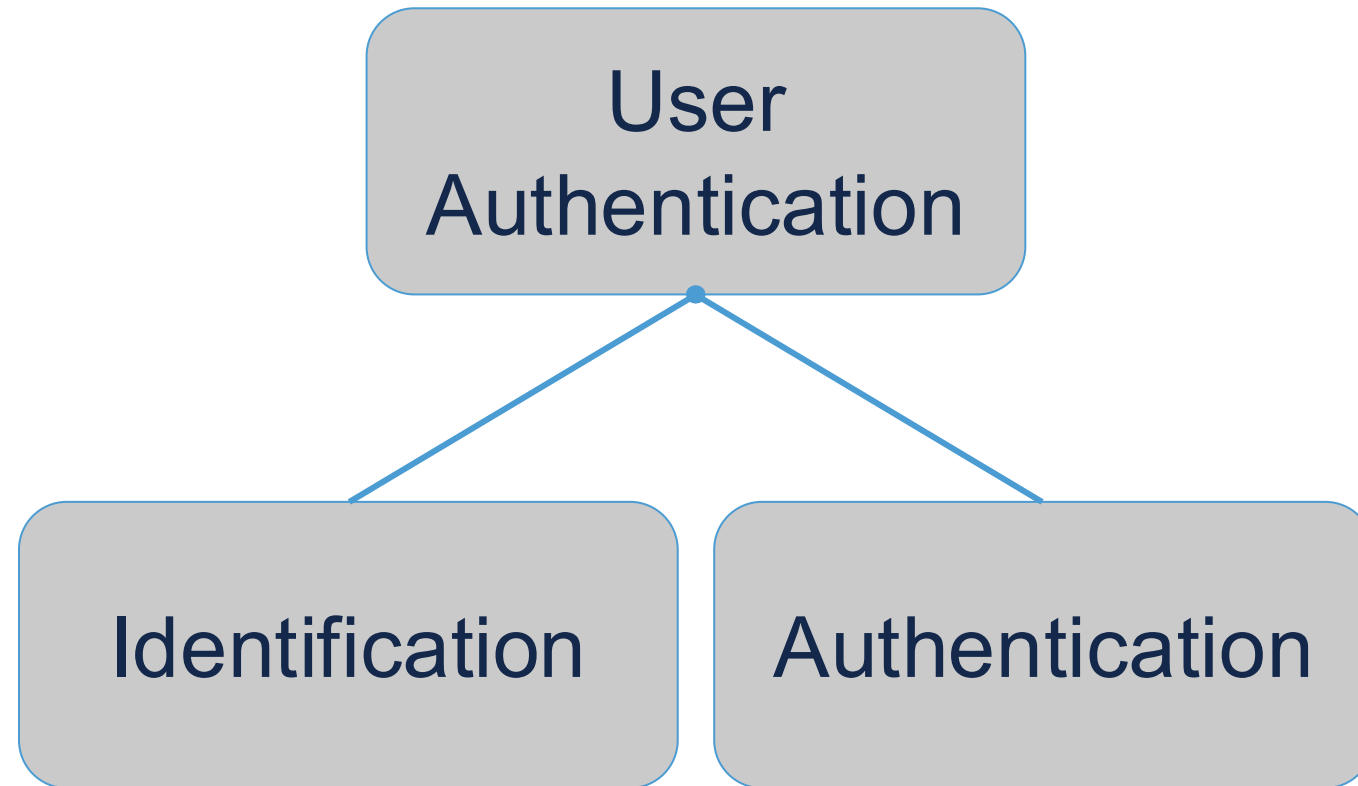
Policy

May Enter

+ family  
members  
+ friends

May Not Enter

- everyone  
else



# Identification



Def'n: Asserting who a person is

E.g., "Hello, my name is Kaki"

# Implicit



# Explicit



Hello  
my name is



# Authentication



Def'n: Proving an asserted identity

E.g., \*shows driver's license\*

# STATE ISSUED ID



# STUDENT ID





## Identification

- public
- well known

## Authentication

- private
- unforgeable
- authentic

# Means of Authentication



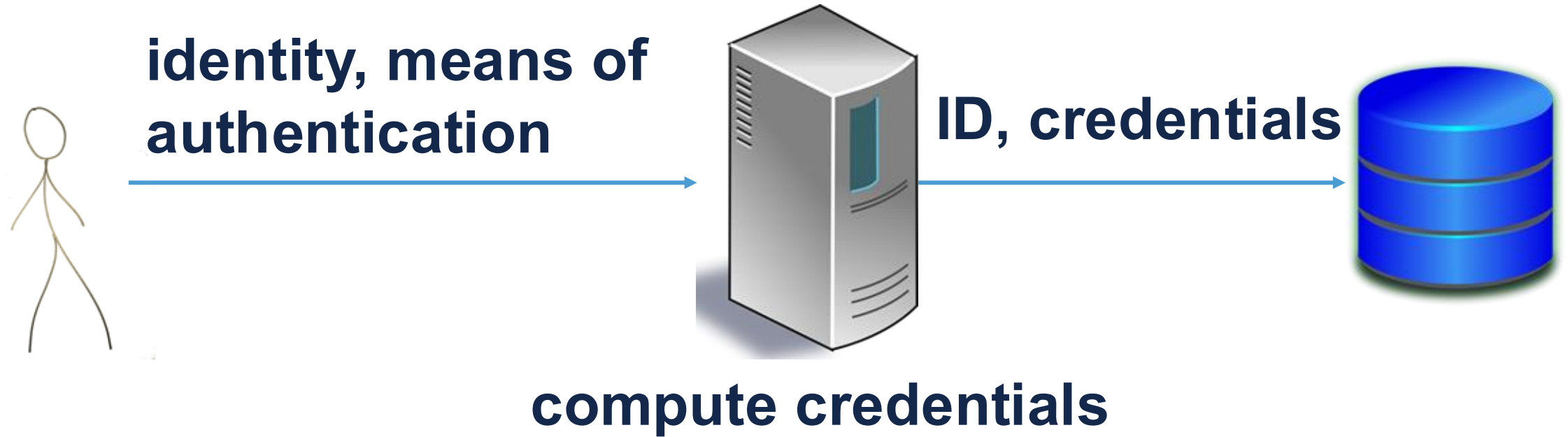
Which of the following is NOT a means of authenticating a user's identity?

1. Drivers license
2. Passport
3. Door key
4. A name tag

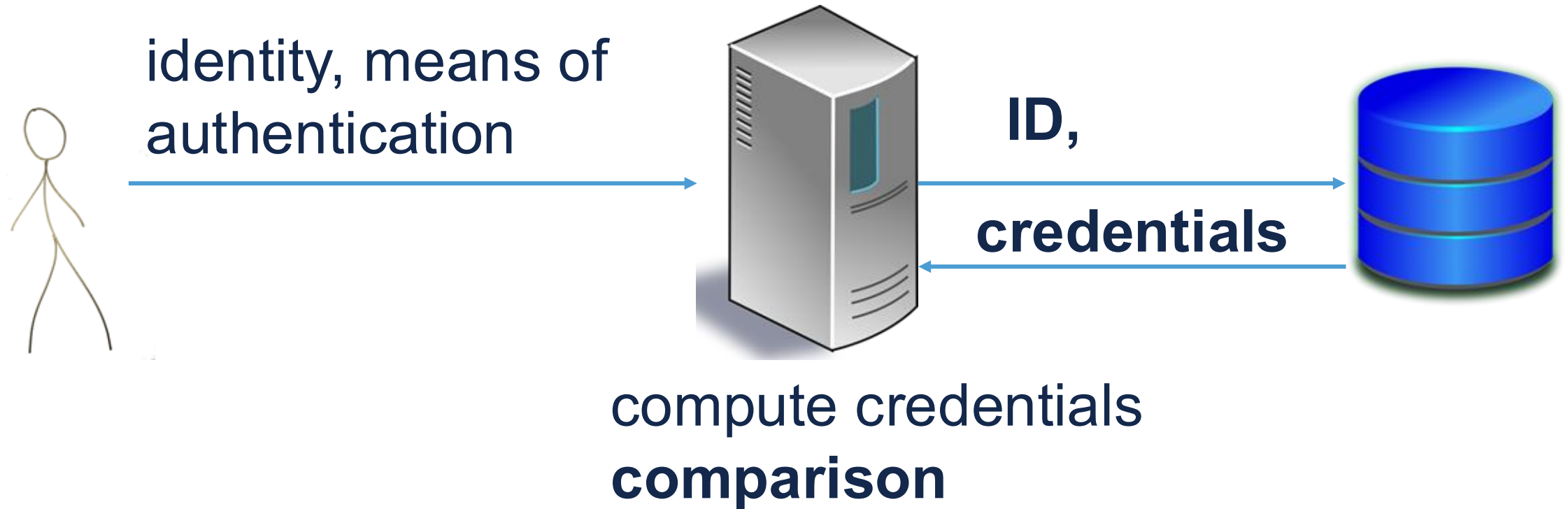


# User Authentication... in *Computer Systems!*

# Phase 1: Registration



# Phase 2: Authentication



# Means of Authentication



What you  
know

What you  
do or what  
you are

What you  
have

Where you  
are



# Passwords

# Means of Authentication

What you  
know

What you  
do or what  
you are

What you  
have

Where you  
are

passwords

# Passwords



- Ubiquitous
- Easy to use
- No extra hardware

Pros

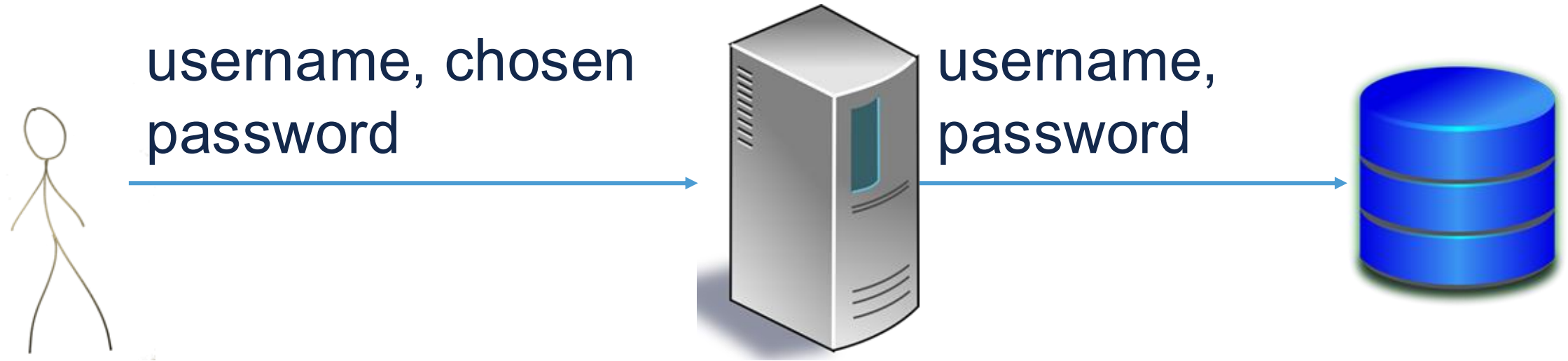


- Easy to attack
- Difficult to use well

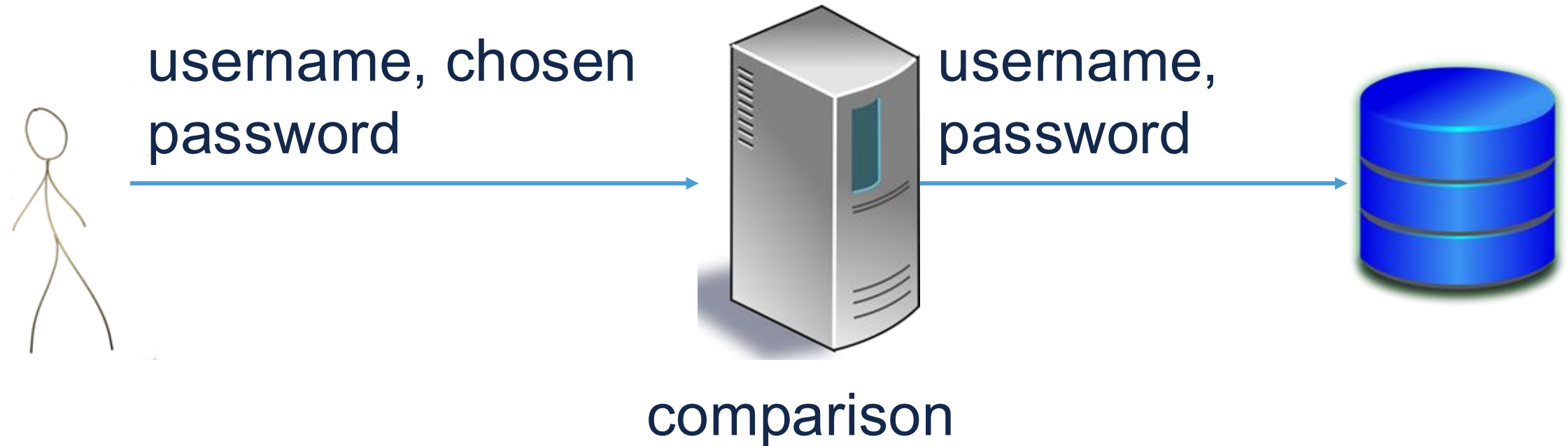
Cons

# Phase 1: Registration

---



## Phase 2: Authentication





# What is the attacker's chance of guessing correctly?

8-character passwords

printable characters (excluding space)

- 52 alphabetic
- 10 numeric
- 32 add'l (excluding space)

WS Q1