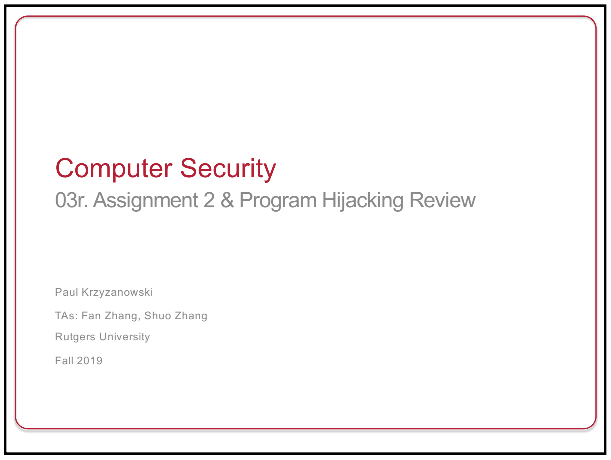


Computer Security

03r. Assignment 2 & Program Hijacking Review

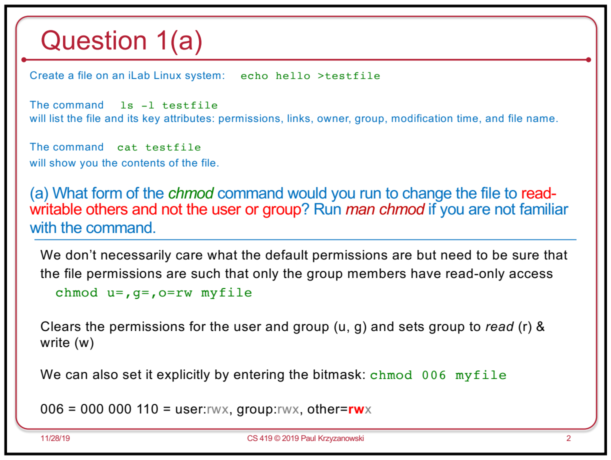
Paul Krzyzanowski

TAs: Fan Zhang, Shuo Zhang

Rutgers University

Fall 2019

1



Question 1(a)

Create a file on an iLab Linux system: `echo hello >testfile`

The command `ls -l testfile`
will list the file and its key attributes: permissions, links, owner, group, modification time, and file name.

The command `cat testfile`
will show you the contents of the file.

(a) What form of the *chmod* command would you run to change the file to read-writable others and not the user or group? Run *man chmod* if you are not familiar with the command.

We don't necessarily care what the default permissions are but need to be sure that the file permissions are such that only the group members have read-only access

```
chmod u=,g=,o=rw myfile
```

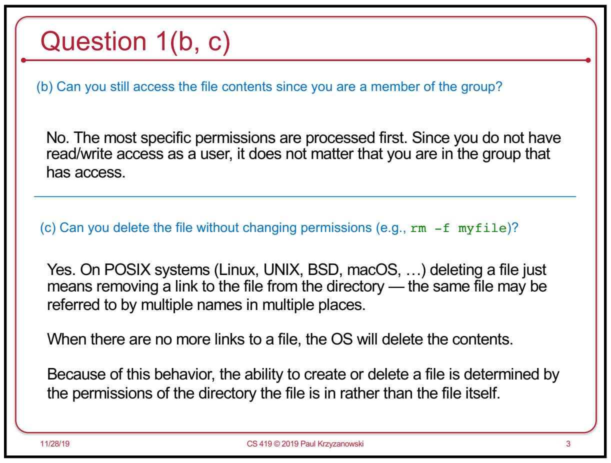
Clears the permissions for the user and group (u, g) and sets group to *read* (r) & write (w)

We can also set it explicitly by entering the bitmask: `chmod 006 myfile`

006 = 000 000 110 = user:rwx, group:rwx, other=**rw**x

11/28/19 CS 419 © 2019 Paul Krzyzanowski 2

2



Question 1(b, c)

(b) Can you still access the file contents since you are a member of the group?

No. The most specific permissions are processed first. Since you do not have read/write access as a user, it does not matter that you are in the group that has access.

(c) Can you delete the file without changing permissions (e.g., `rm -f myfile`)?

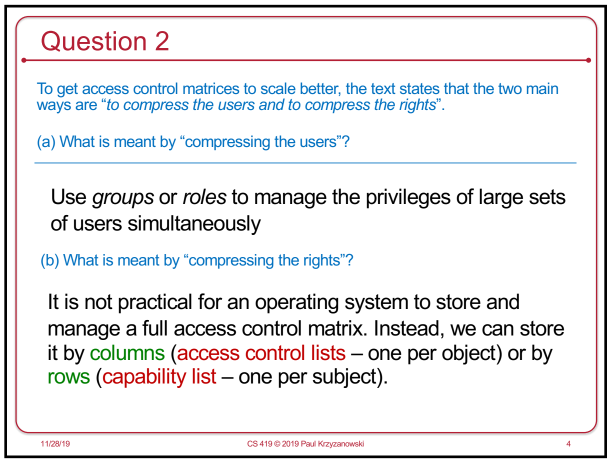
Yes. On POSIX systems (Linux, UNIX, BSD, macOS, …) deleting a file just means removing a link to the file from the directory — the same file may be referred to by multiple names in multiple places.

When there are no more links to a file, the OS will delete the contents.

Because of this behavior, the ability to create or delete a file is determined by the permissions of the directory the file is in rather than the file itself.

11/28/19 CS 419 © 2019 Paul Krzyzanowski 3

3



Question 2

To get access control matrices to scale better, the text states that the two main ways are "*to compress the users and to compress the rights*".

(a) What is meant by "compressing the users"?

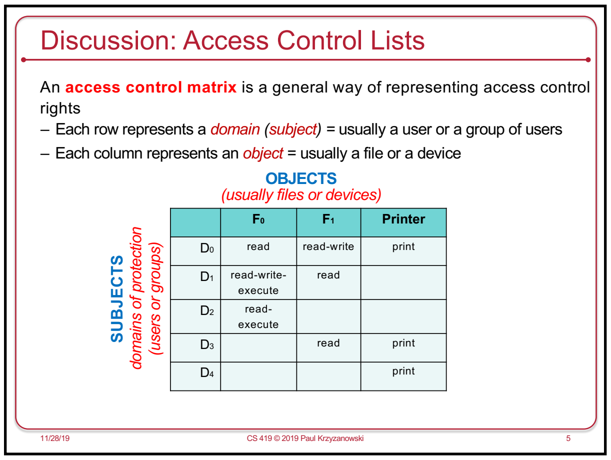
Use *groups* or *roles* to manage the privileges of large sets of users simultaneously

(b) What is meant by "compressing the rights"?

It is not practical for an operating system to store and manage a full access control matrix. Instead, we can store it by columns (access control lists – one per object) or by rows (capability list – one per subject).

11/28/19 CS 419 © 2019 Paul Krzyzanowski 4

4



Discussion: Access Control Lists

An **access control matrix** is a general way of representing access control rights

– Each row represents a *domain (subject)* = usually a user or a group of users

– Each column represents an *object* = usually a file or a device

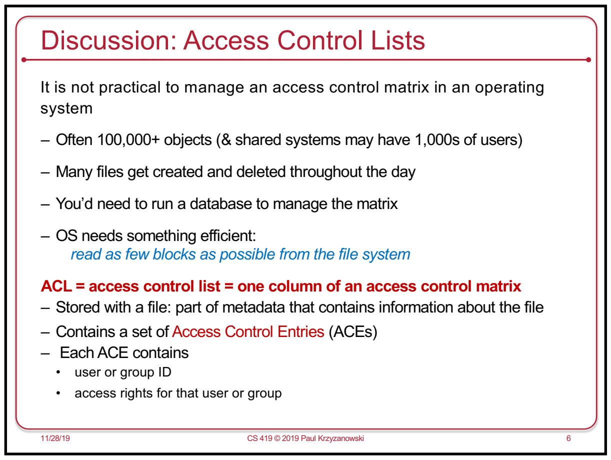
OBJECTS
(usually files or devices)

SUBJECTS
domains of protection (users or groups)

	F_0	F_1	Printer
D_0	read	read-write	print
D_1	read-write-execute	read	
D_2	read-execute		
D_3		read	print
D_4			print

11/28/19 CS 419 © 2019 Paul Krzyzanowski 5

5



Discussion: Access Control Lists

It is not practical to manage an access control matrix in an operating system

– Often 100,000+ objects (& shared systems may have 1,000s of users)

– Many files get created and deleted throughout the day

– You'd need to run a database to manage the matrix

– OS needs something efficient:
read as few blocks as possible from the file system

ACL = access control list = one column of an access control matrix

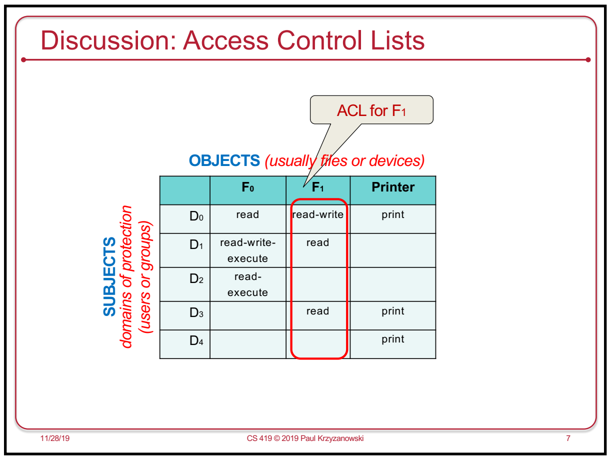
– Stored with a file: part of metadata that contains information about the file

– Contains a set of Access Control Entries (ACEs)

– Each ACE contains
- user or group ID
- access rights for that user or group

11/28/19 CS 419 © 2019 Paul Krzyzanowski 6

6

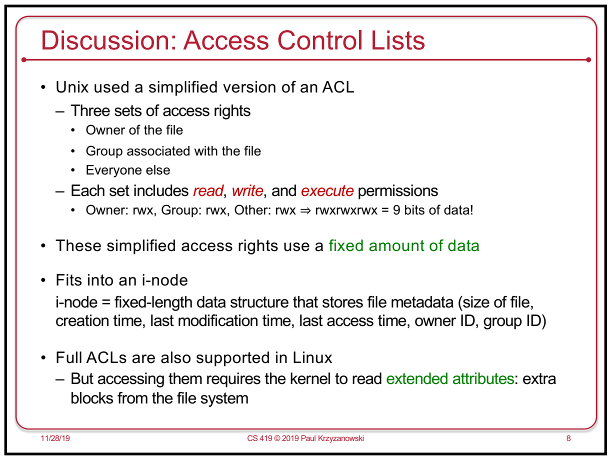


Discussion: Access Control Lists

	F₀	F₁	Printer
D₀	read	read-write	print
D₁	read-write-execute	read	
D₂	read-execute		
D₃		read	print
D₄			print

11/28/19 CS 419 © 2019 Paul Krzyzanowski 7

7

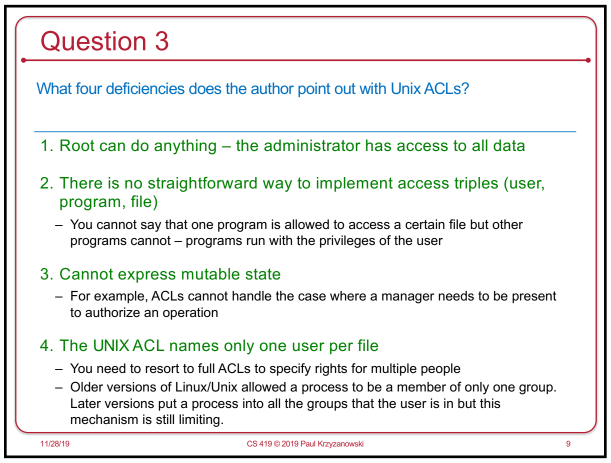


Discussion: Access Control Lists

- Unix used a simplified version of an ACL
  - Three sets of access rights
    - Owner of the file
    - Group associated with the file
    - Everyone else
  - Each set includes *read*, *write*, and *execute* permissions
    - Owner: rwx, Group: rwx, Other: rwx ⇒ rwxrwxrwx = 9 bits of data!
- These simplified access rights use a fixed amount of data
- Fits into an i-node
  i-node = fixed-length data structure that stores file metadata (size of file, creation time, last modification time, last access time, owner ID, group ID)
- Full ACLs are also supported in Linux
  - But accessing them requires the kernel to read extended attributes: extra blocks from the file system

11/28/19 CS 419 © 2019 Paul Krzyzanowski 8

8



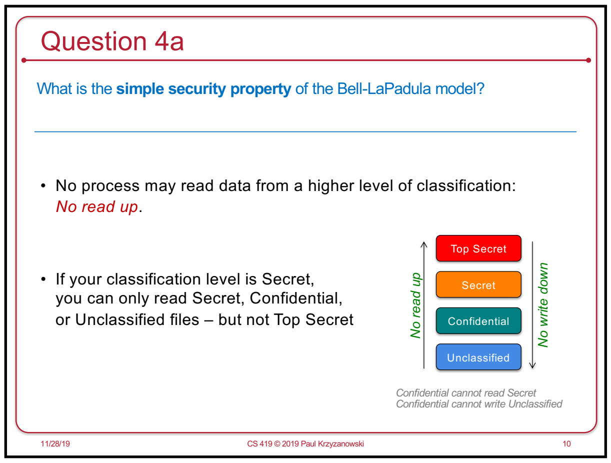
Question 3

What four deficiencies does the author point out with Unix ACLs?

1. Root can do anything – the administrator has access to all data
2. There is no straightforward way to implement access triples (user, program, file)
   - You cannot say that one program is allowed to access a certain file but other programs cannot – programs run with the privileges of the user
3. Cannot express mutable state
   - For example, ACLs cannot handle the case where a manager needs to be present to authorize an operation
4. The UNIX ACL names only one user per file
   - You need to resort to full ACLs to specify rights for multiple people
   - Older versions of Linux/Unix allowed a process to be a member of only one group. Later versions put a process into all the groups that the user is in but this mechanism is still limiting.

11/28/19 CS 419 © 2019 Paul Krzyzanowski 9

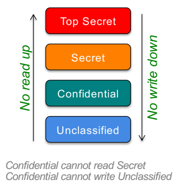
9



Question 4a

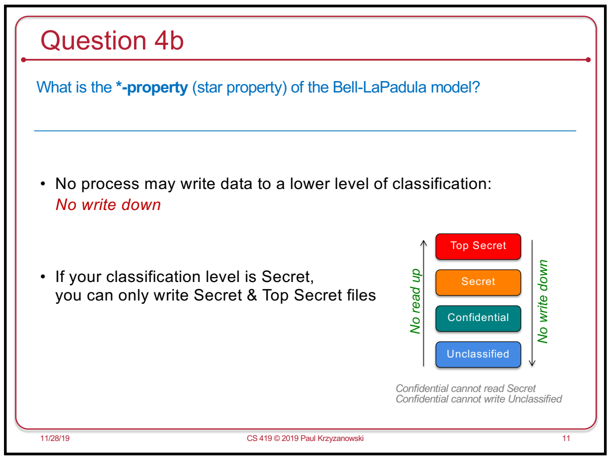
What is the **simple security property** of the Bell-LaPadula model?

- No process may read data from a higher level of classification: *No read up*.
- If your classification level is Secret, you can only read Secret, Confidential, or Unclassified files – but not Top Secret


Top Secret
Secret
Confidential
Unclassified
No read up
No write down
Confidential cannot read Secret
Confidential cannot write Unclassified

11/28/19 CS 419 © 2019 Paul Krzyzanowski 10

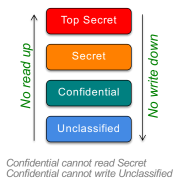
10



Question 4b

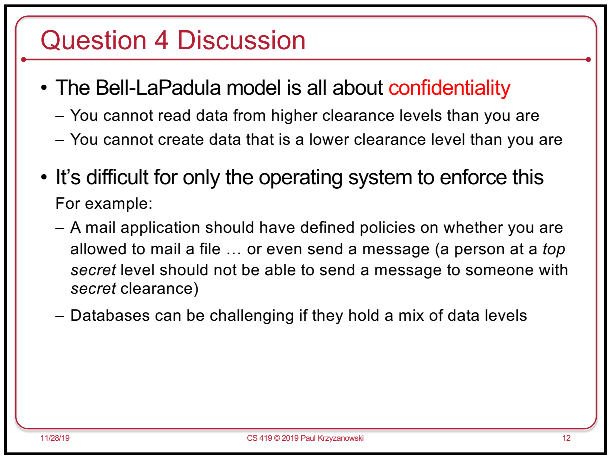
What is the ***-property** (star property) of the Bell-LaPadula model?

- No process may write data to a lower level of classification: *No write down*
- If your classification level is Secret, you can only write Secret & Top Secret files


Top Secret
Secret
Confidential
Unclassified
No read up
No write down
Confidential cannot read Secret
Confidential cannot write Unclassified

11/28/19 CS 419 © 2019 Paul Krzyzanowski 11

11

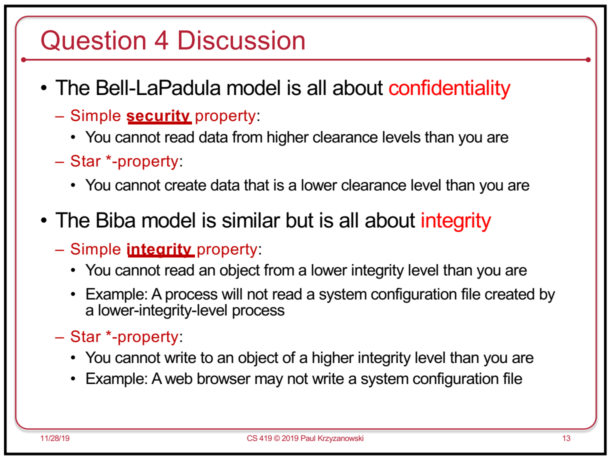


Question 4 Discussion

- The Bell-LaPadula model is all about confidentiality
  - You cannot read data from higher clearance levels than you are
  - You cannot create data that is a lower clearance level than you are
- It's difficult for only the operating system to enforce this
  For example:
  - A mail application should have defined policies on whether you are allowed to mail a file … or even send a message (a person at a *top secret* level should not be able to send a message to someone with *secret* clearance)
  - Databases can be challenging if they hold a mix of data levels

11/28/19 CS 419 © 2019 Paul Krzyzanowski 12

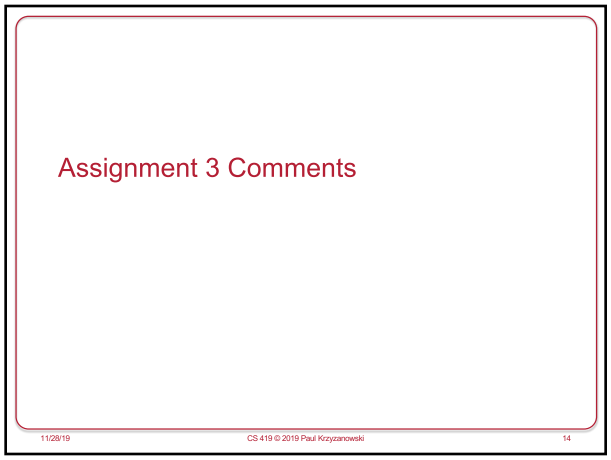
12



Question 4 Discussion

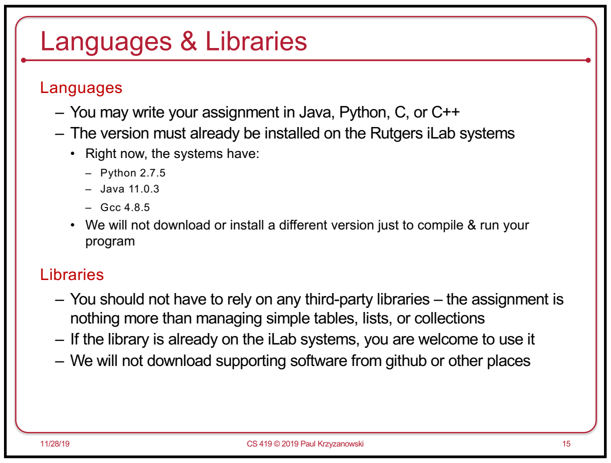
- The Bell-LaPadula model is all about confidentiality
  - Simple security property:
    - You cannot read data from higher clearance levels than you are
  - Star *-property:
    - You cannot create data that is a lower clearance level than you are
- The Biba model is similar but is all about integrity
  - Simple integrity property:
    - You cannot read an object from a lower integrity level than you are
    - Example: A process will not read a system configuration file created by a lower-integrity-level process
  - Star *-property:
    - You cannot write to an object of a higher integrity level than you are
    - Example: A web browser may not write a system configuration file

13



Assignment 3 Comments

14



Languages & Libraries

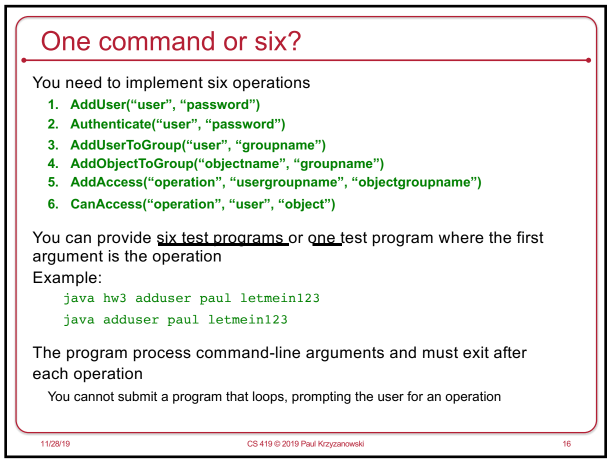
Languages

- You may write your assignment in Java, Python, C, or C++
- The version must already be installed on the Rutgers iLab systems
  - Right now, the systems have:
    - Python 2.7.5
    - Java 11.0.3
    - Gcc 4.8.5
  - We will not download or install a different version just to compile & run your program

Libraries

- You should not have to rely on any third-party libraries – the assignment is nothing more than managing simple tables, lists, or collections
- If the library is already on the iLab systems, you are welcome to use it
- We will not download supporting software from github or other places

15



One command or six?

You need to implement six operations

1. **AddUser("user", "password")**
2. **Authenticate("user", "password")**
3. **AddUserToGroup("user", "groupname")**
4. **AddObjectToGroup("objectname", "groupname")**
5. **AddAccess("operation", "usergroupname", "objectgroupname")**
6. **CanAccess("operation", "user", "object")**

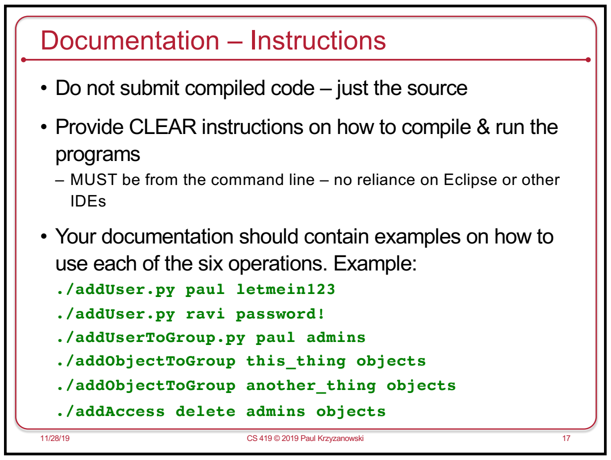
You can provide six test programs or one test program where the first argument is the operation
Example:

```
java hw3 adduser paul letmein123
java adduser paul letmein123
```

The program process command-line arguments and must exit after each operation

You cannot submit a program that loops, prompting the user for an operation

16

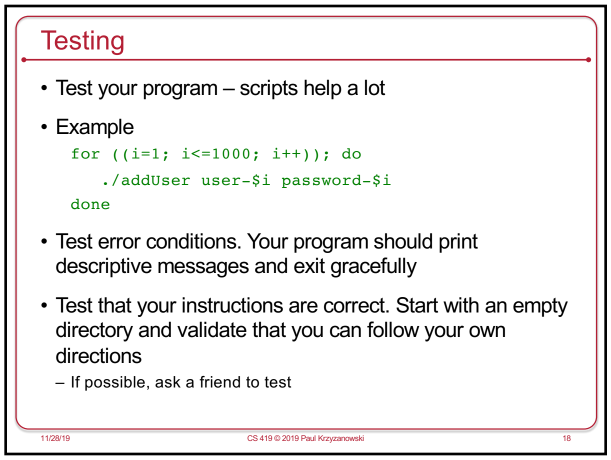


Documentation – Instructions

- Do not submit compiled code – just the source
- Provide CLEAR instructions on how to compile & run the programs
  - MUST be from the command line – no reliance on Eclipse or other IDEs
- Your documentation should contain examples on how to use each of the six operations. Example:

```
./addUser.py paul letmein123
./addUser.py ravi password!
./addUserToGroup.py paul admins
./addObjectToGroup this_thing objects
./addObjectToGroup another_thing objects
./addAccess delete admins objects
```

17



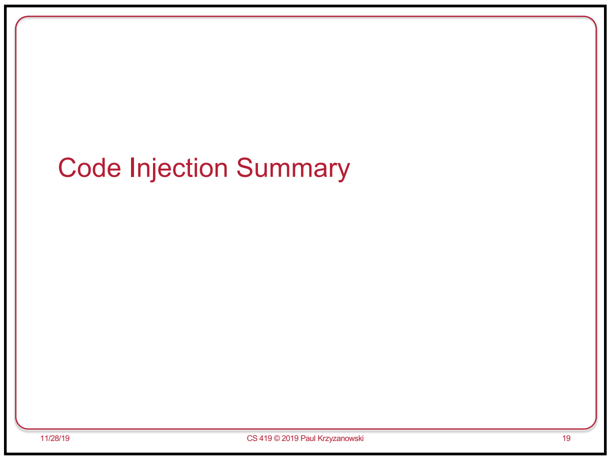
Testing

- Test your program – scripts help a lot
- Example

```
for ((i=1; i<=1000; i++)); do
    ./addUser user-$i password-$i
done
```

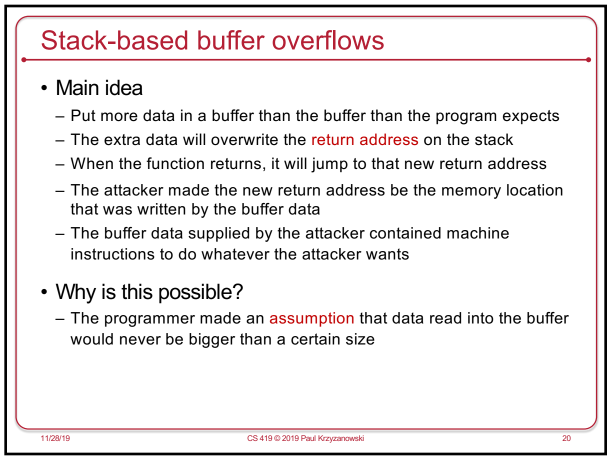
- Test error conditions. Your program should print descriptive messages and exit gracefully
- Test that your instructions are correct. Start with an empty directory and validate that you can follow your own directions
  - If possible, ask a friend to test

18



Code Injection Summary

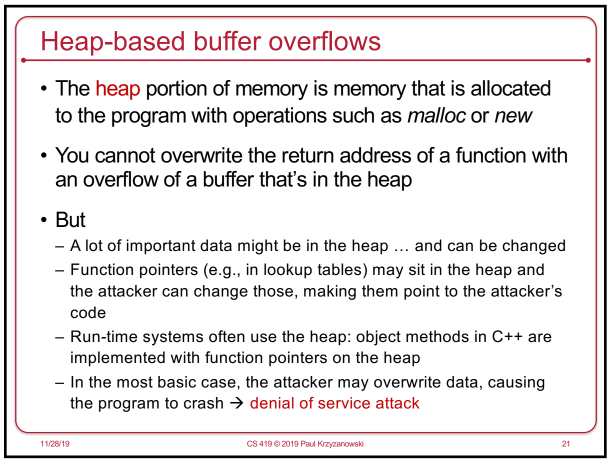
19



Stack-based buffer overflows

- Main idea
  - Put more data in a buffer than the buffer than the program expects
  - The extra data will overwrite the return address on the stack
  - When the function returns, it will jump to that new return address
  - The attacker made the new return address be the memory location that was written by the buffer data
  - The buffer data supplied by the attacker contained machine instructions to do whatever the attacker wants
- Why is this possible?
  - The programmer made an assumption that data read into the buffer would never be bigger than a certain size

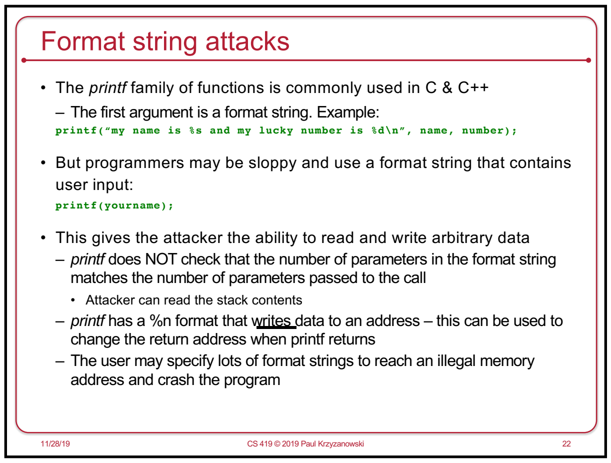
20



Heap-based buffer overflows

- The heap portion of memory is memory that is allocated to the program with operations such as *malloc* or *new*
- You cannot overwrite the return address of a function with an overflow of a buffer that's in the heap
- But
  - A lot of important data might be in the heap … and can be changed
  - Function pointers (e.g., in lookup tables) may sit in the heap and the attacker can change those, making them point to the attacker's code
  - Run-time systems often use the heap: object methods in C++ are implemented with function pointers on the heap
  - In the most basic case, the attacker may overwrite data, causing the program to crash → denial of service attack

21



Format string attacks

- The *printf* family of functions is commonly used in C & C++
  - The first argument is a format string. Example:

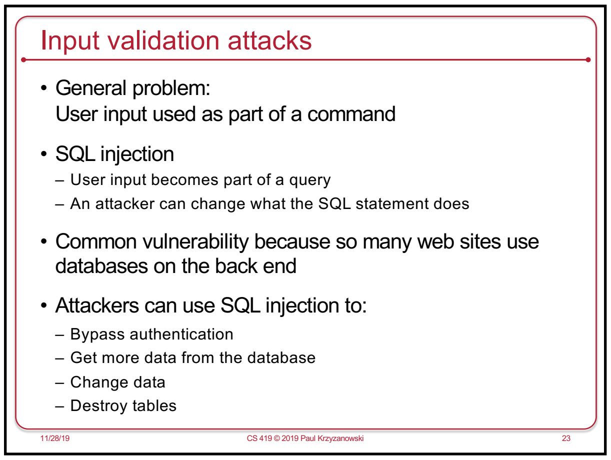
```
printf("my name is %s and my lucky number is %d\n", name, number);
```

- But programmers may be sloppy and use a format string that contains user input:

```
printf(yourname);
```

- This gives the attacker the ability to read and write arbitrary data
  - *printf* does NOT check that the number of parameters in the format string matches the number of parameters passed to the call
    - Attacker can read the stack contents
  - *printf* has a %n format that writes data to an address – this can be used to change the return address when printf returns
  - The user may specify lots of format strings to reach an illegal memory address and crash the program

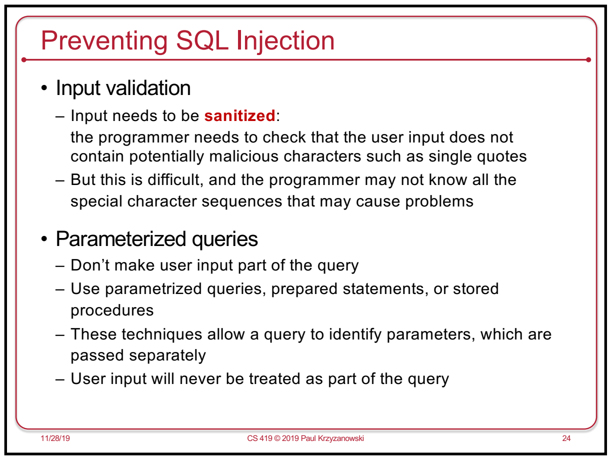
22



Input validation attacks

- General problem:
  User input used as part of a command
- SQL injection
  - User input becomes part of a query
  - An attacker can change what the SQL statement does
- Common vulnerability because so many web sites use databases on the back end
- Attackers can use SQL injection to:
  - Bypass authentication
  - Get more data from the database
  - Change data
  - Destroy tables

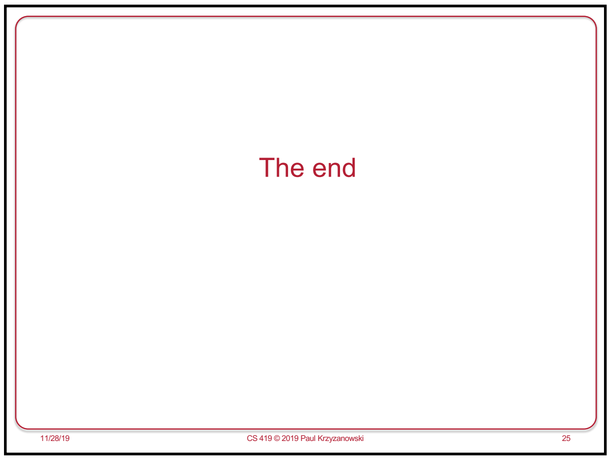
23



Preventing SQL Injection

- Input validation
  - Input needs to be **sanitized**:
    the programmer needs to check that the user input does not contain potentially malicious characters such as single quotes
  - But this is difficult, and the programmer may not know all the special character sequences that may cause problems
- Parameterized queries
  - Don't make user input part of the query
  - Use parametrized queries, prepared statements, or stored procedures
  - These techniques allow a query to identify parameters, which are passed separately
  - User input will never be treated as part of the query

24



The end

25