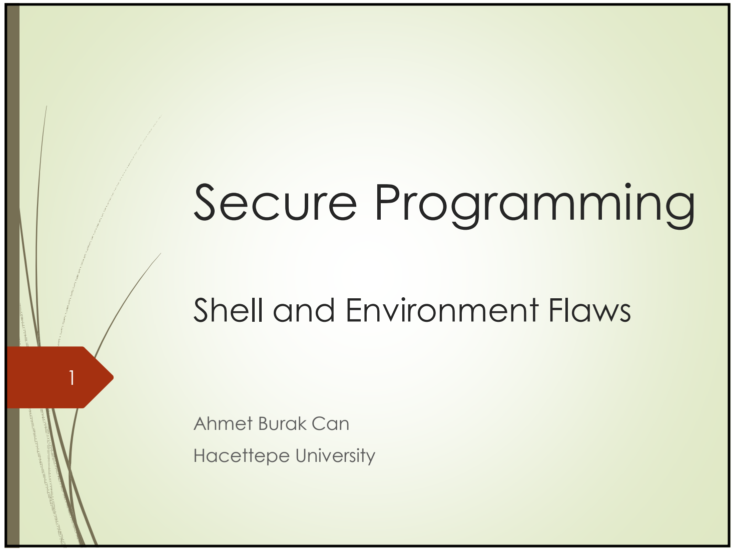
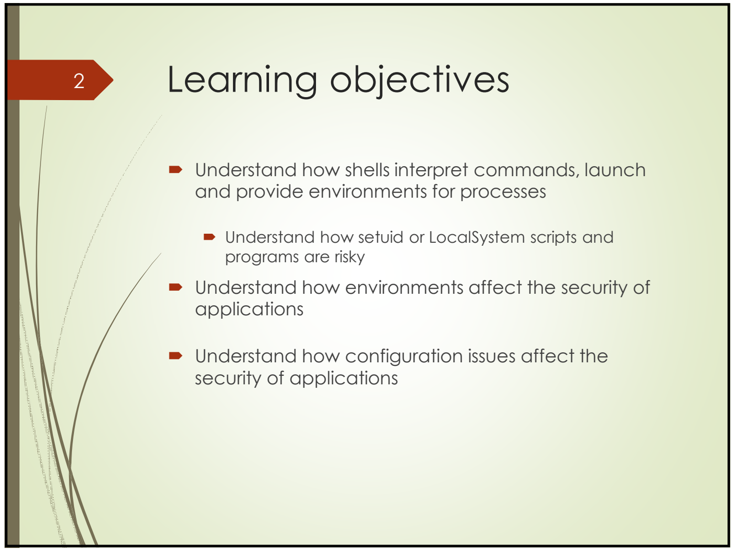


Secure Programming

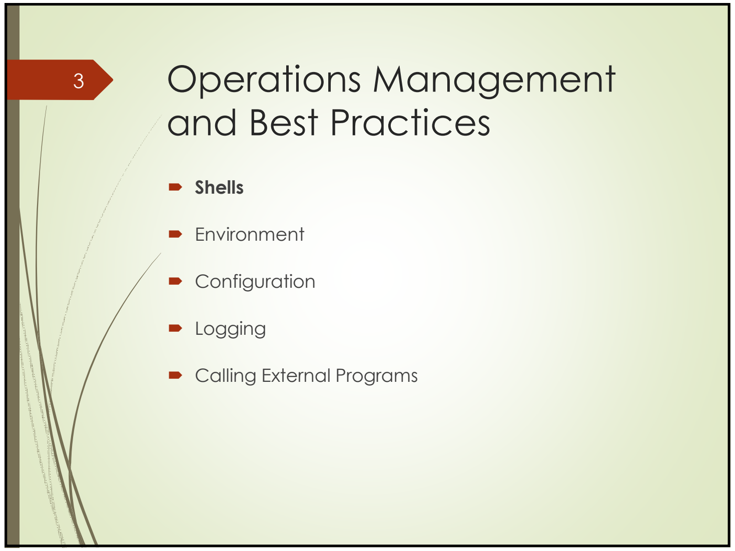
Shell and Environment Flaws

Ahmet Burak Can
Hacettepe University



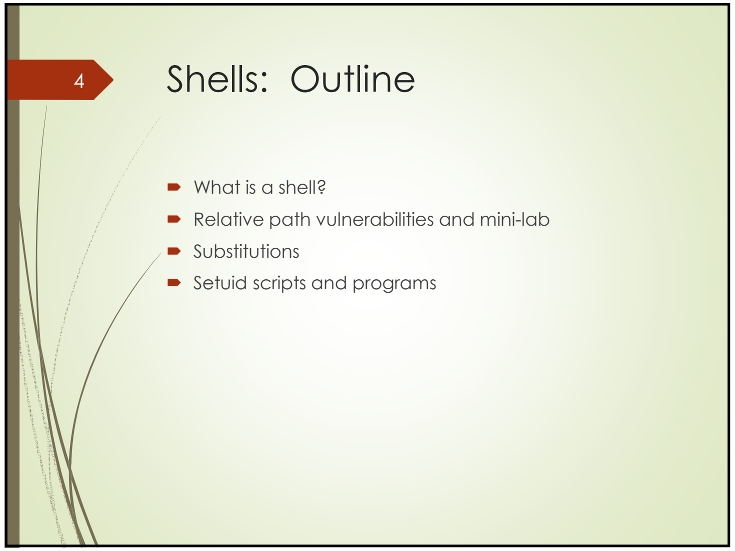
Learning objectives

- Understand how shells interpret commands, launch and provide environments for processes
  - Understand how setuid or LocalSystem scripts and programs are risky
- Understand how environments affect the security of applications
- Understand how configuration issues affect the security of applications



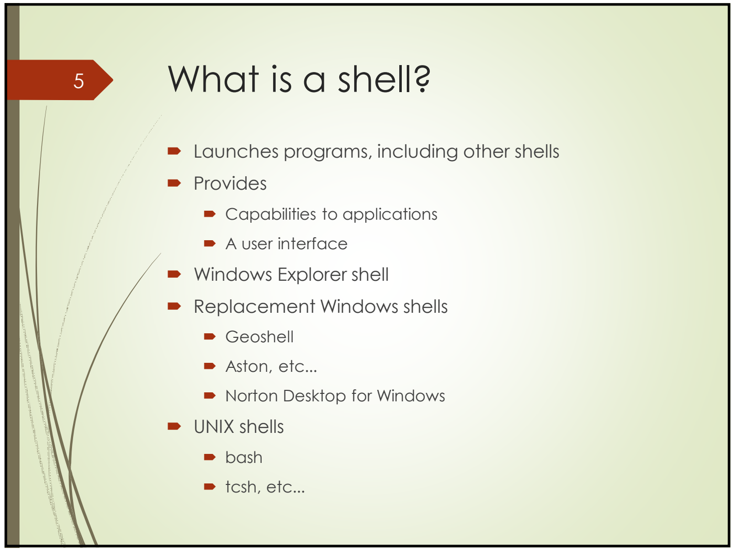
Operations Management and Best Practices

- **Shells**
- Environment
- Configuration
- Logging
- Calling External Programs



Shells: Outline

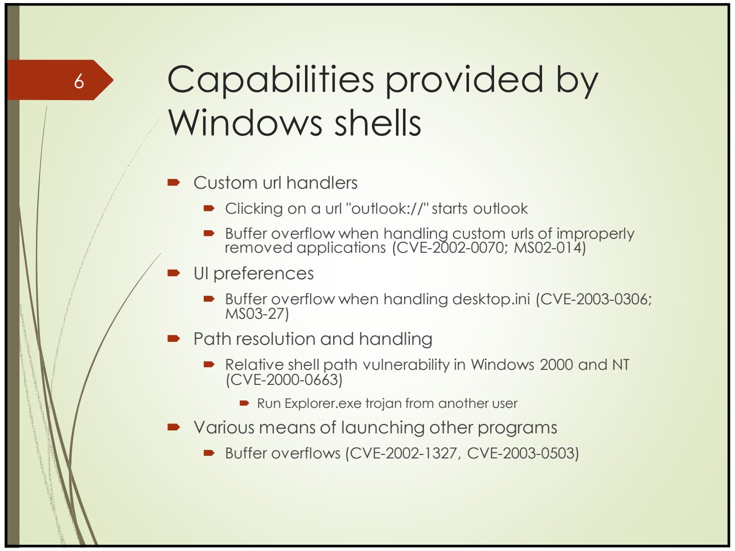
- What is a shell?
- Relative path vulnerabilities and mini-lab
- Substitutions
- Setuid scripts and programs



5

What is a shell?

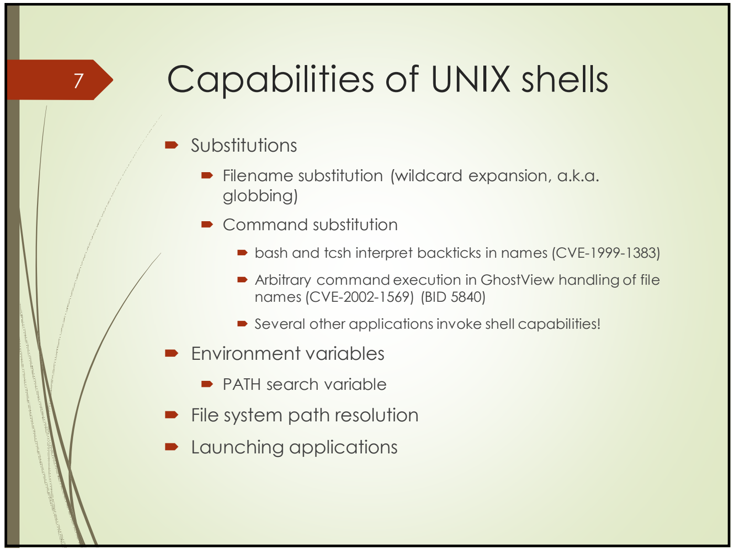
- Launches programs, including other shells
- Provides
  - Capabilities to applications
  - A user interface
- Windows Explorer shell
- Replacement Windows shells
  - Geoshell
  - Aston, etc...
  - Norton Desktop for Windows
- UNIX shells
  - bash
  - tcsh, etc...



6

Capabilities provided by Windows shells

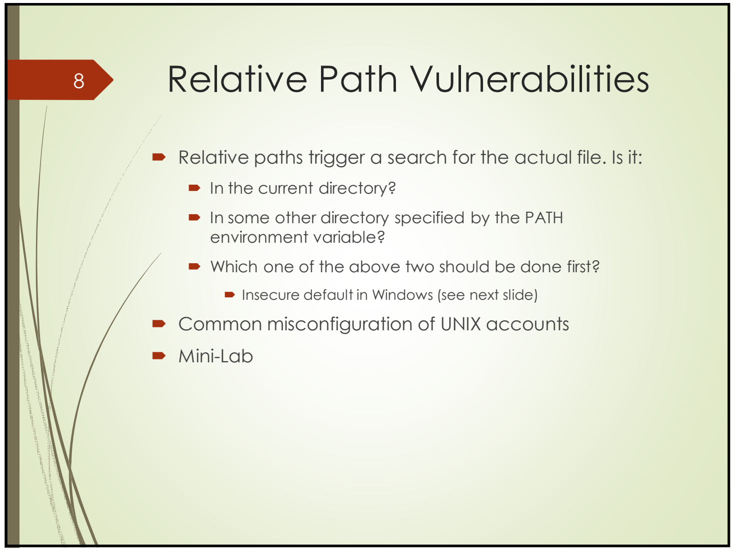
- Custom url handlers
  - Clicking on a url "outlook://" starts outlook
  - Buffer overflow when handling custom urls of improperly removed applications (CVE-2002-0070; MS02-014)
- UI preferences
  - Buffer overflow when handling desktop.ini (CVE-2003-0306; MS03-27)
- Path resolution and handling
  - Relative shell path vulnerability in Windows 2000 and NT (CVE-2000-0663)
    - Run Explorer.exe trojan from another user
- Various means of launching other programs
  - Buffer overflows (CVE-2002-1327, CVE-2003-0503)



7

Capabilities of UNIX shells

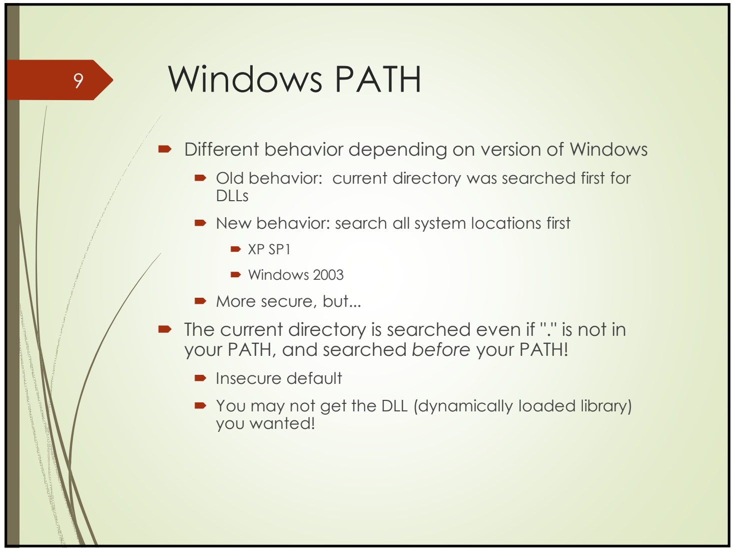
- Substitutions
  - Filename substitution (wildcard expansion, a.k.a. globbing)
  - Command substitution
    - bash and tcsh interpret backticks in names (CVE-1999-1383)
    - Arbitrary command execution in GhostView handling of file names (CVE-2002-1569) (BID 5840)
    - Several other applications invoke shell capabilities!
- Environment variables
  - PATH search variable
- File system path resolution
- Launching applications



8

Relative Path Vulnerabilities

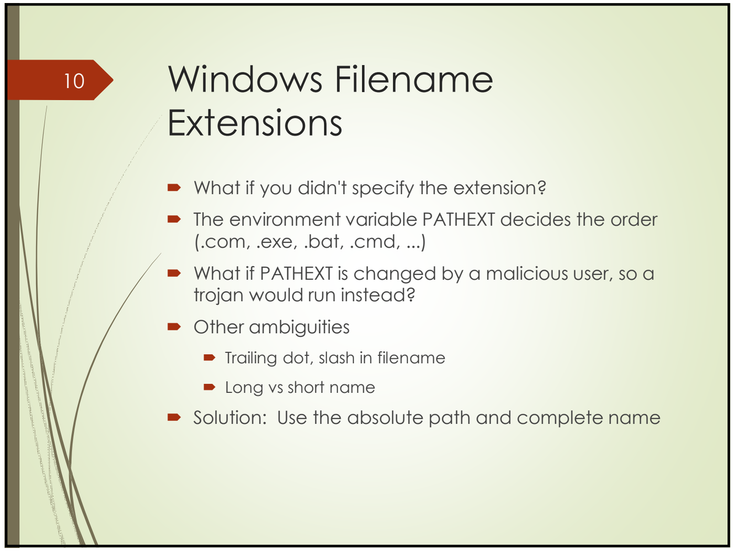
- Relative paths trigger a search for the actual file. Is it:
  - In the current directory?
  - In some other directory specified by the PATH environment variable?
  - Which one of the above two should be done first?
    - Insecure default in Windows (see next slide)
- Common misconfiguration of UNIX accounts
- Mini-Lab



9

Windows PATH

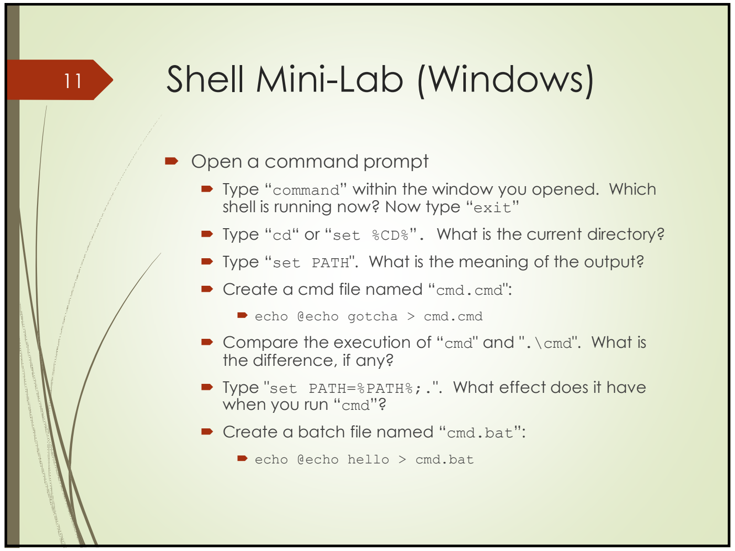
- Different behavior depending on version of Windows
  - Old behavior: current directory was searched first for DLLs
  - New behavior: search all system locations first
    - XP SP1
    - Windows 2003
  - More secure, but...
- The current directory is searched even if "." is not in your PATH, and searched *before* your PATH!
  - Insecure default
  - You may not get the DLL (dynamically loaded library) you wanted!



10

Windows Filename Extensions

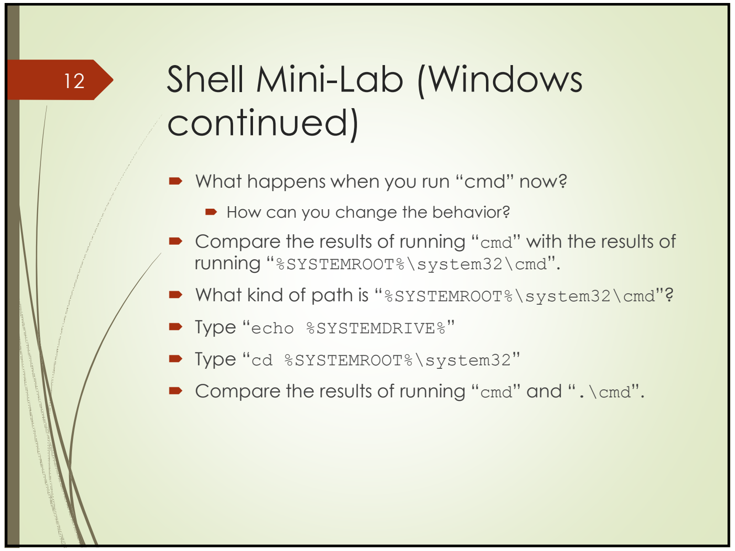
- What if you didn't specify the extension?
- The environment variable PATHEXT decides the order (.com, .exe, .bat, .cmd, ...)
- What if PATHEXT is changed by a malicious user, so a trojan would run instead?
- Other ambiguities
  - Trailing dot, slash in filename
  - Long vs short name
- Solution: Use the absolute path and complete name



11

Shell Mini-Lab (Windows)

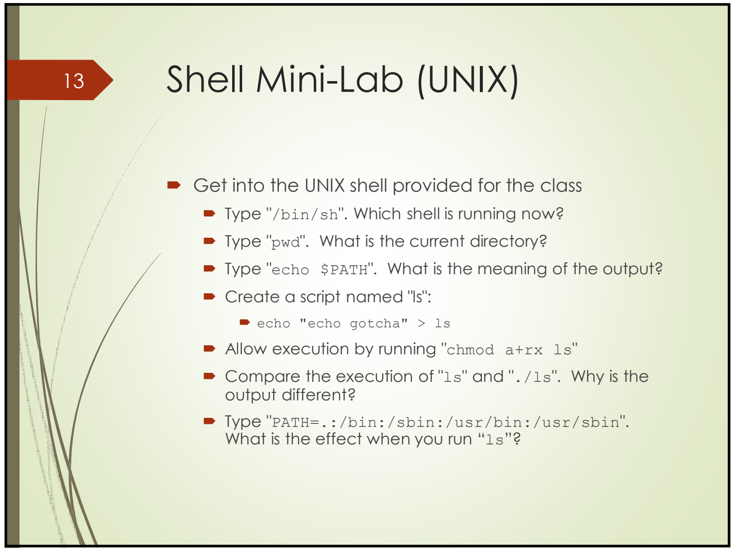
- Open a command prompt
  - Type “`command`” within the window you opened. Which shell is running now? Now type “`exit`”
  - Type “`cd`“ or “`set %CD%`”. What is the current directory?
  - Type “`set PATH`". What is the meaning of the output?
  - Create a cmd file named “`cmd.cmd`":
    - `echo @echo gotcha > cmd.cmd`
  - Compare the execution of “`cmd`" and "`.\cmd`". What is the difference, if any?
  - Type "`set PATH=%PATH%;.`". What effect does it have when you run “`cmd`”?
  - Create a batch file named “`cmd.bat`”:
    - `echo @echo hello > cmd.bat`



12

Shell Mini-Lab (Windows continued)

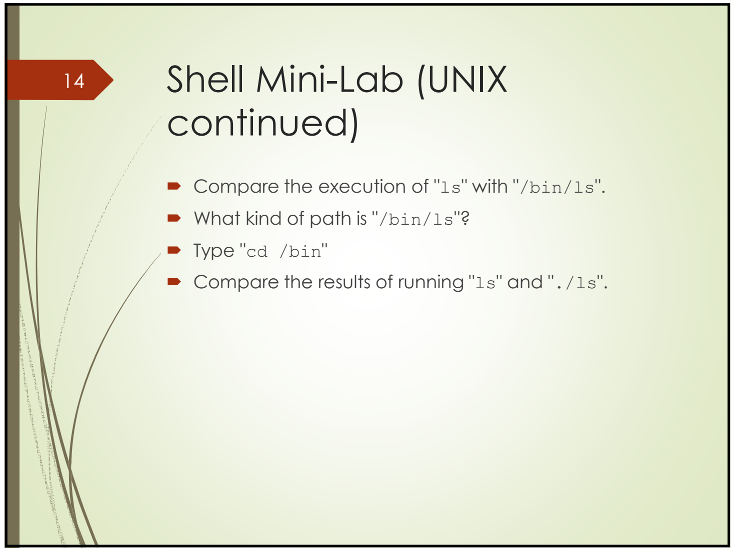
- What happens when you run “cmd” now?
  - How can you change the behavior?
- Compare the results of running “`cmd`” with the results of running “`%SYSTEMROOT%\system32\cmd`”.
- What kind of path is “`%SYSTEMROOT%\system32\cmd`”?
- Type “`echo %SYSTEMDRIVE%`”
- Type “`cd %SYSTEMROOT%\system32`”
- Compare the results of running “`cmd`” and “`.\cmd`”.



13

Shell Mini-Lab (UNIX)

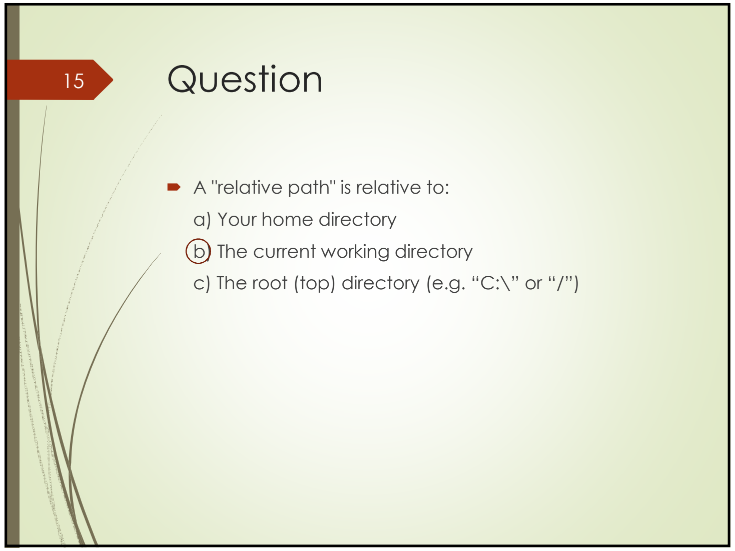
- Get into the UNIX shell provided for the class
  - Type "`/bin/sh`". Which shell is running now?
  - Type "`pwd`". What is the current directory?
  - Type "`echo $PATH`". What is the meaning of the output?
  - Create a script named "ls":
    - `echo "echo gotcha" > ls`
  - Allow execution by running "`chmod a+rx ls`"
  - Compare the execution of "`ls`" and "`./ls`". Why is the output different?
  - Type "`PATH=.:/bin:/sbin:/usr/bin:/usr/sbin`". What is the effect when you run "`ls`"?



14

Shell Mini-Lab (UNIX continued)

- Compare the execution of "`ls`" with "`/bin/ls`".
- What kind of path is "`/bin/ls`"?
- Type "`cd /bin`"
- Compare the results of running "`ls`" and "`./ls`".



15

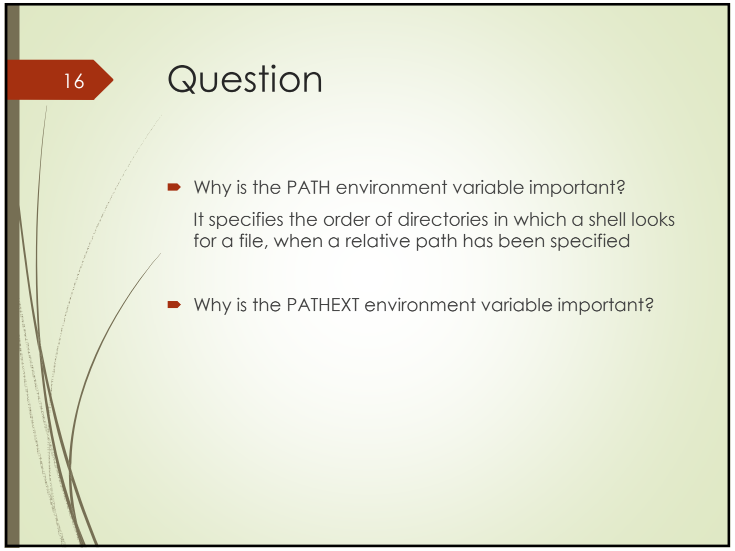
Question

- A "relative path" is relative to:

  a) Your home directory

  [✓ b)] The current working directory

  c) The root (top) directory (e.g. "C:\" or "/")



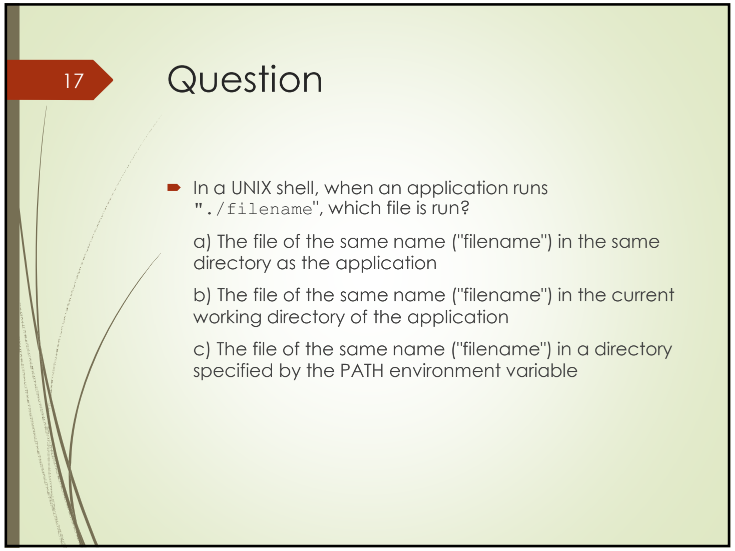
16

Question

- Why is the PATH environment variable important?

  It specifies the order of directories in which a shell looks for a file, when a relative path has been specified

- Why is the PATHEXT environment variable important?



17

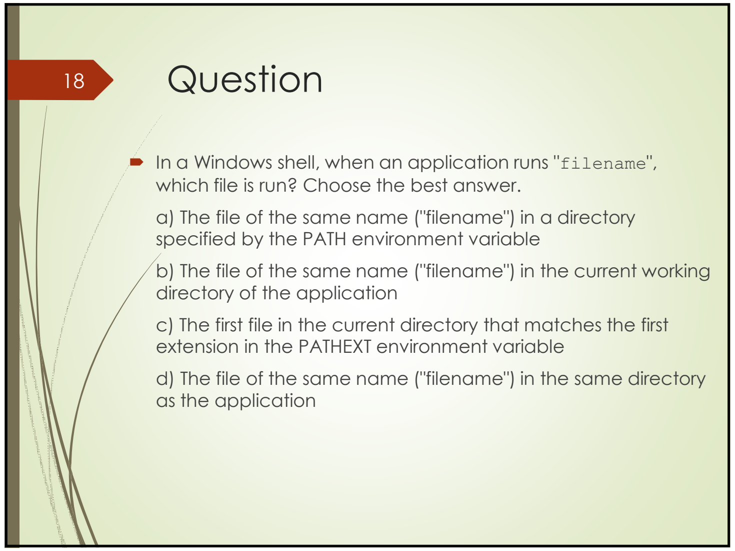
Question

- In a UNIX shell, when an application runs "`./filename`", which file is run?

  a) The file of the same name ("filename") in the same directory as the application

  b) The file of the same name ("filename") in the current working directory of the application

  c) The file of the same name ("filename") in a directory specified by the PATH environment variable



18

Question

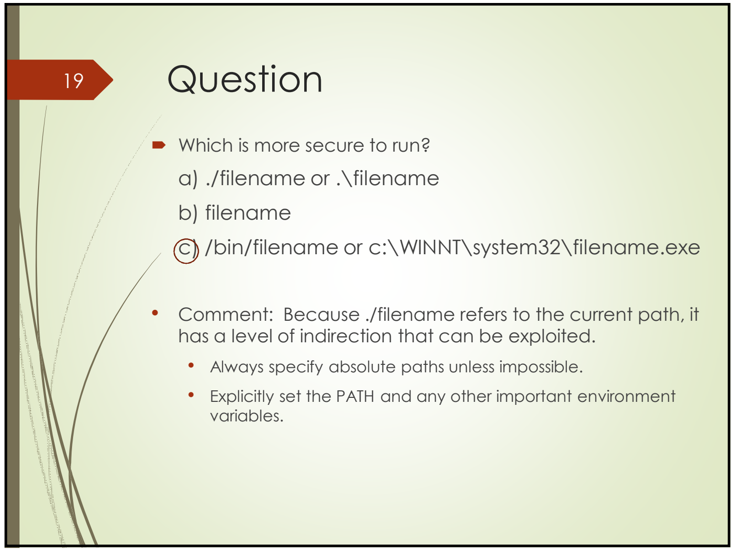
- In a Windows shell, when an application runs "`filename`", which file is run? Choose the best answer.

  a) The file of the same name ("filename") in a directory specified by the PATH environment variable

  b) The file of the same name ("filename") in the current working directory of the application

  c) The first file in the current directory that matches the first extension in the PATHEXT environment variable

  d) The file of the same name ("filename") in the same directory as the application



19

Question

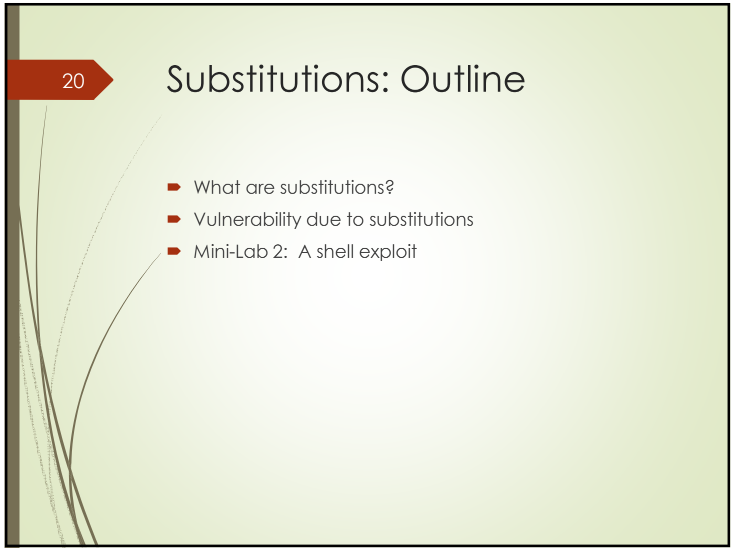
- Which is more secure to run?

  a) ./filename or .\filename

  b) filename

  [✓ c)] /bin/filename or c:\WINNT\system32\filename.exe

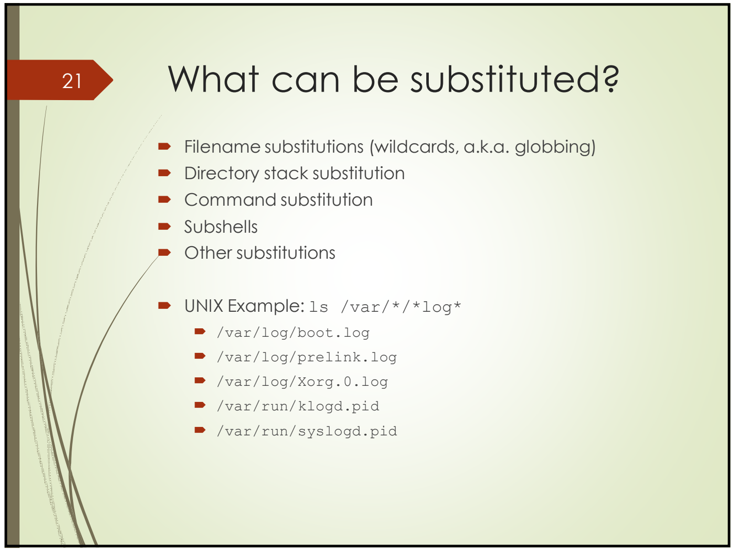
- Comment: Because ./filename refers to the current path, it has a level of indirection that can be exploited.
  - Always specify absolute paths unless impossible.
  - Explicitly set the PATH and any other important environment variables.



20

Substitutions: Outline

- What are substitutions?
- Vulnerability due to substitutions
- Mini-Lab 2: A shell exploit

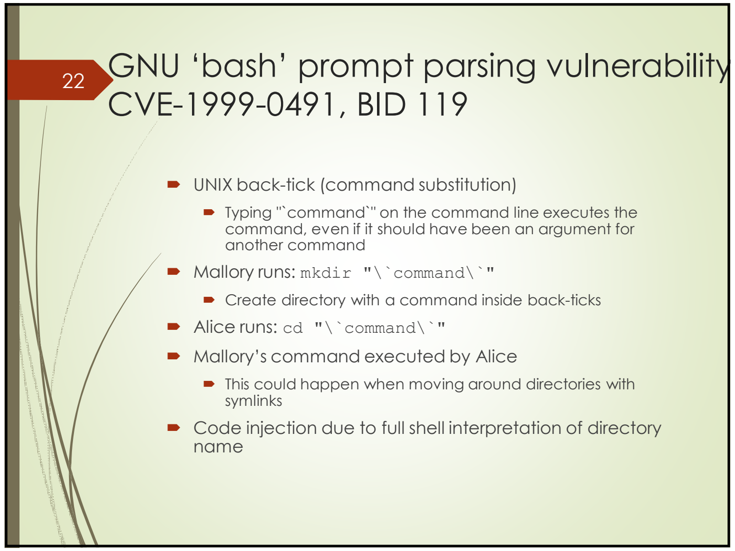


21

What can be substituted?

- Filename substitutions (wildcards, a.k.a. globbing)
- Directory stack substitution
- Command substitution
- Subshells
- Other substitutions

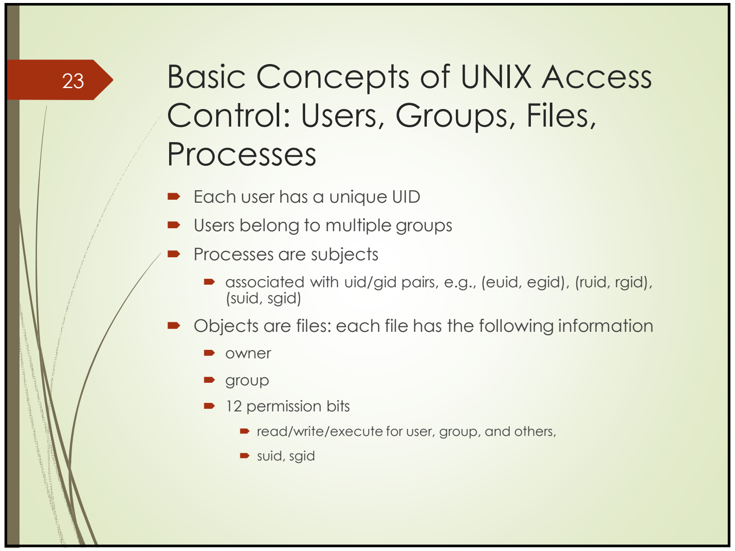
- UNIX Example: `ls /var/*/*log*`
  - `/var/log/boot.log`
  - `/var/log/prelink.log`
  - `/var/log/Xorg.0.log`
  - `/var/run/klogd.pid`
  - `/var/run/syslogd.pid`



22

GNU ‘bash’ prompt parsing vulnerability CVE-1999-0491, BID 119

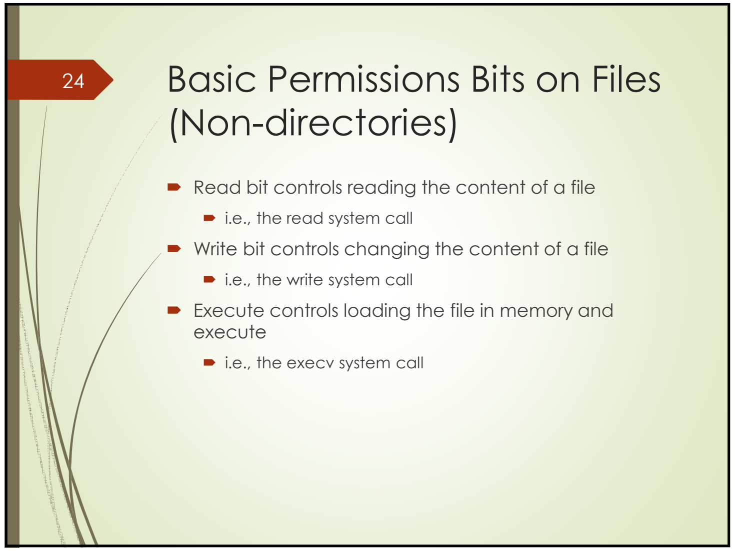
- UNIX back-tick (command substitution)
  - Typing "`command`" on the command line executes the command, even if it should have been an argument for another command
- Mallory runs: `mkdir "\`command\`"`
  - Create directory with a command inside back-ticks
- Alice runs: `cd "\`command\`"`
- Mallory’s command executed by Alice
  - This could happen when moving around directories with symlinks
- Code injection due to full shell interpretation of directory name



23

Basic Concepts of UNIX Access Control: Users, Groups, Files, Processes

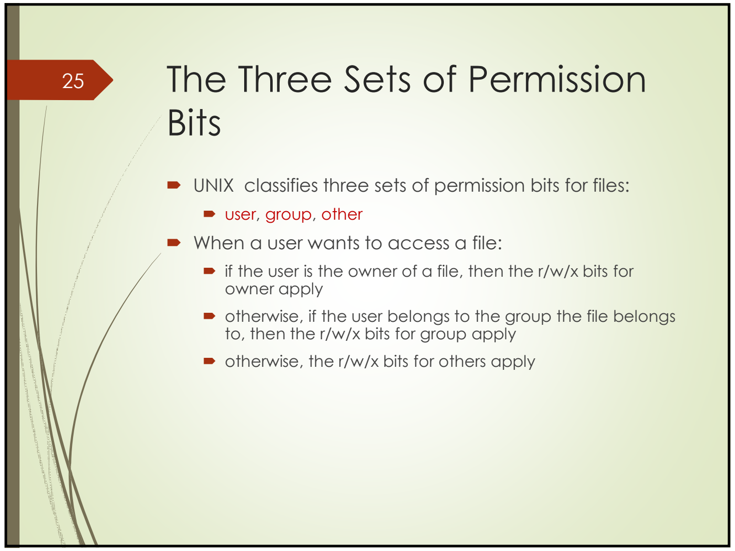
- Each user has a unique UID
- Users belong to multiple groups
- Processes are subjects
  - associated with uid/gid pairs, e.g., (euid, egid), (ruid, rgid), (suid, sgid)
- Objects are files: each file has the following information
  - owner
  - group
  - 12 permission bits
    - read/write/execute for user, group, and others,
    - suid, sgid



24

Basic Permissions Bits on Files (Non-directories)

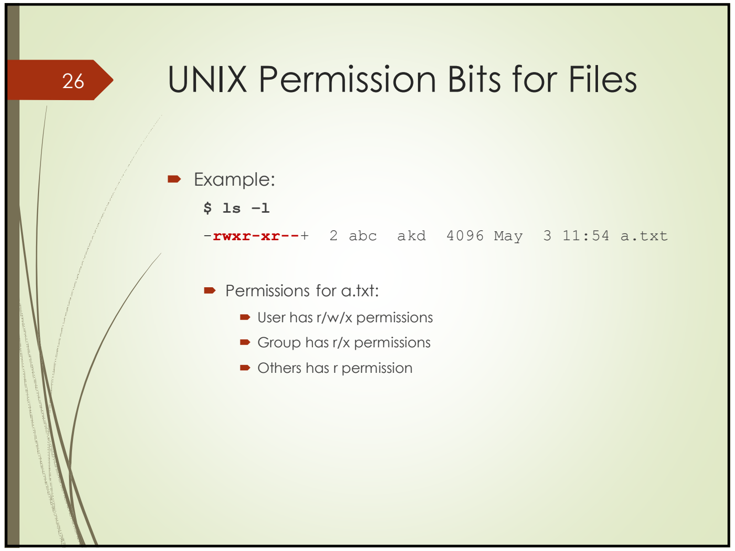
- Read bit controls reading the content of a file
  - i.e., the read system call
- Write bit controls changing the content of a file
  - i.e., the write system call
- Execute controls loading the file in memory and execute
  - i.e., the execv system call



25

The Three Sets of Permission Bits

- UNIX classifies three sets of permission bits for files:
  - user, group, other
- When a user wants to access a file:
  - if the user is the owner of a file, then the r/w/x bits for owner apply
  - otherwise, if the user belongs to the group the file belongs to, then the r/w/x bits for group apply
  - otherwise, the r/w/x bits for others apply



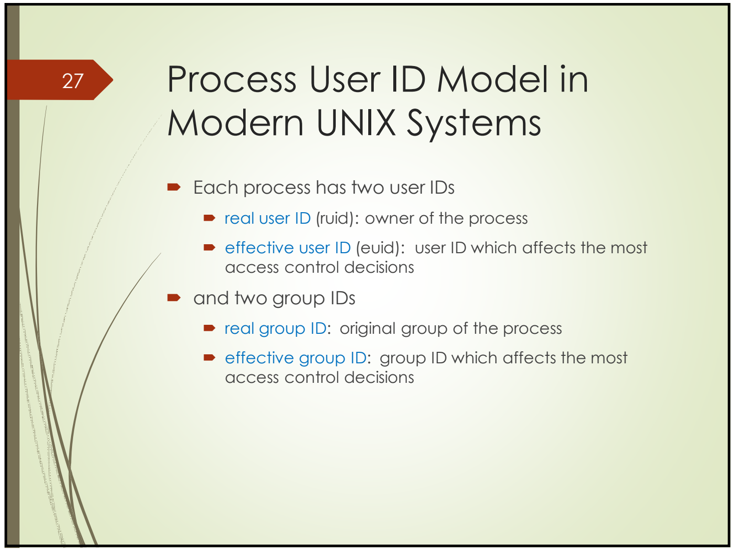
26

UNIX Permission Bits for Files

- Example:

```
$ ls -l
-rwxr-xr--+  2 abc  akd  4096 May  3 11:54 a.txt
```

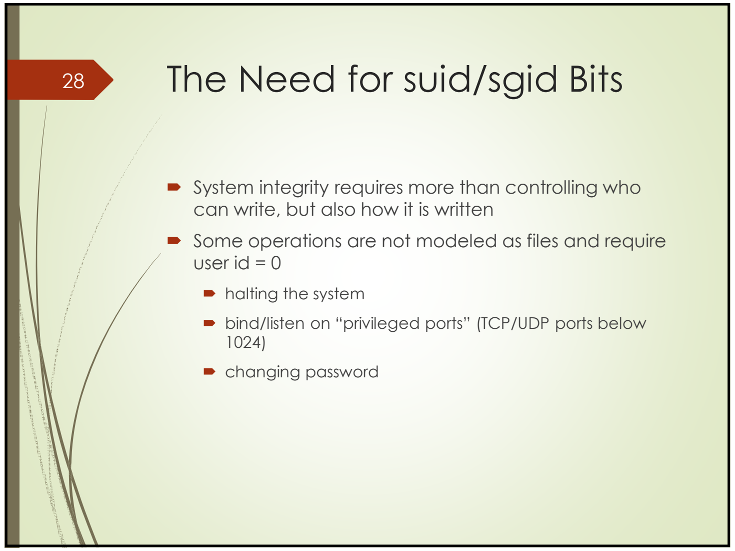
  - Permissions for a.txt:
    - User has r/w/x permissions
    - Group has r/x permissions
    - Others has r permission



27

Process User ID Model in Modern UNIX Systems

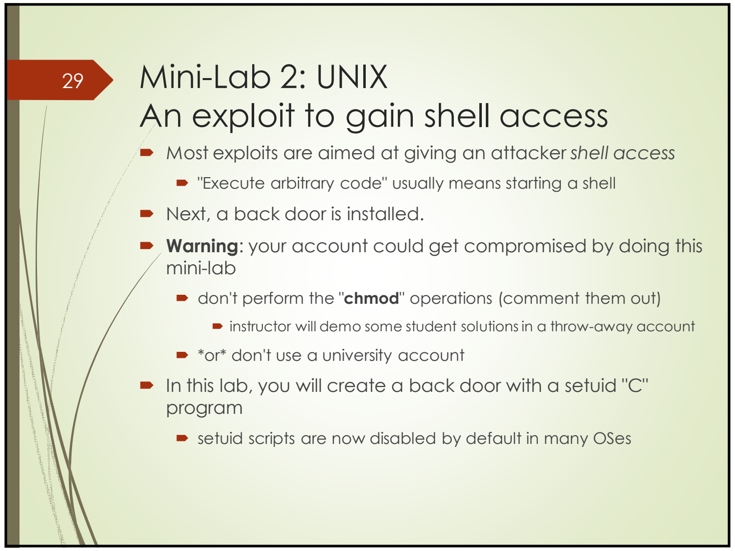
- Each process has two user IDs
  - real user ID (ruid): owner of the process
  - effective user ID (euid): user ID which affects the most access control decisions
- and two group IDs
  - real group ID: original group of the process
  - effective group ID: group ID which affects the most access control decisions



28

The Need for suid/sgid Bits

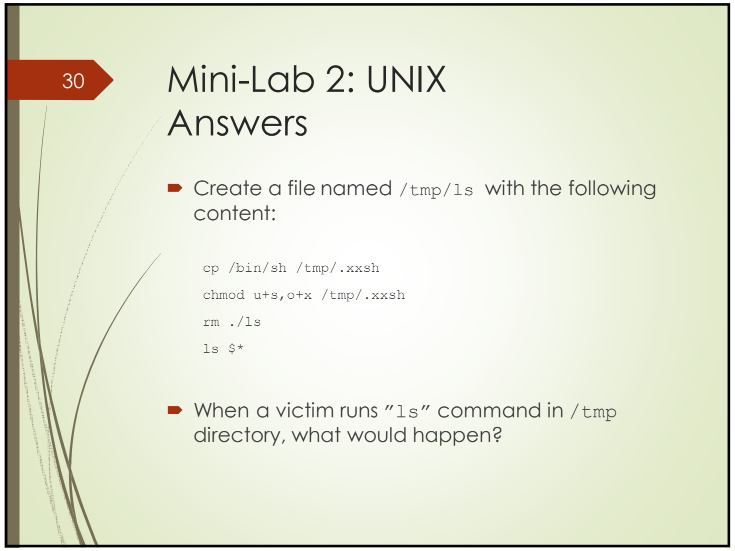
- System integrity requires more than controlling who can write, but also how it is written
- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on “privileged ports” (TCP/UDP ports below 1024)
  - changing password



29

Mini-Lab 2: UNIX An exploit to gain shell access

- Most exploits are aimed at giving an attacker *shell access*
  - "Execute arbitrary code" usually means starting a shell
- Next, a back door is installed.
- **Warning**: your account could get compromised by doing this mini-lab
  - don't perform the "**chmod**" operations (comment them out)
    - instructor will demo some student solutions in a throw-away account
  - *or* don't use a university account
- In this lab, you will create a back door with a setuid "C" program
  - setuid scripts are now disabled by default in many OSes



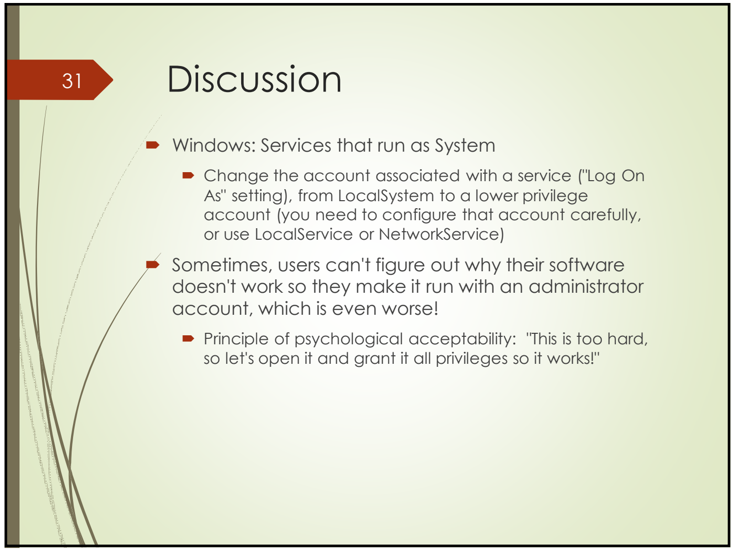
30

Mini-Lab 2: UNIX Answers

- Create a file named `/tmp/ls` with the following content:

```
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls
ls $*
```

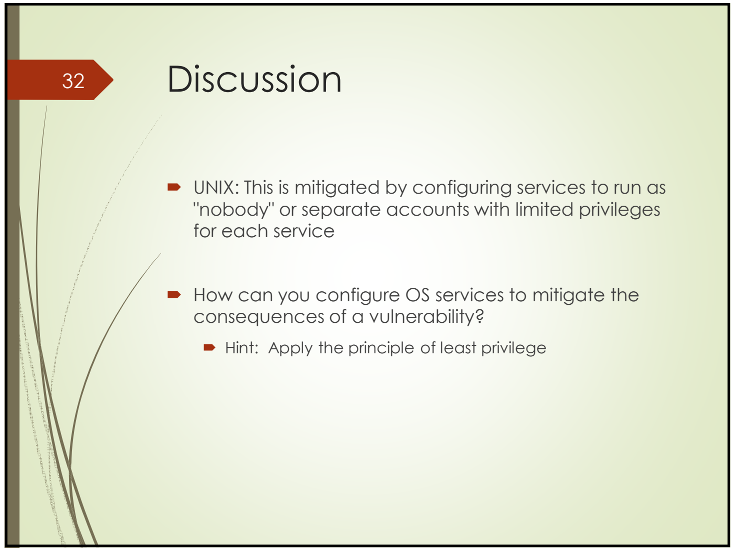
- When a victim runs "`ls`" command in `/tmp` directory, what would happen?



31

Discussion

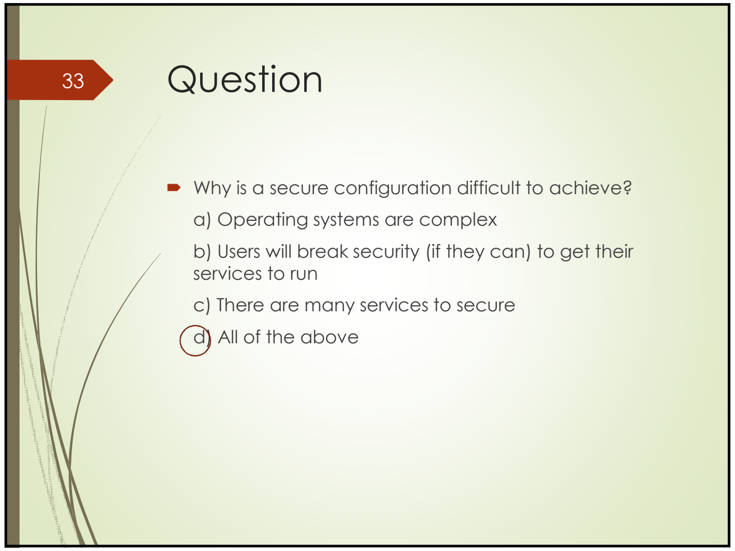
- Windows: Services that run as System
  - Change the account associated with a service ("Log On As" setting), from LocalSystem to a lower privilege account (you need to configure that account carefully, or use LocalService or NetworkService)
- Sometimes, users can't figure out why their software doesn't work so they make it run with an administrator account, which is even worse!
  - Principle of psychological acceptability: "This is too hard, so let's open it and grant it all privileges so it works!"



32

Discussion

- UNIX: This is mitigated by configuring services to run as "nobody" or separate accounts with limited privileges for each service
- How can you configure OS services to mitigate the consequences of a vulnerability?
  - Hint: Apply the principle of least privilege



33

Question

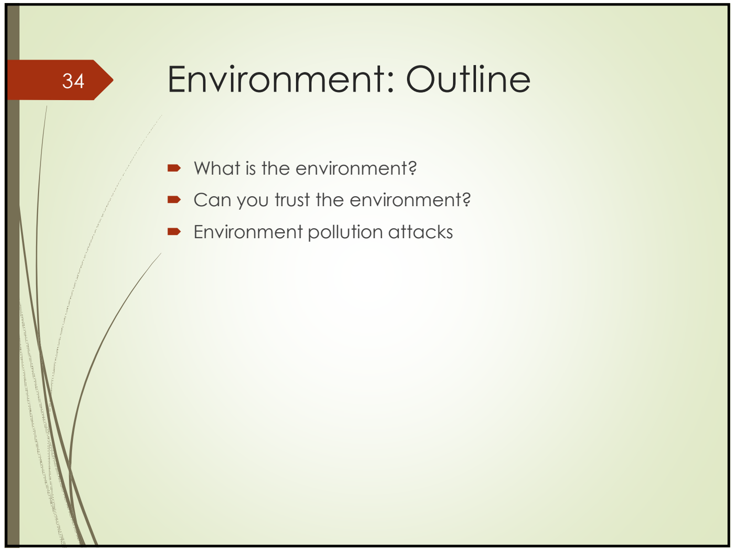
- Why is a secure configuration difficult to achieve?

  a) Operating systems are complex

  b) Users will break security (if they can) to get their services to run

  c) There are many services to secure

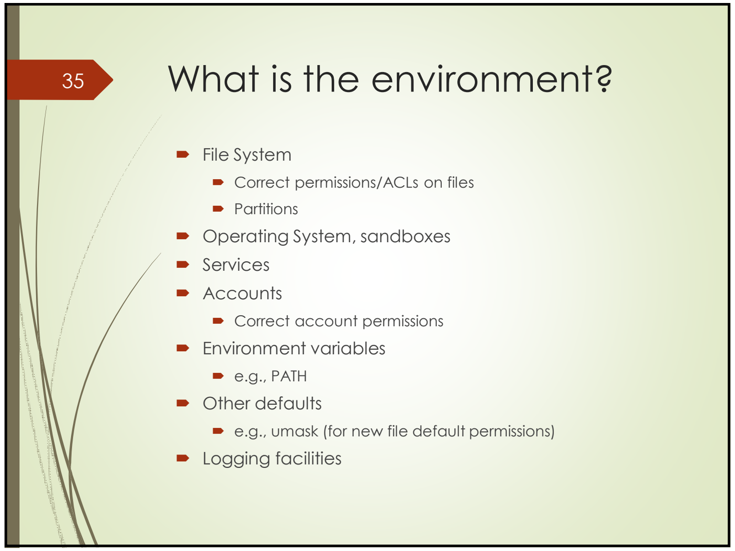
  [✓ d) All of the above]



34

Environment: Outline

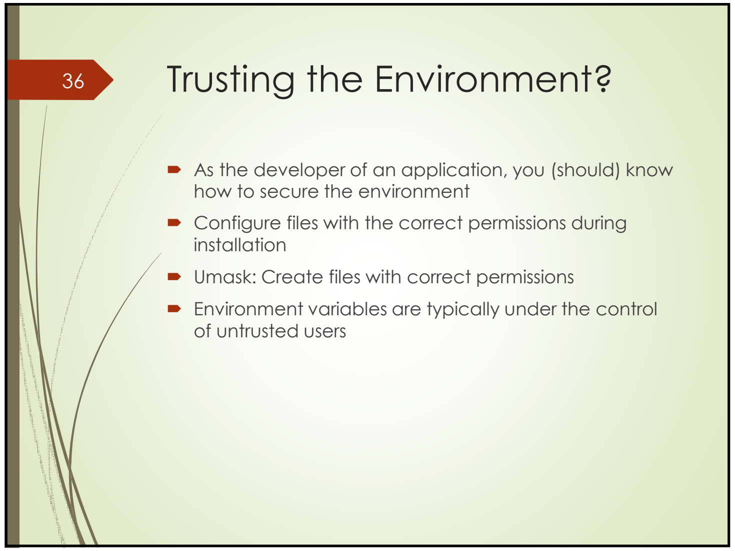
- What is the environment?
- Can you trust the environment?
- Environment pollution attacks



35

What is the environment?

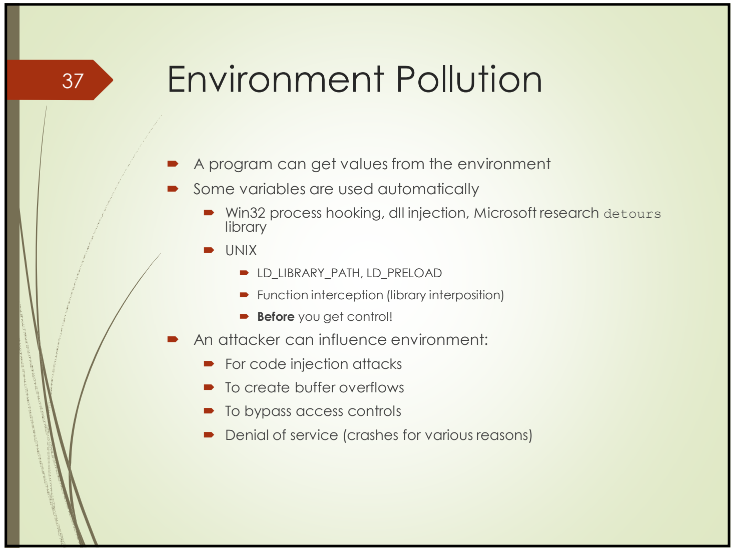
- File System
  - Correct permissions/ACLs on files
  - Partitions
- Operating System, sandboxes
- Services
- Accounts
  - Correct account permissions
- Environment variables
  - e.g., PATH
- Other defaults
  - e.g., umask (for new file default permissions)
- Logging facilities



36

Trusting the Environment?

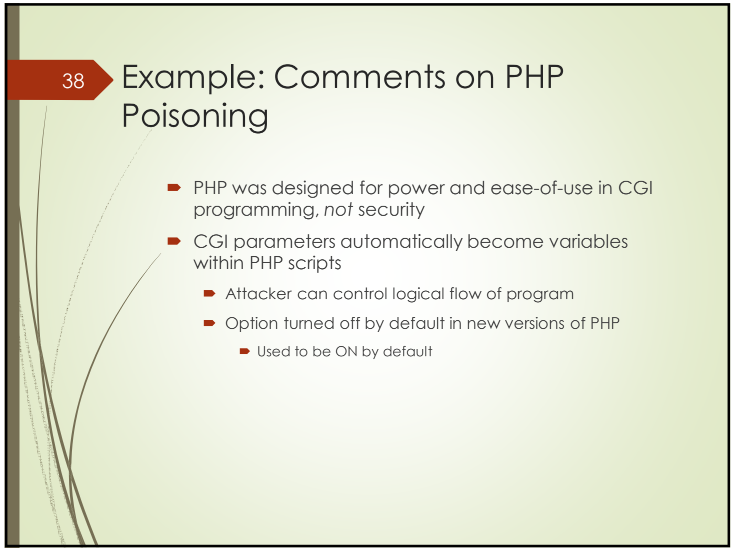
- As the developer of an application, you (should) know how to secure the environment
- Configure files with the correct permissions during installation
- Umask: Create files with correct permissions
- Environment variables are typically under the control of untrusted users



37

Environment Pollution

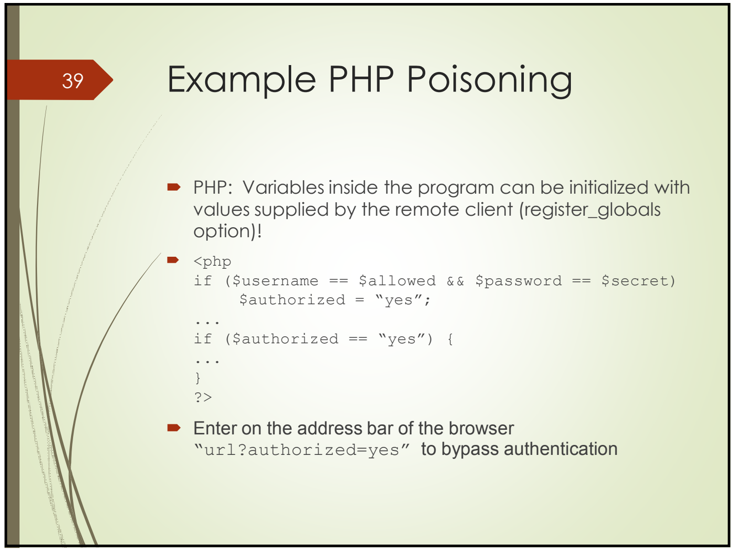
- A program can get values from the environment
- Some variables are used automatically
  - Win32 process hooking, dll injection, Microsoft research `detours` library
  - UNIX
    - LD_LIBRARY_PATH, LD_PRELOAD
    - Function interception (library interposition)
    - **Before** you get control!
- An attacker can influence environment:
  - For code injection attacks
  - To create buffer overflows
  - To bypass access controls
  - Denial of service (crashes for various reasons)



38

Example: Comments on PHP Poisoning

- PHP was designed for power and ease-of-use in CGI programming, *not* security
- CGI parameters automatically become variables within PHP scripts
  - Attacker can control logical flow of program
  - Option turned off by default in new versions of PHP
    - Used to be ON by default

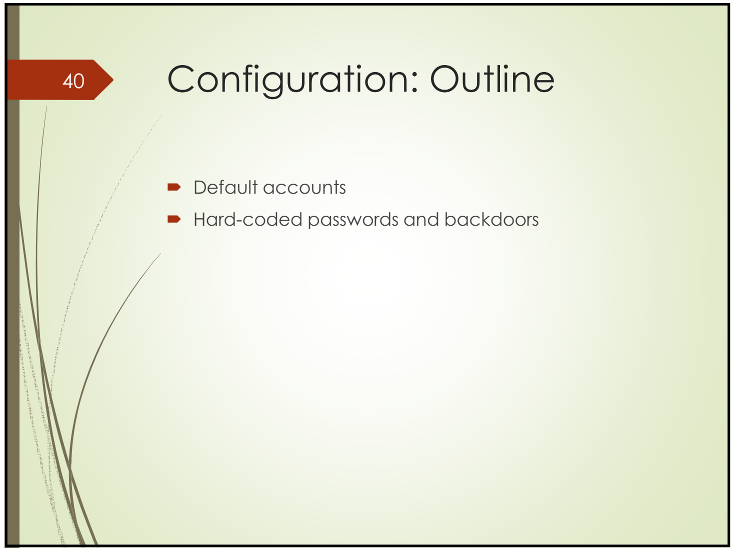


39

Example PHP Poisoning

- PHP: Variables inside the program can be initialized with values supplied by the remote client (register_globals option)!

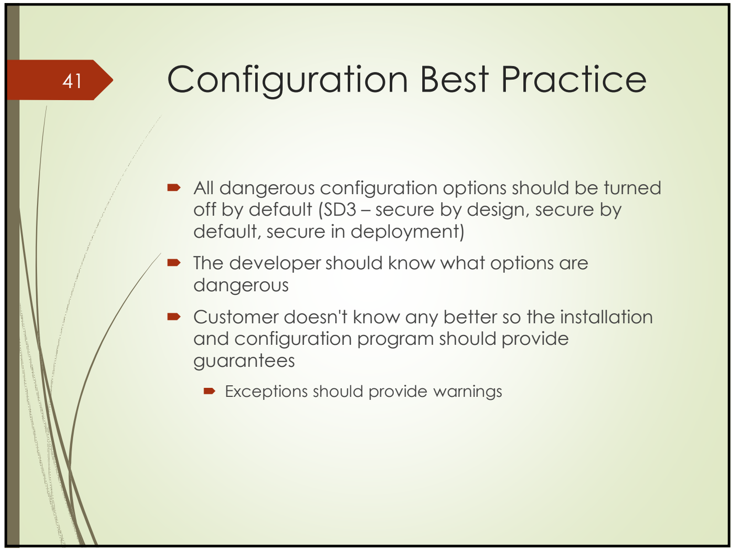
-
  ```
  <php
  if ($username == $allowed && $password == $secret)
       $authorized = "yes";
  ...
  if ($authorized == "yes") {
  ...
  }
  ?>
  ```
- Enter on the address bar of the browser `"url?authorized=yes"` to bypass authentication



40

Configuration: Outline

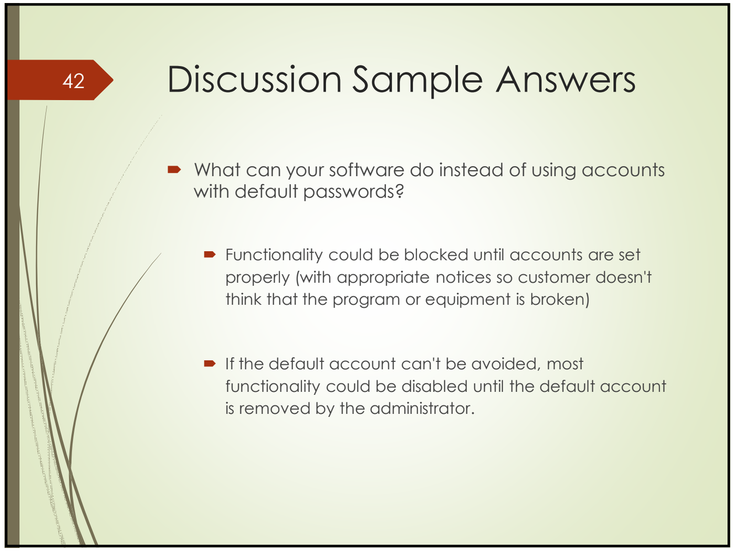
- Default accounts
- Hard-coded passwords and backdoors



41

Configuration Best Practice

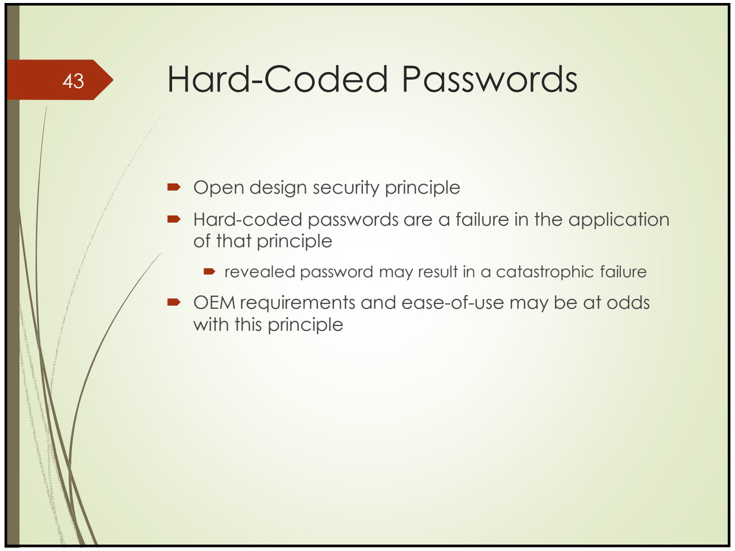
- All dangerous configuration options should be turned off by default (SD3 – secure by design, secure by default, secure in deployment)
- The developer should know what options are dangerous
- Customer doesn't know any better so the installation and configuration program should provide guarantees
  - Exceptions should provide warnings



42

Discussion Sample Answers

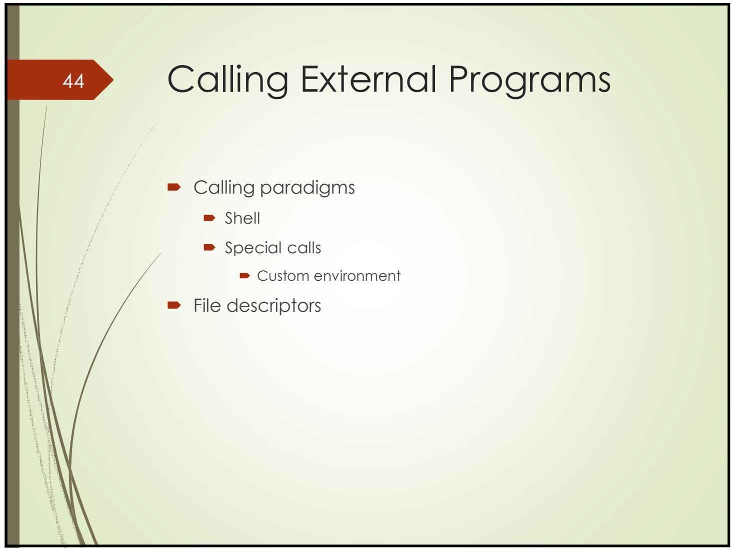
- What can your software do instead of using accounts with default passwords?
  - Functionality could be blocked until accounts are set properly (with appropriate notices so customer doesn't think that the program or equipment is broken)
  - If the default account can't be avoided, most functionality could be disabled until the default account is removed by the administrator.



43

Hard-Coded Passwords

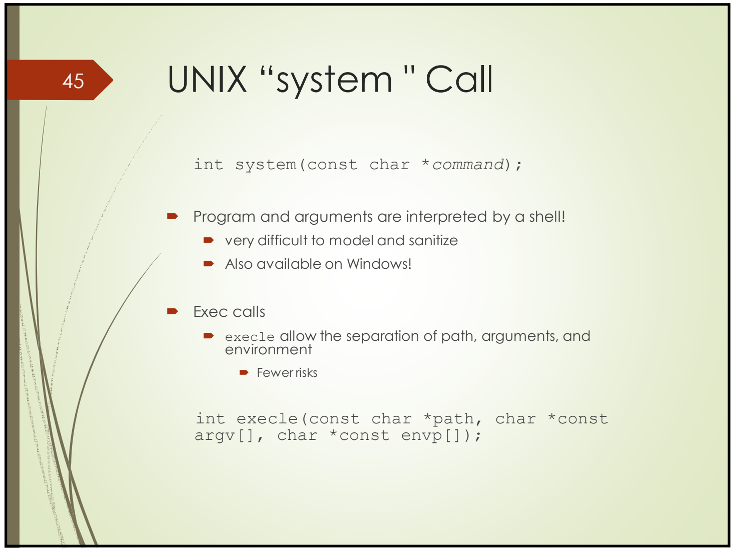
- Open design security principle
- Hard-coded passwords are a failure in the application of that principle
  - revealed password may result in a catastrophic failure
- OEM requirements and ease-of-use may be at odds with this principle



44

Calling External Programs

- Calling paradigms
  - Shell
  - Special calls
    - Custom environment
- File descriptors



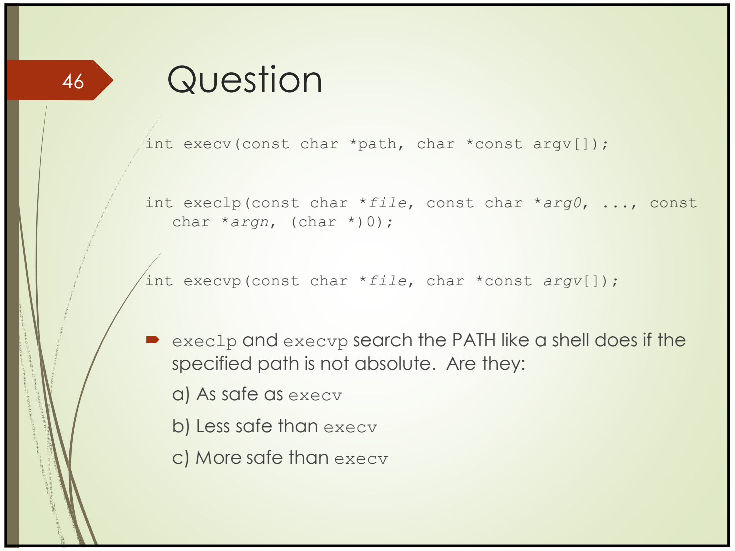
45

UNIX “system " Call

```
int system(const char *command);
```

- Program and arguments are interpreted by a shell!
  - very difficult to model and sanitize
  - Also available on Windows!
- Exec calls
  - `execle` allow the separation of path, arguments, and environment
    - Fewer risks

```
int execle(const char *path, char *const
argv[], char *const envp[]);
```



46

Question

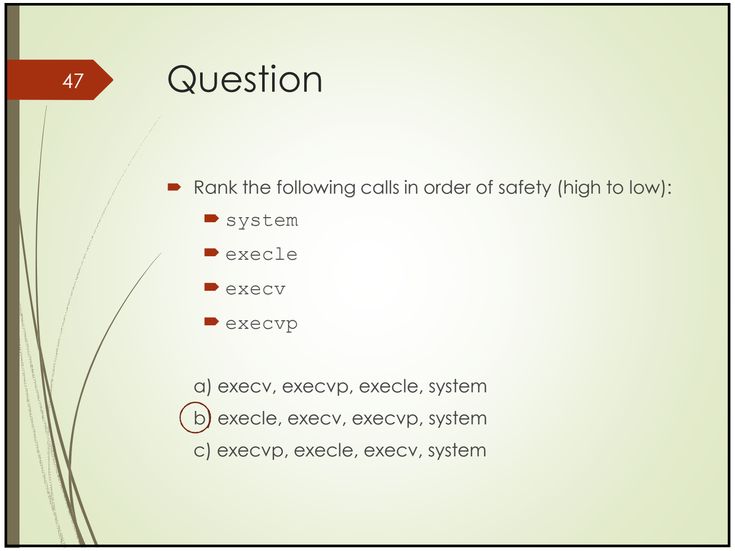
```
int execv(const char *path, char *const argv[]);

int execlp(const char *file, const char *arg0, ..., const
   char *argn, (char *)0);

int execvp(const char *file, char *const argv[]);
```

- `execlp` and `execvp` search the PATH like a shell does if the specified path is not absolute. Are they:

  a) As safe as `execv`

  b) Less safe than `execv`

  c) More safe than `execv`



47

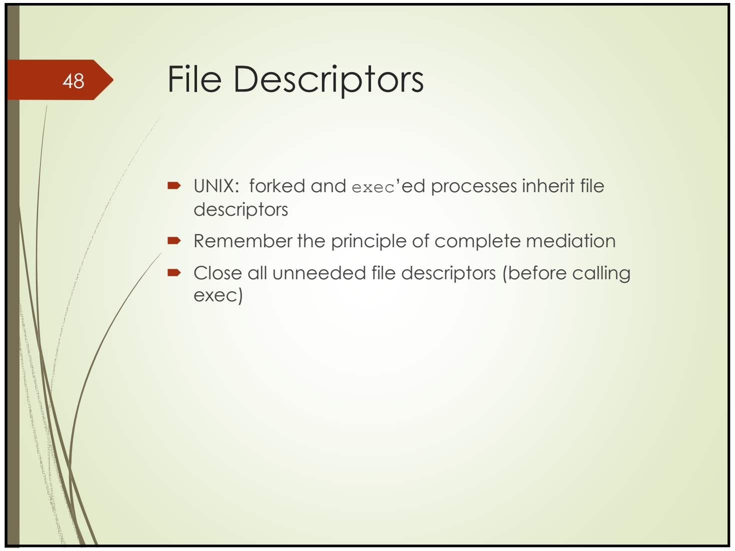
Question

- Rank the following calls in order of safety (high to low):
  - `system`
  - `execle`
  - `execv`
  - `execvp`

a) execv, execvp, execle, system

[✓ b) execle, execv, execvp, system]

c) execvp, execle, execv, system



48

File Descriptors

- UNIX: forked and `exec`’ed processes inherit file descriptors
- Remember the principle of complete mediation
- Close all unneeded file descriptors (before calling exec)